UNIVERSIDAD NACIONAL DE LA PLATA

TESINA DE LICENCIATURA

TÍTULO: Portal de comercialización online para la Economía Social y Solidaria Local
AUTORES: José Francisco Blanco, Iván Gabriel Gorosito
DIRECTORA: Claudia Queiruga
CODIRECTOR: Néstor Castro
ASESORES PROFESIONALES: Sergio Dumrauf, Belén Sendín
CARRERAS: Licenciatura en Informática y Licenciatura en Sistemas

Resumen

El presente trabajo de tesis de grado aborda la problemática de la comercialización de las ferias de la UNLP durante la pandemia COVID-19 y la urgencia de la digitalización de dicho proceso. Se trabajó junto a la comercializadora universitaria "La Justa", que nuclea productores de la agricultura familiar y artesanos del periurbano platense, en la co-construcción de un portal web que automatiza el proceso de comercialización y que además ofrece un perfil consumidor para la compra de productos. Este fue desarrollado íntegramente con tecnologías de código fuente abierto, su código está disponible en los repositorios de GitLab (https://gitlab.com/tesinafacultadinformatica/lajusta-backend y https://gitlab.com/tesinafacultadinformatica/lajusta-frontend) y se encuentra en producción desde febrero de 2021.

Palabras Clave

Economía Social y Solidaria, Co-construcción, Tecnologías Sociales, Desarrollo Web, Kotlin, Angular, Software Libre, Código Fuente Abierto.

Trabajos Realizados

Portal web destinado a la comercialización de productos de la agricultura familiar y de producción artesanal del periurbano platense, a partir de una base teórica sobre la Economía Social y Solidaria e Innovación Social.

Conclusiones

El desarrollo de esta tesina, nos permitió comprender un modelo de negocio distinto, el de la Economía Social y Solidaria, que nos planteó nuevos desafíos en cuanto al desarrollo del portal, dado que el diseño de las herramientas y la construcción de tecnologías son co-construidas permanentemente con los sujetos que las van a utilizar, de manera de realmente convertirse en tecnologías sociales que puedan ser apropiadas por quienes las van a utilizar. En este sentido, fue necesario comprender los engranajes que componen el desarrollo de software, desde otros puntos de vista, y tomar decisiones en diálogo permanente con nuestros directores y la organización adoptante del portal, "La Justa". Este proceso permitió entregar un portal de compra virtual desarrollado con tecnologías actuales, abiertas, libres, competitivas, que no tienen nada que envidiarle a otros sitios de compras del mercado, y concebida como una tecnología social. A su vez, este proyecto está abierto en GitLab, pudiéndose descargar, modificar o mejorar, agregando más funcionalidad.

Trabajos Futuros

Entre los trabajos futuros, identificamos:
- Desarrollar una aplicación móvil destinada a los consumidores a partir del back-end construido para el portal.
- Incorporar nuevas funcionalidades al portal como por ejemplo el pago digital y el perfil productor.

Fecha de la presentación: AGOSTO 2022

AGRADECIMIENTOS

Queremos agradecer a nuestras familias por brindarnos su apoyo en todo momento.
A Claudia Queiruga directora, Nestor Castro codirector y a los asesores profesionales del trabajo Sergio Dumrauf, Belén Sendín, Soledad Rial, por su predisposición y dedicación durante la implementación de esta tesina.

ÍNDICE

CAPÍTULO 1 - INTRODUCCIÓN

Este proyecto de tesina presenta el desarrollo de un portal web destinado a la comercialización de productos de la agricultura familiar y de producción artesanal del periurbano platense, a partir de una base teórica sobre la Economía Social y Solidaria e Innovación Social.

OBJETIVOS

Los objetivos generales de esta tesina de grado son:

- Contribuir al desarrollo de la comercializadora universitaria “La Justa” de la UNLP, con un portal de comercialización online de código abierto.
- Desarrollar un sistema web, para la comercialización online de la Economía Social y Solidaria de la región La Plata, que pueda ser adoptado por redes de productores que adhieran a este paradigma económico.

CONTEXTO Y MOTIVACIÓN

Esta tesina se inscribe en el campo del desarrollo de las tecnologías sociales para la comercialización solidaria en organizaciones de la Economía Social y Solidaria e Innovación Social.

De acuerdo a Deux y Vannini (2006:21), *“la visión socio-técnica [de la tecnología] considera que las sociedades son tecnológicamente construidas al mismo tiempo que las tecnologías son socialmente configuradas […] desde esta perspectiva […] los artefactos se co-construyen con sus usuarios, los productores con los clientes, las sociedades con las tecnologías que utilizan. Porque en el mismo proceso socio-técnico en el que se diseñan, producen y utilizan tecnologías se construyen relaciones sociales de producción, de trabajo, de comunicación, de convivencia, etc.”*

A partir del enfoque socio-técnico de la construcción de tecnologías surge la noción de tecnología social, como el modo de desarrollar e implementar tecnologías orientadas a la generación de dinámicas de inclusión social. Las tecnologías sociales se enmarcan en un proceso de co-construcción, no habiendo soluciones únicas para un mismo problema. Las tecnologías sociales se pueden definir entonces como “el conjunto de técnicas y procedimientos asociados a formas de organización colectiva que brindan soluciones para la inclusión social y la mejora en la calidad de vida” (Lassance, Antonio y otros, 2004, en: Deux y Vannini, 2006: 22). Se las denomina sistemas tecnológicos sociales (Thomas y Becerra, 2008, en: Deux y Vannini, 2006: 22) y no tecnologías puntuales, porque se requieren múltiples dimensiones para concebir en forma integral una inclusión de todos. Estos sistemas tecnológicos sociales implican el acceso a bienes y servicios a partir de la producción de bienes comunes, ampliando el espacio público al transformar en públicos o comunes amplios sectores de la economía.

La Economía Social y Solidaria se refiere a una forma de hacer economía, organizada asociativamente para la producción, distribución, circulación y consumo de bienes y servicios, y que pone en el centro el trabajo y las condiciones para la reproducción ampliada de la vida del conjunto de la sociedad. Las organizaciones, relaciones y prácticas de la economía social y solidaria construyen experiencias socio económicas cuyo propósito es la resolución de las necesidades de sus integrantes y sus comunidades, subordinando los ingresos monetarios al objetivo principal de mejorar las condiciones de vida de sus miembros y comunidades. Como explica J. L Coraggio: "Poner en el centro la reproducción ampliada de la vida humana no supone negar la necesidad de acumulación sino subordinarla a la reproducción de la vida, estableciendo otro tipo de unidad entre la producción (como medio) y la reproducción (como sentido)" (Coraggio J. L. , 2007).

El concepto de innovación social hace referencia al conocimiento incorporado en procesos y equipos que tiene por objetivo aumentar la efectividad de los procesos, servicios o productos destinados a la satisfacción de necesidades sociales (Dagnino y Gomes,0 2000 en: Deux y Vannini, 2006: 40-41). En este sentido "*las organizaciones de la Economía Social y Solidaria enfrentan el desafío de crear e implementar sus propias herramientas de gestión, afines a sus principios y valores, y al mismo tiempo, que les permitan sobrevivir en una economía predominantemente capitalista. La innovación en la gesti*on *se convierte en una estrategia indispensable para superar este desafío y trascender la tensión entre la viabilidad en los mercados y la conservación de su identidad cooperativa y autogestionaria"* (Deux y Vannini, 2006: 42).

Desde hace más de una década, distintas unidades académicas y extensionistas de la UNLP acompañan este esquema de comercialización mediante la instalación de ferias para diferentes organizaciones de productores locales, buscando principalmente ampliar canales de venta de los alimentos producidos por agricultores familiares de nuestro cordón florifrutihortícola platense, y así mejorar su estructura de ingresos. Estas experiencias han permitido construir un vínculo directo entre el productor y el consumidor, que dan un valor agregado al intercambio comercial permitiendo acceder a información de producción y de consumo, fortaleciendo el lazo social y construyendo un consumo consciente. Ejemplo de estas organizaciones de comercialización de la Economía Social y Solidaria apoyadas por la UNLP, son "Manos de la Tierra", "El Paseo", "La Veredita" y "Pueblo a Pueblo".

El aislamiento social, producto de la pandemia COVID-19, forzó a cerrar las ferias presenciales y por ende este importante canal de comercialización. Para dar respuestas a la situación emergente surge la comercializadora universitaria "La Justa", organizada en forma asociativa entre equipos técnicos y profesionales universitarios, productores de "Manos de la Tierra" y la Prosecretaría de Agricultura Familiar de la Facultad de Veterinaria, la Dirección de Fortalecimiento de la Economía Popular, Social y Solidaria, de la Prosecretaría de Políticas Sociales de la UNLP, junto con organizaciones locales culturales, sociales y políticas, que permitieron la realización de pedidos en forma on-line y un sistema de nodos

de entrega en los que se retiran y abonan los productos. Más allá del esfuerzo colectivo, las herramientas digitales escogidas resultaron precarias e ineficientes, limitando la capacidad de respuesta y llegada a los consumidores, la experiencia de compra, la cantidad de productos y productores que pueden participar y la información brindada.

En ese contexto, “La Justa” planteó la necesidad de desarrollar un portal de comercio electrónico para la Economía Social y Solidaria, que permita ampliar la experiencia de los consumidores y de los productores, constituyéndose en una tecnología social basada en software y tecnologías libres y abiertas orientada a la problemática planteada así como también con el intención de constituirse en una herramienta a ser socializada y compartida con otros grupos de la Economía Social y Solidaria, como las ferias de la UNLP, favoreciendo una apropiación colectiva de este portal. De acuerdo a Deux y Vannini, *“una forma innovadora de aplicar estas soluciones tecnológicas es compartiendo su desarrollo, uso, apropiación o ajuste entre organizaciones de un mismo rubro de actividad, como ser las Federaciones de cooperativas, en las que se pueden construir soluciones que resuelvan las necesidades de todos o muchos de sus integrantes”* (Deux y Vannini, 2006: 54).

En este sentido, la comercialización en tiempos de COVID-19, planteó una demanda de innovación social para dar soporte y crecimiento a las experiencias colectivas de comercialización locales.

RESULTADOS ESPERADOS

Como resultado de nuestra tesina, planteamos implementar un portal de comercio electrónico on-line que se incorpore a las comercializadora universitaria de la UNLP “La Justa”, accesible desde cualquier portal vía acceso web. Asimismo, atendiendo a la apropiación colectiva del portal y acorde al desarrollo de tecnologías sociales, el código fuente generado en este desarrollo quedará disponible en los repositorios de Gitlab (lajusta-backend[1] y lajusta-frontend[2]) para su uso y adaptación por otras redes/experiencias de la Economía Social y Solidaria.

En un sentido más amplio, con la realización de esta tesina se trata de fomentar que los profesionales del área informática comprendan e intervengan en las problemáticas locales y favorezcan el desarrollo socio-económico regional, entendiendo que las tecnologías digitales hoy constituyen un instrumento fundamental en el desarrollo de procesos socioeconómicos de inclusión social en nuestra región.

[1] https://gitlab.com/tesinafacultadinformatica/lajusta-backend
[2] https://gitlab.com/tesinafacultadinformatica/lajusta-frontend

CAPÍTULO 2 - ECONOMÍA SOCIAL Y SOLIDARIA

Desde una perspectiva histórica, la Economía Social y Solidaria (ESS) surge como una alternativa frente al modelo neoliberal, afianzándose como otro modo de entender la economía que prioriza el trabajo sobre el capital y al ser humano sobre el dinero, entendiéndose como una Economía del Trabajo. Es una economía que organiza su trabajo sin privilegios relativos a la posición jerárquica, la no explotación del trabajo infantil, la equidad de género, el cuidado del medio ambiente, la reducción de la intermediación para un precio justo, producción, circulación y consumo responsable y el desarrollo de finanzas solidarias; así como, la promoción de valores de solidaridad, cooperación, hermandad, respeto, autonomía y responsabilidad. En este sentido, las prácticas de la Economía Social y Solidaria fomentan la diversidad cultural, la armonía y la valoración de la naturaleza, la dignificación del trabajo, la igualdad, la justicia social, la ayuda mutua, y el comercio justo, equitativo y ético.

De todas las definiciones que existen de ESS nos gusta esta: *"El sistema de procesos de producción, distribución, circulación y consumo que, a través de principios, instituciones y prácticas, que en cada momento histórico, organizan las comunidades y sociedades para obtener las bases materiales de resolución de necesidades y deseos legítimos de todos sus miembros, actuales y de generaciones futuras, de modo, que permitan la reproducción y desarrollo de la vida, sosteniendo los equilibrios psíquicos, interpersonales, entre comunidades y con la naturaleza"* (Coraggio, 2007*).*

ACTORES DE LA ECONOMÍA SOCIAL Y SOLIDARIA

Se constituyen como actores de la ESS personas jurídicas o físicas, emprendedores vinculados con otros emprendedores de forma asociativa o a través del trabajo autogestivo, clubes de trueque, ferias populares, redes de comercio justo, asociaciones civiles, cooperativas, mutuales, fábricas recuperadas, Estados municipales / comunales y Estado provincial; que respondan en la práctica, a los valores y criterios de la ESS donde prevalece el bien común, la participación democrática, la solidaridad, el cuidado del medio ambiente y el trabajo colectivo en sus diversas formas.

PANDEMIA Y TECNOLOGÍAS SOCIALES

Como ha sido suficientemente demostrado, los y las productoras familiares tienen dificultades para lograr relaciones equitativas en los diferentes mercados por su menor poder de negociación. Lo anterior es consecuencia de: falta de acceso a la tierra en cuento una apropiación de adecuada escala productiva, limitado acceso a información y financiamiento, falta de infraestructura para acopio y acondicionamiento, alto costo del flete, bajos niveles de formalización en aspectos jurídicos, sanitarios y comerciales, escaso valor agregado de los productos, entre otros (Alcoba, Dumrauf et al. 2011, Caracciolo y Fontana, 2015).

Esta situación se manifiesta y varía según las regiones-especialización productiva y por cierto también según el tipo de reproducción socioeconómica de las unidades productivas. A modo de ejemplo, los productores que están insertos en cadenas agroindustriales como caña de azúcar, yerba mate, té, tabaco, algodón, peras y manzanas, vitivinicultura, etc. no pueden acceder a mercados locales o de proximidad, y menos a los consumidores directos (García, 2013), están muy condicionados por la agroindustria. Por su parte, los productores de hortalizas y los granjeros y/o agroindustria alimentaria artesanal que producen en fresco o procesado tienen la opción de la venta directa que elimina intermediaciones.

Respecto a la horticultura, con alta presencia de agricultores familiares, Viteri (2013) plantea la alta heterogeneidad de los canales que forman la trama comercial hortícola en la Argentina. Los miles de horticultores familiares de los numerosos cordones en las grandes ciudades, en general, intercambian productos con acopiadores zonales, y reciben menor precio por sus productos aunque tienen una mayor seguridad en el cobro. Otras razones que llevan al productor a vender su mercadería a intermediarios se relacionan al escaso conocimiento, falta de redes comerciales consolidadas, problemas de logística (sin transporte, geográficamente distante de los mercados físicos, bajo volumen productivo, etc.). Todas estas alternativas comerciales finalizan en los mercados mayoristas (Viteri, 2013).

Este trabajo de tesis se inscribe en los marcos teóricos de la Economía Social o Popular en transición o búsqueda de formas Solidarias, es decir en aquellos enfoques que valorizan el trabajo de los que viven de su trabajo, sean actividades unipersonales, familiares o asociativas, que bregan por una economía que pueda resolver las necesidades legítimas de todos sus habitantes cuidando a un tiempo el medio natural que las sostiene para avanzar hacia la reproducción ampliada de la vida (Coraggio, 2010).

Desde mediados de los años 90, vienen creciendo en el país los espacios comerciales alternativos (Caracciolo, Dumrauf y Moricz, 2012) en donde los productores se pueden relacionar en forma más directa con los consumidores y de esta manera obtener mejores precios, al igual que los consumidores, y mejor calidad. El más conocido de estos espacios es la Feria del Productor al Consumidor. En los últimos años, a lo anterior se ha sumado el incremento de la demanda de productos agroecológicos que se instala como una necesidad en una porción creciente de la población y que hace referencia a un concepto clave para el desarrollo de los pueblos como es la soberanía alimentaria[3].

Sin embargo, estos mercados alternativos están atravesados por tensiones internas y principalmente, por tensiones en relación con otros actores del sistema agroalimentario en particular y del contexto macro en general, porque se juegan relaciones de poder y/o conflictos de intereses (Alcoba, Dumrauf et al., 2011). Cada tipo de espacio comercial tiene

[3] *"La soberanía alimentaria es el derecho de los pueblos a definir sus propias políticas y estrategias sustentables de producción, distribución y consumo de alimentos que garanticen el derecho a la alimentación para toda la población, con base en la pequeña y mediana producción, respetando sus propias culturas y la diversidad de los modos campesinos, pesqueros e indígenas de producción agropecuaria, de comercialización y de gestión de los espacios rurales, en los cuales la mujer desempeña un papel fundamental"* (Conclusiones del foro mundial sobre soberanía alimentaria. La Habana, Cuba, Septiembre 2001)

actores y relaciones sociales, redes, modalidades de gestión, formas jurídicas, distancia con los productores, criterios para la fijación de los precios, normativas, y articulaciones entre sus objetivos económicos y sociopolíticos que son similares en algunos aspectos y diferentes en otros. Por tales motivos, los procesos de reproducción ampliada o de acumulación solidaria (Caracciolo, 2018) de estos espacios, cada día con más productores y más consumidores que satisfacen sus necesidades en los mismos, resultan muy complejos.

En La Plata, a partir del Aislamiento Social Preventivo y Obligatorio (ASPO) que inicia el 20/03/2020, las Ferias ligadas a la Universidad Nacional de La Plata fueron cerradas, al igual que la mayoría de las otras ferias de la ciudad, quedando los productores sin estos espacios de comercialización de sus productos, por lo tanto, desde el Equipo de Dirección de Economía Popular, Social y Solidaria de la Prosecretaría de Políticas Sociales de la UNLP, y desde la Prosecretaría de Agricultura Familiar de la Facultad de Ciencias Veterinarias, se comenzó a gestionar una salida.

Antes de la pandemia, las ferias eran de venta directa, esto es, del productor al consumidor. Al cerrar esos espacios, los equipos técnicos se ponen rápidamente en marcha y desarrollaron un sistema de intermediación solidaria, es decir mediar sin objeto de lucro entre la producción y el consumo local. Se entiende por organizaciones de intermediación solidaria a grupos de personas dedicadas a la distribución de productos alimenticios y no alimenticios, producidos y comercializados bajo determinados criterios explícitos. En relación a la producción, estos criterios giran en torno a los valores cooperativos y la producción sostenible tanto social como ambiental. En cuanto a la distribución, las comercializadoras realizan el trabajo de acercar el producto al consumidor bajo criterios de precio justo. Otra característica de las organizaciones de intermediación solidaria es el carácter cooperativo que asumen como forma de organización interna, si bien esto no implica necesariamente que asuman la forma jurídica de cooperativa. Por último, un último rasgo de las organizaciones estudiadas es que el trabajo de distribución es acompañado por un intento de visibilizar las consecuencias del consumo más allá del acto del intercambio (Mosse, 2019).

Es así como se creo una comercializadora de Intermediación Solidaria, que llamaron “La Justa”, integrada por los productos de dos ferias de la Red de Ferias de la UNLP: “Manos de la Tierra” y “La Veredita”, como salida superadora ante el cierre presencial de los espacios de comercialización directa de esos proyectos.

COMERCIALIZADORA UNIVERSITARIA “LA JUSTA”

“La Justa” es una comercializadora de la economía social que, desde la Universidad Nacional de La Plata, y en red con organizaciones sociales, comunitarias, políticas y culturales, promociona una compra de proximidad de alimentos y otros bienes elaborados artesanalmente por productores y productoras de la economía popular y la agricultura familiar, intermediando solidariamente entre el consumo y la producción local.

El esquema de trabajo de la comercializadora se inicia con la difusión vía redes sociales de la oferta de alimentos disponibles por parte de los productores familiares, se recepcionan y confirman los pedidos (tipo de producto, cantidad y precio) y se entregan en los 13 nodos de retiro de la red, localizados en distintos puntos de la ciudad de La Plata y de Berisso. En términos generales, la propuesta de la "La Justa" es que esta trama de organizaciones que tienen un rol en la entrega de los productos encargados online, pudieran generar actividades en torno a los ejes que atraviesan al proyecto: agroecología, soberanía alimentaria, agricultura familiar y economía social. El dispositivo de trabajo, posibilita conocer a los productores en sus ámbitos de producción, transformación y acondicionamiento de los bienes producidos. Los desafíos están en relación a mejorar los procesos de logística y distribución, mejorar la comunicación con consumidores y fortalecer y establecer nuevas alianzas con comercializadoras y productores, que permitan ampliar el volumen y los canales de venta.

(Véase: https://www.facebook.com/La-Justa-comercializadora-105707981222282 y https://www.instagram.com/lajusta.comercializadora/).

"La Justa" involucra unas 200 familias productoras, organizadas en torno a 15 organizaciones. El equipo de trabajo lo conforman unas 40 personas que se organizan en equipos de trabajo: acompañamiento técnico a productores, comunicación, logística y coordinación general. Durante 2020 se vendieron 86 toneladas de alimentos y en 2021, 61 toneladas.

SITUACIÓN PREVIA DEL PORTAL DE VENTA ON-LINE

Como hemos adelantado más arriba, la experiencia de construcción del portal de venta on-line nació en marzo de 2020, en el marco del ASPO y la pandemia COVID-19, para dar continuidad a las ferias de venta directa del productor al consumidor, y otros proyectos productivos que se acompañan desde distintos equipos técnicos de UNLP como "Manos de la Tierra" y mercado popular "La Veredita".

Ferias Presenciales

Las ferias presenciales que convergen en "La Justa" fueron primordialmente: "Manos de la Tierra", que funciona los días miércoles en la facultad de Ciencias Agrarias y Forestales, y los días viernes en la facultad de Ingeniería, y "La Veredita", que funciona los días lunes en la facultad de Artes y los días jueves en la facultad de Trabajo Social.

Barros (2015) describe de esta manera a la organización "Manos de la tierra": *"La experiencia 'Manos de la tierra' es un caso de circuito corto de alcance local que implica la venta directa (sin intermediación) de productos de la agricultura familiar y de la economía social y solidaria, entre productores del área del Gran La Plata y los consumidores. Se trata de agricultores/as familiares rurales y peri-urbanos con pequeñas superficies arrendadas. Comenzó a desarrollarse en octubre del año 2008, en el marco de las Jornadas de Extensión de la Facultad de Ciencias Agrarias y Forestales. Brevemente señalar, que un*

contexto de marcada presencia en la intermediación entre ambos actores -productores y consumidores- con una elevada apropiación del excedente en la comercialización de hortalizas, describe con fuerza la problemática del sector. La venta a "culata de camión" con las desventajas que esto implica -asimetrías de información respecto de los precios y cobro diferido del producto- traduce tanto una posición estructural, como la complejidad de conmover un esquema de resolución que ubica al productor/a como eslabón primario de una larga cadena de producción, distribución y consumo. Sujeción que redunda en una posición sumamente desventajosa para el productor/a en tanto la obtención de ingreso monetario percibido. Los pequeños productores se ven obligados a vender a un intermediario que les paga a precios inferiores la mercancía. Las alternativas de diferenciación de la producción son escasas, por lo que lo realizado por los productores familiares de tipo convencional debe competir con los grandes productores. Su baja escala los condena a una situación de marginación constante, por lo que de existir otros modelos productivos de bajos insumos, así como estrategias diferenciales de comercialización los posicionará mejor en la cadena que abastece de alimentos frescos al área metropolitana. La Feria "Manos de la Tierra" surge en este contexto, posicionándose como una alternativa para que productores familiares puedan vender su producción a un precio más justo".

Por su parte, el mercado popular "La Veredita" de la facultad de Trabajo Social, surge en el año 2014, en el marco del Programa de Extensión 'Políticas públicas y nuevas ruralidades: Un aporte para el fortalecimiento de organizaciones sociales y sectores populares vinculados a la gestión rural' de la Facultad de Trabajo Social de la UNLP. . El proyecto toma las experiencias previas de mercados populares como "Manos de la Tierra" y "Mercado de la Ribera", y se propone "*[...] continuar aportando a la construcción de estrategias de intervención, desde equipos de trabajo multidisciplinarios, favoreciendo los canales de comercialización existentes entre productores familiares y consumidores. Estas acciones, son pensadas en el marco de precios justos y solidarios, promoviendo el consumo responsable de alimentos que son producidos desde la perspectiva de la transición agroecológica, por grupos de productores familiares de la zona rural y periurbana de La Plata y Berisso"* (Bulich, et. al. 2017).

Primer Acercamiento a la Digitalización: Formularios de Google

"La Justa" nace desde su concepción con una impronta fuertemente digital, de venta online, pero con un componente territorial y material que es la entrega en nodos barriales. Para la comercialización digital se utilizó, en un principio, la herramienta que se encontró más a mano: "Google Forms". Se diseñó un formulario en el que volcaron los principales productos, con sus características, precio y fotos, y selección de retiro en un nodo (en un principio 4 pero hacia el segundo semestre de 2020 ya eran 10), que resuelve provisoriamente la construcción de la oferta (mostrar los productos disponibles de las familias productoras) y la demanda (selección de productos y nodos por parte de los consumidores). Esta decisión vino aparejada de múltiples inconvenientes para el trabajo interno de la comercializadora.

Por una parte, no permitía informar stocks, con lo cual los días de apertura de pedidos fue necesario disponer de una parte del equipo, controlando el formulario, en turnos, para no superar los stocks disponibles y "quitar" la opción de compra del producto cuando se superaba el stock. Es decir, un control absolutamente manual, fuertemente propenso a errores. Por otra parte, solía suceder que algunos pedidos no impactaban correctamente en la base de datos, y para evitar errores, el equipo de trabajo se ocupaba, varios días a la semana, de tomar la planilla de datos que arroja el formulario y confirmar los pedidos por whatsapp con cada cliente, para lo cual fue necesario trabajar muchísimo la planilla, necesitando de muchas columnas y dificultando el cálculo de los totales. Esta actividad quitaba mucho tiempo del trabajo del equipo, que en otros momentos hubieran sido horas dedicadas al acompañamiento técnico y organizativo de los grupos de productores y otras tareas relativas al fortalecimiento de los mismos.

Digitalización de Procesos

Más allá del esfuerzo colectivo, las herramientas digitales escogidas resultaron precarias, limitando la capacidad de respuesta y llegada a los consumidores, la experiencia de compra, la cantidad de productos y productores que pueden participar y la información brindada. Una de las consecuencias que apareció claramente en esta primera aproximación a la digitalización fue el desborde, por ejemplo ante la dificultad de controlar automáticamente la disponibilidad de productos, la dificultad de dar a conocer los productos ofrecidos y los productores, cuya consecuencia fue reducir la oferta de productos.

En ese contexto "La Justa" plantea la necesidad de desarrollar un portal de comercio electrónico, para la ESS, que permita ampliar la experiencia de los consumidores y de los productores, constituyéndose en una tecnología social, basada en software y tecnologías libres y abiertas, orientada a la problemática planteada así como también con el intención de constituirse en una herramienta a ser socializada y compartida con otros grupos de la ESS, como las ferias de la UNLP, favoreciendo una apropiación colectiva de este portal. De acuerdo a Deux y Vannini, *"una forma innovadora de aplicar estas soluciones tecnológicas es compartiendo su desarrollo, uso, apropiación o ajuste entre organizaciones de un mismo rubro de actividad, como ser las Federaciones de cooperativas, en las que se pueden construir soluciones que resuelvan las necesidades de todos o muchos de sus integrantes"* (Deux y Vannini, 2006: 54).

En este sentido, la comercialización en tiempos de COVID-19, planteó una demanda de innovación social para dar soporte y crecimiento a las experiencias colectivas de comercialización locales. Pero al pensar en una herramienta específica para la economía social y en particular para "La Justa", surge la creación de un portal que vaya más allá de la comercialización, y que abarque otras dimensiones que son parte del proyecto como son la promoción de otras formas de producción, intercambio y consumo, con lo cual las herramientas a ser creadas debían contemplar no solo la venta de productos, sino también poder acortar distancias entre productores y consumidores, poder contar las historias detrás

de quienes producen los alimentos y artesanías, difundir contenido como recetas o videos sobre la producción, entre otros, con la intención de sensibilizar respecto a la agricultura familiar, la agroecología, y la soberanía alimentaria.

CAPÍTULO 3 - ESTADO DEL ARTE

A continuación se describirán algunos antecedentes de proyectos de digitalización de espacios de intermediación solidaria que han cobrado un destacado relieve en los últimos años, sobre todo a partir de la emergencia del ASPO producto de la pandemia COVID-2019.

PROYECTO CHASQUI - NACIONAL

El proyecto Chasqui[4] es una plataforma digital desarrollada por la Universidad Nacional de Quilmes, basada en tecnologías de código fuente abierto, que busca fomentar la ESS, dándole un lugar a las familias de productores locales la oportunidad de exponer sus productos, entre los cuales se destacan, en su mayoría, los del rubro alimenticio, y de esta manera tener más llegada o un mejor alcance dentro de los consumidores.

El proyecto fue llevado a cabo en conjunto con distintas universidades, con un enfoque multidisciplinario, sumándose también cooperativas de desarrollo de software e iniciativas de intermediación solidaria. Además contó con apoyos de los Ministerios Nacionales de Desarrollo Social, Educación, y de Ciencia Tecnología e Innovación.

La plataforma ofrece la posibilidad a las usuarias y usuarios de comprar cualquier producto de los expuestos por las distintas familias de productores y su retiro puede ser: en el lugar, en algún nodo, o bien con entrega a domicilio. Las figuras 1, 2 y 3 muestran algunas pantallas de la plataforma Chasqui, en las que es posible identificar la forma de interacción.

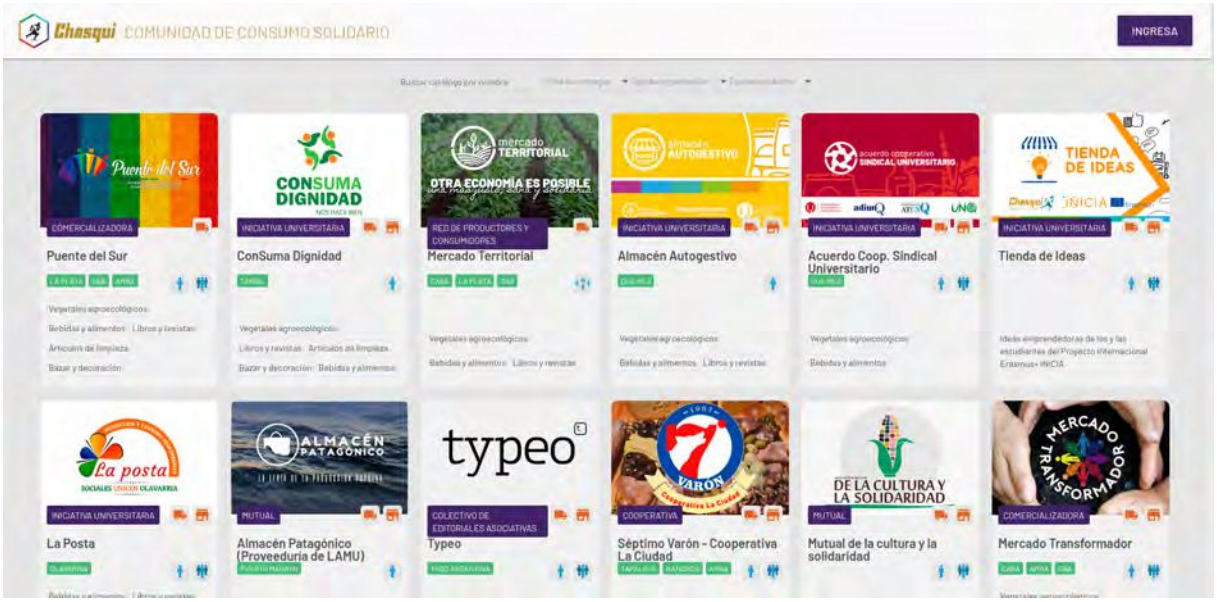


Figura 1: Chasqui comunidad de consumo solidario

Del mismo modo las usuarias y usuarios pueden realizar las compras en la plataforma de manera individual, o a través de un grupo o cooperativa (nodos solidarios), como también hacer el seguimiento de una compra.

Para los productores ofrece una página personalizada donde pueden agregar información acerca de la producción que tienen o sobre la familia que se encarga de llevarla a cabo, un

[4] Sitio oficial del proyecto Chasqui: https://proyectochasqui.com/#/

banco de recursos compartidos entre todos los productores, y asistencia de un equipo para cualquier consulta relacionada a la plataforma.

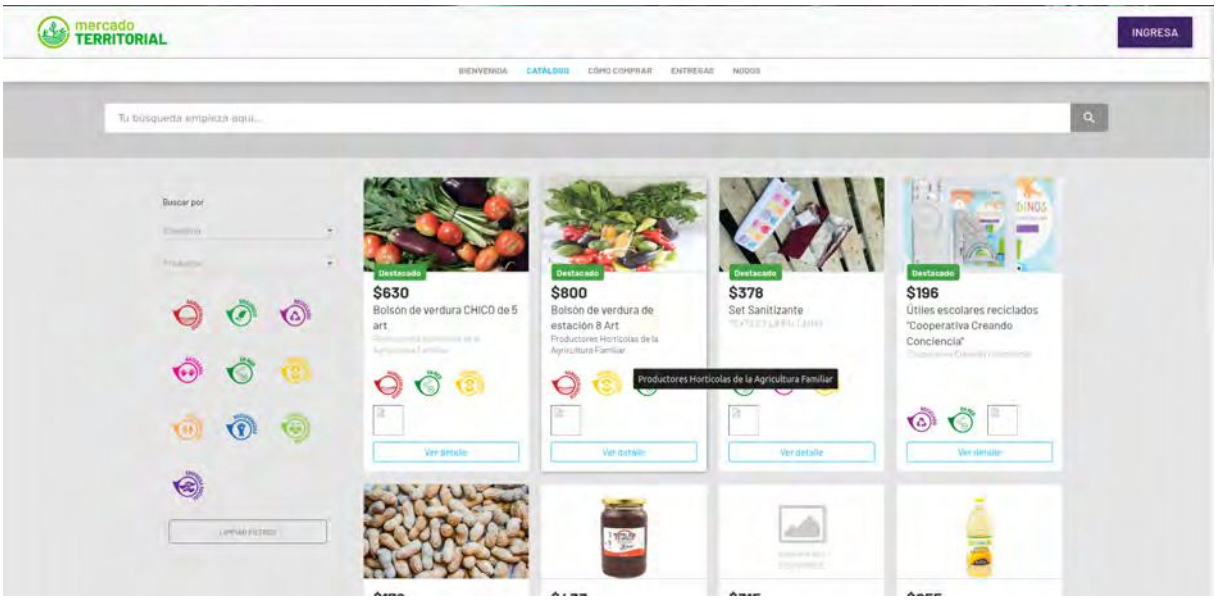


Figura 2: Mercado Territorial

Como se puede apreciar en las pantallas de la plataforma Chasqui, varios de las funcionalidades ofrecidas coinciden con las de “La Justa”: manejo de usuarios, retiro en puntos de entrega, compras por medio del portal, envío de notificaciones vía correo electrónico y otros canales de comunicación. En nuestro sistema se agrega, a diferencia de Chasqui, las rondas y la administración de las mismas, los usuarios disponen de un tiempo determinado para realizar las compras, una vez pasado dicho tiempo se procede a armar los pedidos. En este sentido, podemos afirmar que el portal de “La Justa” responde de una manera apropiada a la metodología de trabajo adoptada por la comercializadora, atendiendo a las particularidades de comercialización del grupo de productores.

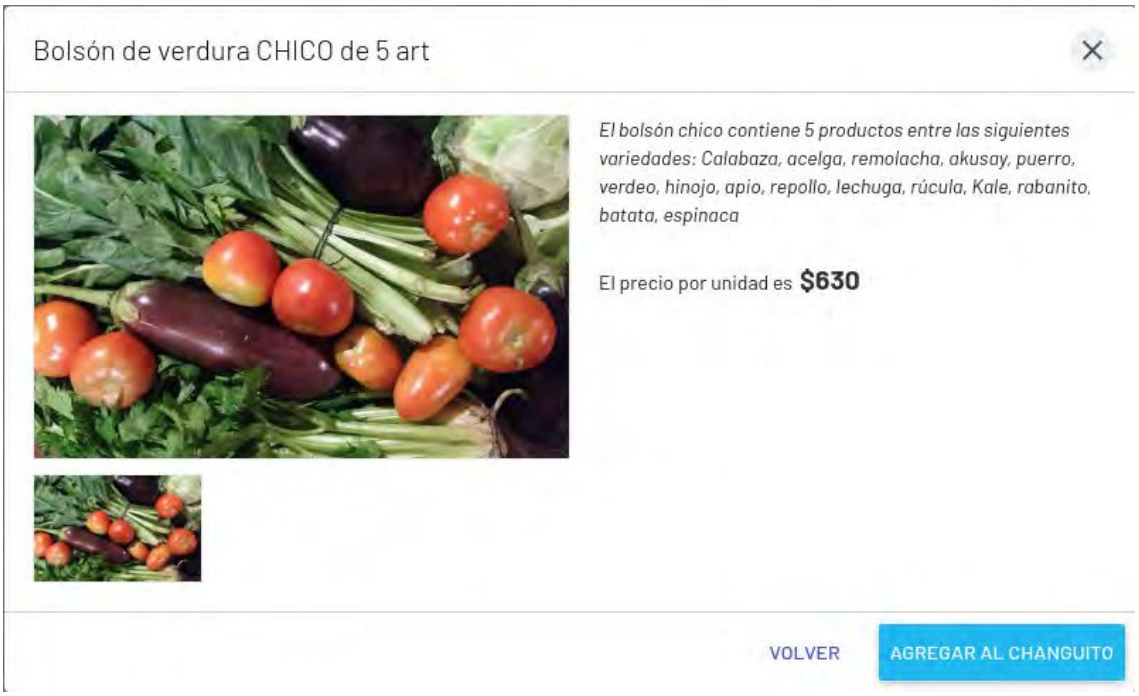


Figura 3: Chasqui producto

En cuanto a las tecnologías utilizadas para el desarrollo, podemos notar que la plataforma Chasqui está desarrollada con Angular 1.6 para el front end, NGINX como servidor y una base de datos MySQL, todas de código fuente abierto. Las antes mencionadas son similitudes tecnológicas que comparten el sitio de Chasqui con el portal de La Justa, la gran diferencia entre las dos radica en cuanto al servicio que prestan a sus productores y consumidores, dado que Chasqui favorece más la autogestión, mientras que el portal de La Justa es una herramienta diseñada para facilitar los tiempos y procesos de su equipo.

GOBIERNO DE MENDOZA - SITIO OFICIAL

Si bien en este proyecto no estamos hablando de un sistema web realizado directamente por una cooperativa o grupo social, cuando hablamos de ESS podemos hacer referencia a los distintos grupos de productores formados en la provincia de Mendoza que publicitan por medio de su página web oficial[5]. La intención de estos grupos es acercar a los consumidores los productos de familias agroproductoras de la provincia de Mendoza. Técnicamente es un sitio web estático en el que se publicitan los productos y la referencia de los productores, los pedidos se manejan telefónicamente y, en algunos casos ofrecen “delivery” mientras que otros la modalidad “take away” en distintos puntos.

En contraposición, el sistema de comercialización de “La Justa” ofrece un portal de compra vía web que facilita el acceso a sus usuarias y usuarios y, a las productoras y productores de las distintas familias productoras, a la vez que se constituye como una herramienta fundamental para la gestión del equipo de la comercializadora, facilitando la generación de información y reportes al equipo de trabajo, favoreciendo la organización y la optimización del tiempo, permitiendo dedicar ese tiempo en otras tareas. Toda la actividad del portal, se realiza de manera autogestionable, donde el usuario administrador puede actualizar y modificar el portal por medio de un sistema de backoffice. Si bien la diferencia entre el sitio oficial del gobierno de Mendoza (figura 4) y el portal de comercialización de “La Justa”, es muy grande, este es un buen ejemplo de cómo el fin o la visión detrás de un desarrollo tecnológico, puede trazar cierta afinidad.

[5] https://www.mendoza.gov.ar/catalogoeconomiasocial/comercializadoras/

Figura 4: Gobierno Mendoza

PROYECTO INTERNACIONAL O REGIONAL

Dentro de esta categoría podemos señalar las múltiples plataformas de venta online que hoy existen, así como comparamos a “La Justa” con el sitio del gobierno de Mendoza, dado que ambos comparten ideologías de la ESS, no podemos dejar de comparar el sistema web de “La Justa” con cualquier E-Commerce existentes en la actualidad, entre ellos Mercadolibre, OLX, o cualquier plataforma donde los productores puedan registrarse y vender sus productos. Es decir, desde el punto de vista técnico podrían relacionarse con “La Justa”, dado que estas plataformas adoptan tecnologías y metodologías de desarrollo similares, entre ellas podemos señalar, los motores de bases de datos, los servidores, el framework para frontend o backend, metodologías ágiles de desarrollo, cloud computing, serverless, front y back separados y microservicios. Sin embargo, estas mismas plataformas no se pueden comparar desde el punto de vista de su base ética, como se denomina en economía social. El mercado netamente capitalista no es neutral, tiene una ética: la de reproducción ampliada del capital privado, de la maximización individual de la ganancia por sobre cualquier otra variable, mientras la economía social tiene una ética que implica la reproducción ampliada de la vida, es decir que obtener ingresos o cualquier otra actividad se subordina a una responsabilidad colectiva por la vida de todos. Esa es la principal característica de los proyectos que se enmarcan en la economía social, que en general las

decisiones sobre qué/cómo producir, comercializar y consumir no están sujetos a lo que decide unilateralmente el “mercado”, ni la relación “oferta y demanda”, sino que se sujetan a las necesidades vitales de la población, y se legitiman socialmente.

CAPÍTULO 4 - DISEÑO DEL PORTAL DE COMERCIALIZACIÓN ON-LINE PARA LA ESS LOCAL

El diseño del portal de comercialización on-line lo organizamos en dos partes, que se fueron desarrollando en forma simultánea. En la primera se trabajó en conocer las necesidades de los distintos grupos de productores de la agricultura familiar y pequeños productores de alimentos del periurbano platense, nucleados en la comercializadora universitaria “La Justa” y del equipo de trabajo que lleva adelante la comercialización, y a partir de este diálogo construímos el problema que finalmente abordamos. Para ello realizamos reuniones periódicas, en las que compartimos entregables de los diseños y maquetados, con la intención de ir adecuando la solución propuesta a las demandas que iban surgiendo. En simultáneo se trabajó en el diseño del modelo de objetos del dominio de la aplicación que también se fue compartiendo con el equipo de trabajo de “La Justa” y en la definición de la arquitectura que finalmente adoptaría el portal. En este sentido, se adoptó una arquitectura RestFul con Spring Boot (Craig Walls, 2016) en Kotlin (Dmitry Jamerov y Svetlana Isakova, 2017) y SPA (Single Page Application) (Richardson L. et al, 2013) (One framework. Mobile & desktop, sf) en Angular facilitando su acceso desde cualquier dispositivo que posea un navegador web.

En una primera instancia realizamos reuniones frecuentes, con el objetivo de conocer las necesidades del equipo de “La Justa”, a la vez que analizamos los formularios que utilizaban en el trabajo cotidiano para poder comprender el problema y su forma de trabajo, y así adaptar el portal a sus necesidades.

ANÁLISIS DEL PROBLEMA

Desde un primer momento, realizamos reuniones frecuentes con el equipo de “La Justa” para entender el manejo y formas de comercialización: cómo es la toma de los pedidos de los consumidores mediada por tecnologías digitales, específicamente a través de mensajes de whatsapp y formularios de Google, cómo se distribuyen y administran los pedidos en cada nodo de entrega. El primer paso fue entender cómo era el flujo de pedidos y entregas de productos. Las primeras preguntas que surgieron fueron: ¿cuándo se hacen los pedidos?, ¿pueden hacerse en cualquier momento?, ¿cuándo el consumidor retira su pedido, en qué nodo de entrega?, ¿cómo se acuerda la entrega?

La dinámica de los pedidos funciona por rondas que se activan cada cierto lapso de tiempo (normalmente cada 2 semanas) y en las que se habilita a los consumidores a hacer sus pedidos. Los mismos serán retirados en un nodo de entrega, que elige el consumidor, con una diferencia de días respecto del pedido. Por ejemplo: la ronda se abre para los pedidos de viernes a martes, para entregarse el viernes siguiente (misma semana del cierre de la ronda) en los distintos nodos de entrega. Los nodos de entrega se encuentran localizados en diferentes barrios de la ciudad de La Plata (casco urbano, Villa Elisa, Arturo Segui, Gorina, Abasto, Tolosa) y en Berisso, garantizando de esta manera una cobertura adecuada.

¿Cómo organizamos el trabajo con el equipo de “La Justa”?

En un primer momento definimos entregables cada 20 días, para ir compartiendo los avances e ideas que iban surgiendo. Esta interacción, la hicimos de manera virtual, mediante herramientas de videoconferencia (google meet) y también usamos una pizarra colaborativa (Trello[6]).

Análisis del problema: un primer enfoque

Antes de comenzar el proceso de desarrollo, realizamos varias reuniones con el equipo de “La Justa” para comprender su manera de trabajar: cómo es que disponen de la mercadería, de qué manera los productores les hacían llegar los productos, es decir, si los productores les envían los productos o los retiran de las quintas o establecimientos de fabricación, si la producción es constante o es por períodos de tiempo (ejemplo el cultivo de las verduras), cómo es que toman los pedidos, y cómo es la distribución de los pedidos, etc. En estas reuniones nos enteramos, por ejemplo que, algunos de los productos que se comercializan, como carne de cerdo y pollo, necesitan frío (de heladera), que los pedidos los tomaban no solo por formularios de Google, sino también por whatsapp y hasta por correo electrónico y que las entregas de los pedidos las hacían en puntos de retiro o nodos distribuídos en diferentes barrios de La Plata y no todos cuentan con heladeras. También, nos hicieron saber que la venta no es constante, que realizan rondas de venta, en donde levantan los pedidos a partir de los productos que los productores pueden ofrecer y que en una semana registran en una planilla de cálculo la mercadería pedida por cada consumidor y el nodo de retiro elegido y a la semana siguiente se envía dicho pedido al nodo de retiro. Asimismo mantenían múltiples planillas separadas, entre ellas: las que contenían información sobre qué productos se deben enviar a cada nodo de retiro, planillas de balance de las ventas de la ronda, de pago de productores y de balance general.

A partir de esta interacción, fuimos construyendo un modelo de comercialización que atendió el requerimiento de rondas que se activan y desactivan, que ofrece productos para la compra con ciertas restricciones, tanto en cantidades disponibles como en condiciones de conservación, por ejemplo si requerían frío o no y, nodos de entrega con características propias (por ej. si disponen de heladeras).

También, en estas reuniones, se identificaron los diferentes usuarios que tendría el portal, en principio llegamos a la conclusión que el sistema iba a constar solo de dos tipos de usuarios: consumidores y administradores (que también pueden realizar pedidos). El rol de productor si bien se modeló, no es un usuario en esta implementación. En un futuro, los productores podrían ser los encargados de realizar la carga de la mercadería que pueden proveer, es decir cargar sus propios productos. En este punto es relevante mencionar que se analizaron, mediante historias de usuarios, quiénes serían usuarios, qué acciones se les habilitaría, es decir qué permisos tendrán sobre la información. A partir de lo descrito anteriormente, se

[6] https://trello.com

analizó la necesidad de generar automáticamente las múltiples planillas necesarias para la comercialización. A su vez, se dejó el modelo preparado para incorporar nueva funcionalidad, entre ellas podemos mencionar, las entregas a domicilio, descuentos y formas de pago.

En síntesis, de este análisis se distinguen dos tipos de usuarios: consumidores y administradores. Los usuarios consumidores pueden comprar productos, ver el historial de sus compras, ver dónde se encuentran los nodos de retiro e informarse acerca de los productores y productos, por ejemplo conocer el origen de los productos que compran, qué establecimiento o productor lo produce y su historia, qué otros productos produce, la forma de producción, etc. Los usuarios administradores cuentan, entre sus principales funciones:

- Cargar y configurar la información de los productos que ofrece la comercializadora.
- Llevar un control/balance sobre las ventas, gastos y stocks disponibles.
- Acceder a la información de los consumidores registrados y conocer qué productos consumen habitualmente.
- Cargar, configurar y dar de baja, la información de los nodos de entrega de los productos.
- Cargar la información de los productores.
- Configurar las rondas: días en que se habilitarán las compras y días de entrega en los nodos.

Ambos perfiles de usuarios deberán registrarse en el portal. Esto permitirá contar con un registro preciso de compras, información sistematizada de contacto con los consumidores y productores. El acceso de los usuarios es mediante un correo y contraseña.

EL PUNTO DE PARTIDA: EL MODELADO DE OBJETOS DEL DOMINIO

En este apartado se describe el modelo de dominio construído para el desarrollo del proyecto. Como es sabido, un modelo de objetos del dominio es una representación gráfica de todos los datos que se van a almacenar en el sistema y también las relaciones que existen entre ellos. Se identifican las clases conceptuales, significativas en un dominio de problema, enfocándose en las abstracciones relevantes, en el vocabulario e información del dominio, en síntesis el modelo de objetos del dominio es el artefacto clave del análisis orientado a objetos (García-Peñalvo y García-Holgado, 2018).

La técnica de modelado de objetos se basa en un conjunto de conceptos del paradigma de orientación a objetos, y en una notación gráfica útil en las primeras etapas del desarrollo de software, entre las más usadas se encuentra el lenguaje de modelado unificado, UML, que permite representar de manera visual los objetos, estados y procesos dentro de un sistema. La esencia del modelado de objetos de dominio es la identificación y organización de conceptos (objetos) del dominio de la aplicación. Una vez que se han identificado, organizado y comprendido los conceptos inherentes de la aplicación se pueden tratar en forma efectiva los detalles de las estructuras de datos y de las funciones. De esta manera es posible contar con un panorama completo del problema a resolver, en términos de objetos y

relaciones entre ellos, así como encontrar, reconocer y re-diseñar aspectos que surjan de su análisis.

En el diseño del modelo de objetos del dominio del portal de “La Justa” partimos de un diagrama de clases en el que se describe la relación entre los objetos identificados como relevantes para el problema en cuestión y que se presentará en el siguiente apartado.

Primera Propuesta de Modelo de Objetos del Dominio

Después de las primeras dos reuniones con el equipo de “La Justa” donde nos explicaron los problemas principales y algunas de las características de lo que esperaban para su portal, arribamos al modelo de objetos de la figura 5.

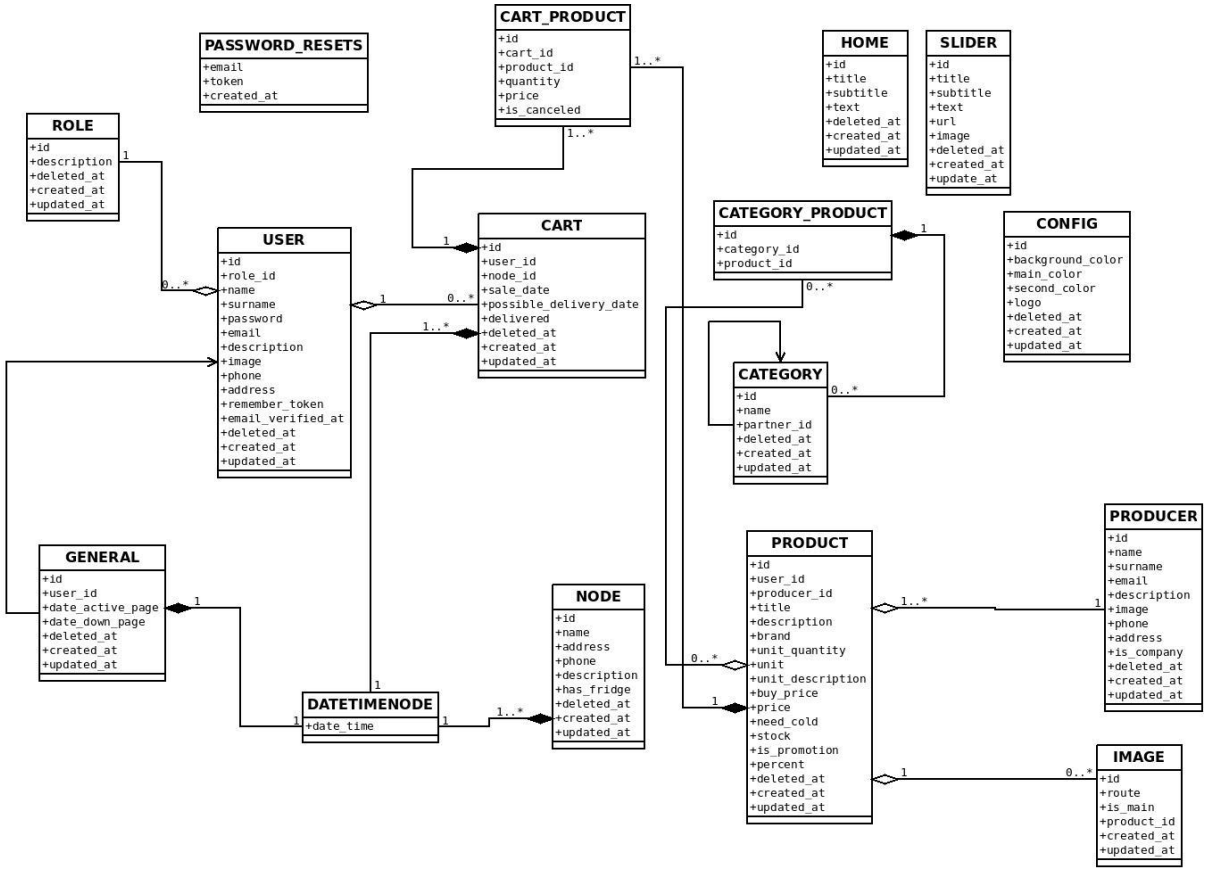


Figura 5: Primer Modelo de Objetos

En la figura 5, se describen los principales objetos del sistema y sus relaciones, en una primera versión definimos la clase USER que hace referencia a los datos del usuario y es la parte central del sistema. En ésta se definen los usuarios que van a poder utilizar los distintos tipos de roles, objeto ROLE (el administrador y el consumidor). El usuario administrador, cuenta con permisos para acceder al backoffice de la aplicación, pudiendo cargar fotos de los productos, así como también crearlos; crear o subir fotos de los productores, cargar información de los productos (como precio y cantidad disponible), configurar las rondas, crear nodos, administrar los banners, y poder acceder a los pedidos pudiendo cancelarlos o eliminarlos, mientras que el consumidor, puede realizar compras eligiendo su lugar de retiro y ver el historial de sus compras realizadas. La clase ROLE es

para poder realizar compras dentro de la aplicación (como usuario consumidor o administrador), pudiendo realizar los pedidos.

Podemos observar, además, que los objetos USER se asocian con una colección de tipo CART, que representa los pedidos que se realizan durante los distintos objetos GENERAL, nombre que luego evolucionó a RONDAS. Los usuarios del sistema pueden realizar la cantidad de compras que ellos crean necesarias en una misma RONDA sin ninguna restricción. Desde el sistema cuando se confeccionan los reportes de los pedidos confirmados, que serán enviados a los nodos de entrega, se agrupan los diferentes pedidos realizados por usuario consumidor en una ronda determinada y se ordenan alfabéticamente; por ejemplo si un consumidor pide un producto al principio del inicio de la RONDA, luego hace otro pedido de otros tres productos más y antes de finalizar la RONDA realiza otro pedido más de un producto, y elige siempre el mismo nodo de retiro, cuando se acerca al nodo encontrará preparado su pedido con los cinco productos, esto permite que el consumidor puede confeccionar su pedido en momentos diferentes mientras la ronda esté activa o abierta, lo mismo podría ocurrir si el consumidor desea eliminar algún producto del pedido. Esta dinámica aporta flexibilidad a la realización del pedido, siendo valorada por los consumidores. El objeto CART posee una colección de CART_PRODUCT, que representa los productos pedidos, esto se modeló de esta manera dado que un PRODUCT representa un producto que se ofrece para la venta y su stock se modifica dinámicamente, lo mismo que su precio, entre otras cuestiones, entonces para poder modelar esa demanda de manera que sea amigable a la programación orientada a objetos creamos la clase CART_PRODUCT que agrega un poco de metadata y además mantiene el precio del momento del pedido, dado que sabemos que en Argentina los precios varian a los pocos días, queremos que el consumidor se quede tranquilo que el precio se mantuvo igual como el dia que realizó el pedido.

Si bien la relación entre el objeto USER y GENERAL se removió de la base de datos, hoy en día sabemos que Usuarios realizaron compras en un General por medio del Carrito de compras. De todas maneras esta relación se mantuvo en el modelo final a modo de auditoría, lo que hacemos es traernos al GENERAL el user_id del usuario que creó dicho GENERAL, de esta manera podemos saber y reclamar a quien lo haya creado por cualquier inconveniente.

Sin ahondar en demasiados detalles sobre el objeto PRODUCT sabemos que solamente una familia de PRODUCER es la encargada de producir un producto en particular, pero una familia de PRODUCER puede producir más de un PRODUCT y los mismos están relacionados a uno o varios objetos CATEGORY y que pueden tener asociada más de una IMAGEN si así lo desean.

El objeto DATETIMENODE es interesante, dado que no es posible modelar las fechas del portal como un simple atributo Date, debido a que una parte importante de la lógica de la comercialización del sistema se basa exclusivamente en las fechas, como ejemplo, los

nodos realizan las entregas en un rango de fechas, o las rondas están abiertas también durante un rango de fechas, esto nos obligó a extraer el simple atributo date a una clase en particular.

Evolución del Modelo de Objetos del Dominio

Entre los ajustes más relevantes del modelo de objetos del dominio, se puede señalar que el objeto DATETIMENODE se transforma en AVAILABLE_NODE, con la intención de enriquecer a los nodos de entrega con información extra y evitar modelar las fechas en los nodos. Con esta adecuación es posible listar los nodos que estuvieron entregando en una ronda en particular, algo importante dado que todos los fines de año el equipo de “La Justa” realiza un reporte anual en el que contempla la actividad de los nodos. Otro objeto que sufrió un cambio importante es IMAGE, en un principio lo modelamos para almacenar imágenes relacionadas con los productos, pero más tarde surgió la demanda que tanto los productores, las categorías y los nodos tuviesen una representación mediante una imagen. Además nosotros propusimos que los usuarios también tengan un thumbnail[7] a modo de hacer al sitio más amigable. Esto impactó en el diseño dado que en un primer momento solo tenía una relación con el objeto PRODUCT, pero eso lo solucionamos rápidamente invirtiendo la relación entre el Producto y la imagen, en lugar de que la Imagen sea quien tenga el ID del Producto, el Producto pasó a tener el ID de la imagen, entonces también pudimos agregar un ID al Usuario, a las Categorías y a todo lo que necesite imágenes. A continuación, en la figura 6, podemos apreciar parte del modelo final de objetos del dominio del portal de “La Justa” que es el que hoy se encuentra en producción.

[7] Una Thumbnail es una versión miniatura de una imagen que sirve a modo de referencia. Pueden ser miniaturas en un menú para reconocer al usuario logueado, como también imágenes en una lista para saber el contenido de una carpeta en un sistema operativo.

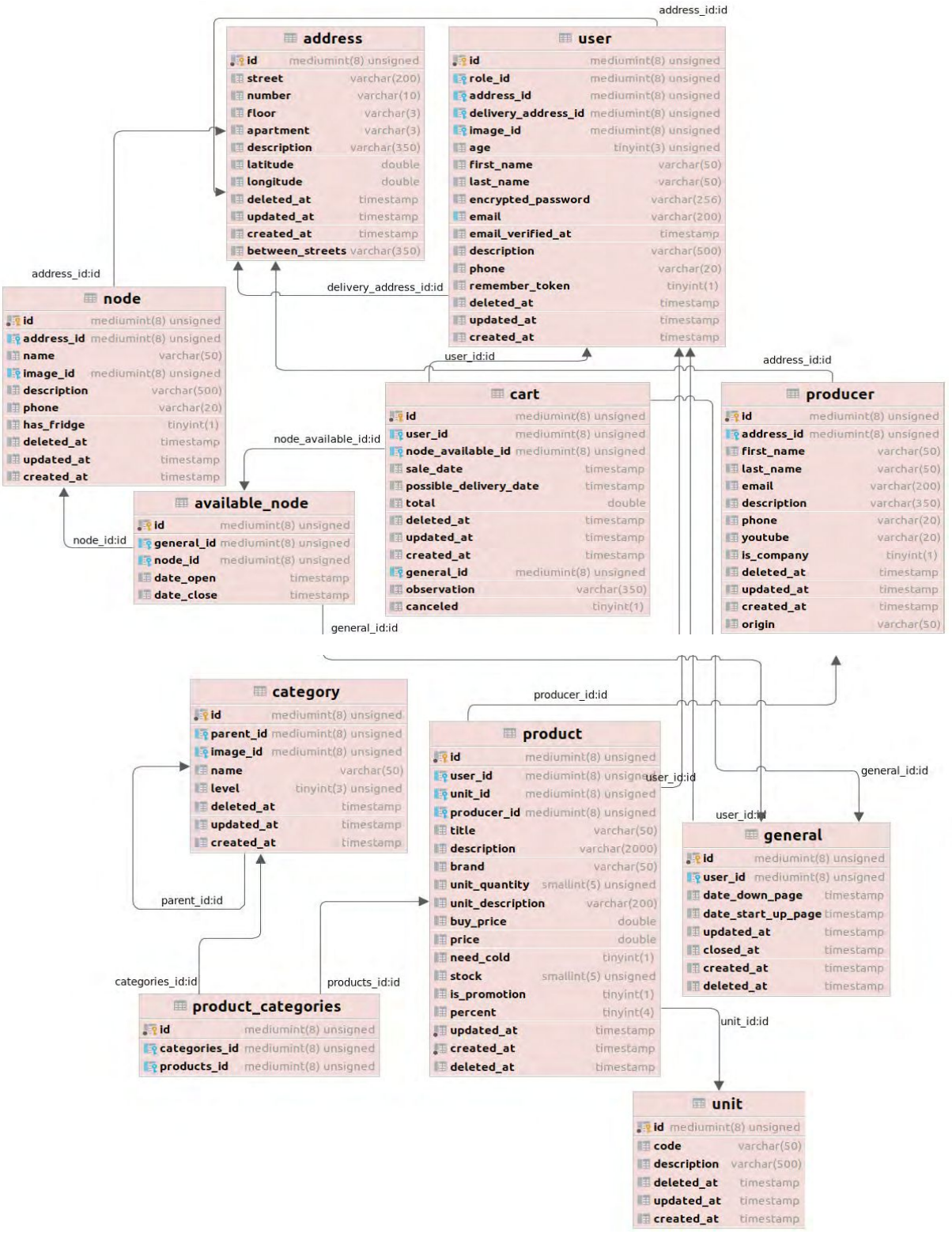


Figura 6: Modelo de Objetos - versión final

EL DISEÑO WEB INICIAL: EL MAQUETADO

A partir del primer modelo y los primeros requerimientos, comenzamos a trabajar con el equipo de diseñadores en comunicación visual de “La Justa”, que nos indicaron los colores y logo que debíamos utilizar, junto con la tipografía. A partir de ello, comenzamos a desarrollar un primer maquetado visual en el que se incluye la estructura de navegación del sitio y los elementos de diseño en detalle. En términos generales, este maquetado es el que se

implementó con algunos ajustes que se fueron incorporando a medida que se recibieron demandas y devoluciones de los usuarios.

El sitio actualmente cuenta con tres tipos de usuarios activos:

- **Usuarios anónimos:** son aquellos que ingresan al sitio, pero no están registrados, solamente pueden navegar e informarse sobre los productos ofrecidos, los productores de dichos productos, las formas de producción, de qué se trata “La Justa” y quiénes la conforman.
- **Usuarios consumidores:** son aquellos que están registrados en el portal, pueden realizar pedidos, informarse sobre los productos y productores.
- **Usuarios administrativos:** son aquellos que pueden cargar, configurar la información, tanto de productos como de productores y consumidores, y acceder a los informes sobre ventas y compras, además de poder realizar pedidos. Estos informes, los administradores los pueden descargar del portal en formato de planillas de cálculo y en los mismos se pueden encontrar los balances, detalles de las ventas, como ser cantidad de bultos vendidos por cada productor, el precio de compra de los bultos, el precio de venta total, la cantidad que deben distribuirse en los distintos nodos, etc. Además, dichas planillas ayudan a las personas que atienden en los nodos a organizar los pedidos de los consumidores, y de este modo se agilizan los tiempos de entrega.

En este punto nos interesa destacar los beneficios aportados a partir de la digitalización del proceso de comercialización, en términos de la eficiencia y precisión: antes de la implementación del portal todos los procesos necesarios para la comercialización se realizaban manualmente y por ejemplo disponer de la información necesaria para los días de entrega ocupaba mucho tiempo al equipo de trabajo de “La Justa”, además de la propensión a errores. Hoy, se cuenta con un proceso automático, preciso y rápido, disponible con un solo “click”. Para lograr el objetivo, buscamos en conjunto con el equipo de “La Justa” la manera de llevar a cabo la configuración del portal para que se pueda disponer de la información (y más) de una manera rápida, eficiente y flexible, mejorando la distribución de los productos en los nodos de entrega y facilitando el acceso a la información de administración a todos los implicados en los pedidos. Esto se logró partiendo de la premisa que se trabaja con un rango de fechas para la venta y un rango de fechas para la entrega en los distintos nodos. La entrega en los distintos nodos se realizan durante un rango horario y día en particular, y estos pueden variar entre rondas. Para poder definir las fechas de venta y entrega se usó la abstracción “Ronda” y para representarla se diseñó una tabla donde se pueden visualizar, cambiar, agregar, editar, copiar y/o eliminar las rondas. En dicha pantalla de configuración, presentada en la figura 7, además de configurar los nodos de entrega, también se tiene acceso al balance general, a las ventas realizadas en general y ventas realizadas por nodo, junto con el balance particular de cada ronda.

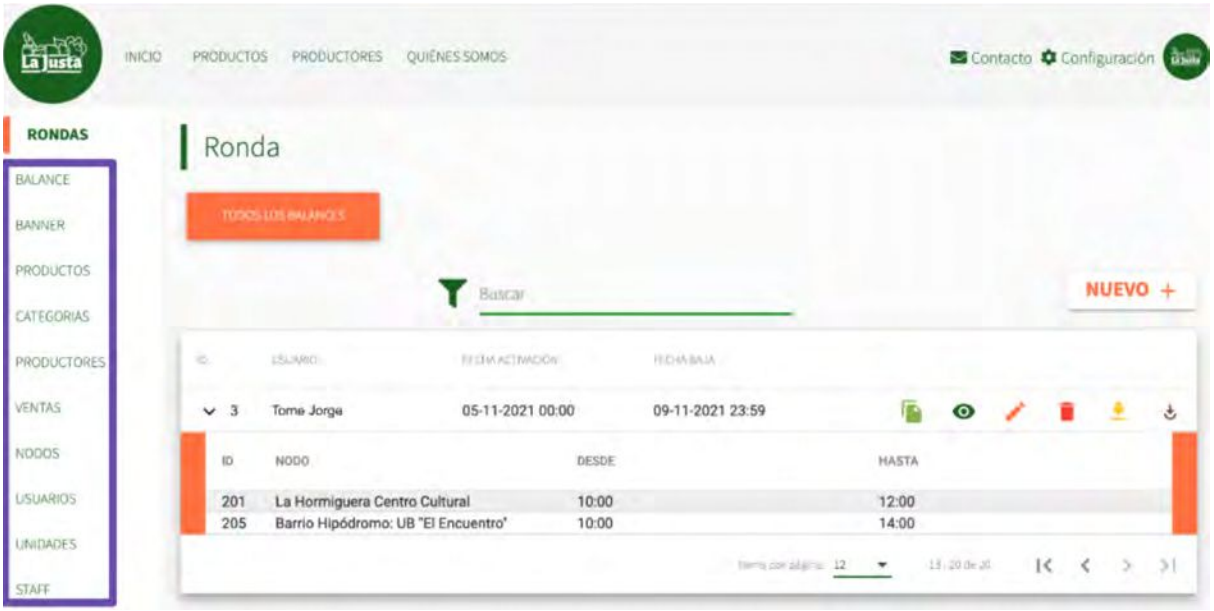


Figura 7: Pantalla de configuración de las rondas y disponibilidad de información de administración del portal

También tuvimos que tener en cuenta aparte de los días y horarios, si los nodos contaban con heladera o no por si algún producto agregado a la canasta necesita frío. Para que en el momento de seleccionar el punto de entrega solo tenga disponibles las opciones que cuenten con heladeras. Un punto importante a tener en cuenta es la configuración de los nodos, esto se realiza en una pantalla que visualiza una lista con todos los nodos en la que se pueden agregar, modificar y eliminar. Al agregar un nodo, se abre un modal como se muestra en la figura 8 donde además de nombre y calle se requieren los campos longitud y latitud (estos datos deben obtenerse de google maps para poder mostrar el mapa en la aplicación) y un check de tipo switch con la opción de si el nodo cuenta o no con heladera.

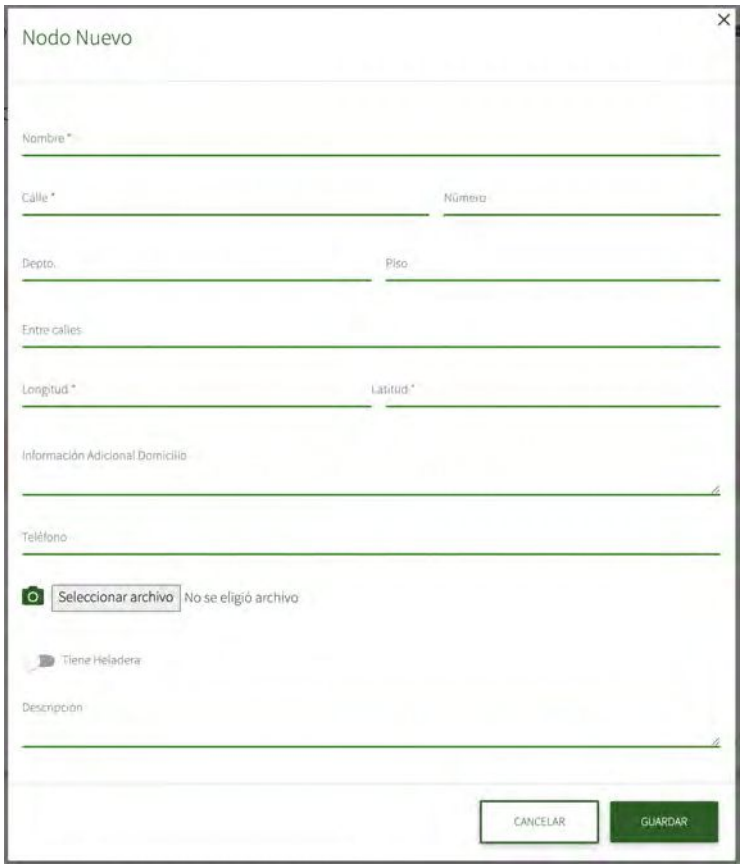


Figura 8: Modal para agregar un nodo

ARQUITECTURA CONCEPTUAL

El diseño de la arquitectura del portal de “La Justa” se basó en un frontend y backend distribuidos, esto significa que en lugar de tener un solo servidor de datos que retorna el HTML a cada consulta, lo primero que hacemos es pedirle al servidor de frontend el HTML y posteriormente solicitar los datos al backend. ¿Por qué se hace esto?, simplemente por la necesidad de reutilizar componentes; la figura 9 describe la arquitectura y cómo es posible reutilizar el mismo backend entre distintas aplicaciones, como por ejemplo web, mobile o algún otro sistema que desee consumir información.

Figura 9: Diagrama de arquitecturas Cliente Servidor

Hoy en dia es muy común encontrar aplicaciones en distintas plataformas u ofrecer servicios a distintas plataformas, si nuestras soluciones son monolíticas (esto quiere decir que el frontend y el backend son lo mismo) perderíamos la posibilidad de reutilizar componentes y en caso de querer integrar una aplicación móvil a nuestro ecosistema, deberíamos implementar todo denuevo, dejando como único elemento reutilizable a la base de datos.

Una de las evoluciones al modelo propuesto para el portal es la de microservicios, actualmente el modelado preferido para una gran mayoría de proyectos dado que lleva la reutilización al máximo. La figura 9 describe cómo sería un backend empleando microservicios, podemos ver que en lugar de tener un solo módulo tenemos varios, y que cada uno se centra en una particularidad específica, por ejemplo el de “envío de emails” solamente se encarga de enviar correos electrónicos a los usuarios o productores.

Las arquitecturas de microservicios es ampliamente utilizada en empresas de plataforma líderes dentro de Argentina, y han demostrado funcionar bastante bien dado que el problema a resolver es chico y acotado, y su reutilización es increíble. Sin embargo, este modelo fue descartado porque si bien son indiscutibles sus ventajas, el manejo de microservicios también aporta complejidad a la hora del desarrollo que deben evaluarse en relación al tamaño del proyecto. Es por ello que, en esta etapa del proceso de digitalización de “La Justa” consideramos que no aplican al desarrollo del portal. Para ponerlo en concreto, por ejemplo mantener 5 servicios como los definidos en la figura 10 es mantener 5 proyectos distintos. En la industria de desarrollo de software esto es factible porque cada equipo se hace responsable de su servicio, pero en un grupo de 2 desarrolladores y dada la magnitud del proyecto, la arquitectura de frontend y backend separados resulta en el enfoque más adecuado.

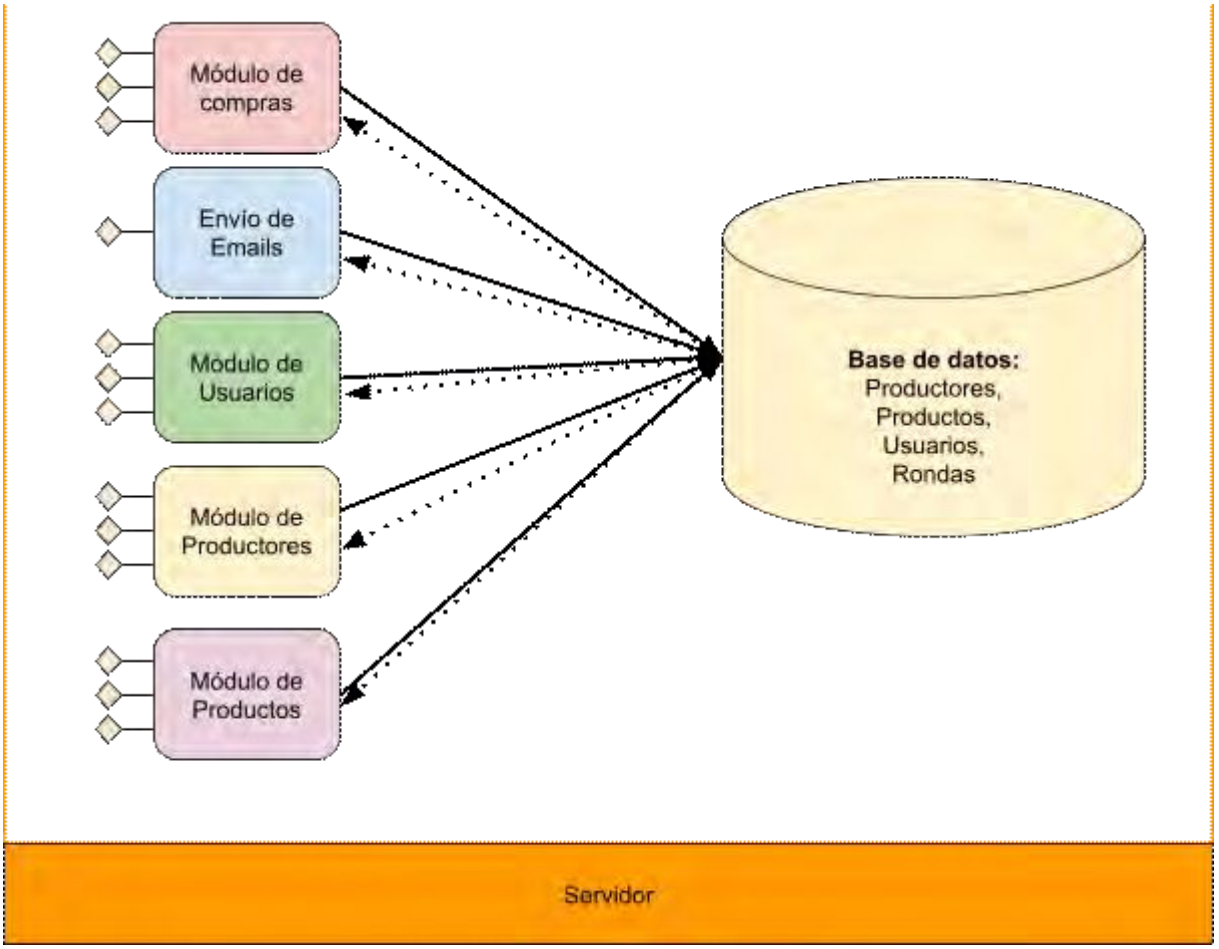


Figura 10: Diagrama de arquitecturas Microservicios

Para finalizar, nos gustaría mencionar que separar la arquitectura del portal en frontend y backend, nos permitió utilizar el backend en 2 materias de la facultad, “Laboratorio de Software” y “Java y Aplicaciones Avanzadas en Internet”. En ambos casos se hizo una copia del backend en el repositorio GitLab de cátedras del LINTI y de la base de datos, que, los estudiantes de “Laboratorio de Software” de la cursada 2020 lo utilizaron para desarrollar una aplicación móvil en Android nativo orientada a los consumidores de “La Justa” y, los estudiantes de “Java y Aplicaciones Avanzadas en Internet”, de la cursada 2021, para un desarrollo web, donde dicho backend se integró con otros sistemas desarrollados por los estudiantes que abordaban el problema de la logística del *delivery*. Cabe aclarar que ambos desarrollos tuvieron la intención de ofrecer a los estudiantes un ambiente de trabajo cercano al profesional, en tanto trabajaron con un backend real (con todo lo que eso implica) y acercarlos a una problemática local, como lo es las cadenas de comercialización de la ESS. Por otro lado, este intercambio fue útil como primer testeo del backend. En cuanto al enfoque adoptado para el desarrollo de la arquitectura, podemos afirmar que estos dos desarrollos hubiesen sido imposibles de realizar si la aplicación se basaba en una arquitectura monolítica donde todo estaba junto.

CAPÍTULO 5 - IMPLEMENTACIÓN DEL PORTAL: TECNOLOGÍAS Y METODOLOGÍAS EVALUADAS Y ADOPTADAS

En este capítulo describiremos las metodologías y tecnologías evaluadas y utilizadas en el desarrollo del portal. En relación a las tecnologías, el criterio de elección de las mismas, tanto para el desarrollo del frontend como del backend, se tuvo en cuenta nuestras experiencias profesionales como desarrolladores de software y las particularidades del proyecto, privilegiando aspectos de escalabilidad, rapidez de desarrollo y adhesión a estándares de software libre y código fuente abierto. Por otro lado, en cuanto a la metodología de trabajo, pusimos en práctica herramientas de las denominadas blandas como por ejemplo KANBAN para organizar las tareas a realizar y el testeo o evaluación.
En esta etapa también se escogieron las herramientas y tecnologías para poder llevar a cabo la implementación y se optó por trabajar con software libre. También, se trabajó en la búsqueda de servidores para alojar el portal, poniéndose en producción en febrero de 2021, en principio con un grupo chico de consumidores con los que se testeo el funcionamiento y actualmente es el portal que utilizan en “La Justa”.

SOFTWARE LIBRE Y OPEN SOURCE

Para llevar a cabo el proyecto decidimos utilizar herramientas y tecnologías open source. A continuación haremos una breve descripción de los conceptos de software libre y open source:

- **«Software libre»**[8], representa un movimiento liderado por Richard Stallman, físico estadounidense, creador del sistema operativo GNU y de la Free Software Foundation. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Es decir, el «software libre» es una cuestión de libertad, no de costo económico; el movimiento de «software libre» respeta la libertad de los usuarios y la comunidad de compartir el software.
- **«Open source»** (Código abierto) es algo distinto: su filosofía está basada y es un término que denota que un producto incluye permisos y licencias para utilizar su código fuente, documentos de diseño o contenido. Mientras que el software libre va dirigido o enfocado a las libertades para estudiarlo, modificado, y utilizarlo libremente, el software de código abierto, cuyo nombre indica que los usuarios pueden acceder al código fuente, se enfoca en las facilidades que ofrece la construcción colaborativa del software y las ventajas que esto genera a la hora de su desarrollo. Cuando se trabaja con código abierto está permitida la distribución del código siempre que se respeten los términos de su licencia y no varíen desde su primera adquisición hasta su distribución. Aunque su definición práctica es diferente, de hecho casi todas las herramientas informáticas y programas de código abierto son libres.

[8] Sitio oficial del sistema operativo GNU: https://www.gnu.org/philosophy/free-sw.es.html

Las expresiones «software libre» y «open source» se refieren, en general, al mismo conjunto de programas. No obstante, al basarse en valores diferentes, lo que dicen acerca de esos programas es algo distinto. El movimiento del software libre defiende la libertad de los usuarios, es un movimiento en pro de la libertad y la justicia. El enfoque de código abierto, valora principalmente las ventajas prácticas como modelo de desarrollo de software.
A continuación, en la figura 11, se describen las tecnologías utilizadas para el desarrollo del portal, todas ellas responden a código abierto.

TECNOLOGÍAS ADOPTADAS PARA EL DESARROLLO DEL BACK-END	
Spring Boot	Framework que contiene las librerías necesarias para desarrollar una API REST, así como también persistir objetos Java en base de datos usando Hibernate. También posee un Tomcat embebido que nos ayuda a correr la aplicación sin necesidad de un servidor.
KOTLIN	Lenguaje de programación usado para el desarrollo de los servicios y la capa de persistencia, desarrollado por JetBrains.
MariaDB	Motor de base de datos relacional, nació como alternativa a MySQL luego de que la misma fuera comprada por Oracle.
Flyway	Herramienta para el versionado de base de datos.
TECNOLOGÍAS ADOPTADA PARA EL DESARROLLO DEL FRONT-END	
Angular 2	Framework utilizado para el desarrollo de aplicaciones SPA (Single Page Applications) desarrollado por Google.
Interceptor	Componente de $http service del core principal de Angular
JWT	Estándar común que se centra en la generación de un Token que contiene datos del usuario encriptados, y nos permite validar los request.
TECNOLOGÍAS ADOPTADAS PARA EL DESPLIEGUE Y MANEJO DE CÓDIGO	
Gradle	Framework encargado del manejo de dependencias, todas las librerías .jar utilizadas en el desarrollo son descargadas y manejadas por Gradle.
Docker	Contenedores y virtualización, esta herramienta nos permite empaquetar un sistema operativo, la aplicación y sus dependencias en un solo elemento.
Gitlab Actions	Ejecutor de tareas para la compilación y creación de los contenedores.
Nginx	Servidor web utilizado para servir el front-end y usado como reverse proxy para acceder al back-end.

Figura 11 - Tecnologías utilizadas para el desarrollo del portal

LA METODOLOGÍA KANBAN

Es una metodología para gestionar el trabajo que surgió en Toyota Production System (TPS). A finales de los años 40, Toyota implementó en su producción el sistema “just in time” (“justo a tiempo”) que en realidad representa un sistema de arrastre. Esto significa que la producción se basa en la demanda de los clientes y no en la práctica tradicional “pull” de

fabricar productos e intentar venderlos en el mercado. Dicho método fue tomado por los desarrolladores de software, llamándolo Kanban que viene del japonés y traducida literalmente quiere decir tarjeta con signos o señal visual. El tablero más básico de Kanban está compuesto por tres columnas: “Por hacer”, “En proceso” y “Hecho”. Si se aplica bien y funciona correctamente, serviría como una fuente de información, ya que demuestra dónde están los cuellos de botella en el proceso y qué es lo que impide que el flujo de trabajo sea continuo e ininterrumpido.

Kanban principios básicos y prácticas

David J. Anderson formuló el método Kanban como una aproximación al proceso evolutivo e incremental y al cambio de sistemas para las organizaciones de trabajo. El método está enfocado en llevar a cabo las tareas pendientes y los principios más importantes pueden ser divididos en cuatro principios básicos y seis prácticas (David J. Anderson, 2011).

Los principios son:

- **Principio 1: Empezar con lo que hace ahora**

  Kanban no requiere configuración y puede ser aplicado sobre flujos reales de trabajo o procesos activos para identificar los problemas. Por eso es fácil implementar Kanban en cualquier tipo de organización, ya que no es necesario realizar cambios drásticos.
- **Principio 2: Comprometerse a buscar e implementar cambios incrementales y evolutivos**

  El método Kanban está diseñado para implementarse con una mínima resistencia, por lo que trata de pequeños y continuos cambios incrementales y evolutivos del proceso actual. En general, los cambios radicales no son considerados, ya que normalmente se encuentran con resistencias debidas al miedo o la incertidumbre del proceso.
- **Principio 3: Respetar los procesos, las responsabilidades y los cargos actuales**

  Kanban reconoce que los procesos en curso, los roles, las responsabilidades y los cargos existentes pueden tener valor y vale la pena conservarlos. El método Kanban no prohíbe el cambio, pero tampoco lo prescribe. Alienta el cambio incremental, ya que no provoca tanto miedo como para frenar el progreso.
- **Principio 4: Animar el liderazgo en todos los niveles**

  Este es el principio más novedoso de Kanban. Algunos de los mejores liderazgos surgen de actos del día a día de gente que está al frente de sus equipos. Es importante que todos los integrantes del equipo fomenten una mentalidad de mejora continua (Kaizen) para alcanzar el rendimiento óptimo a nivel de equipo/ departamento/ empresa. Esto no puede ser una actividad a nivel de dirección.

Las seis prácticas centrales identificadas por David J. Anderson que deben estar presentes para una implementación con éxito.

1. **Visualizar el flujo de trabajo**

   Lo primero y lo más importante se centra en entender qué se necesita para el transcurso de un producto desde su pedido hasta su entrega.

Para visualizar el proceso en Kanban, se necesita un tablero con tarjetas y columnas. Cada columna del tablero representa un paso en su flujo de trabajo. Cada tarjeta Kanban representa un elemento de trabajo.

Cuando se comienza a trabajar en el elemento X, hay que arrastrarlo hasta la columna "Por hacer" y cuando el elemento esté acabado, se lo debe mover hasta la columna "Hecho". De esta forma, se puede seguir el progreso fácilmente y detectar los cuellos de botella.

2. **Eliminar las interrupciones**

   La segunda práctica de Kanban se enfoca en establecer los límites del trabajo en proceso (los límites WIP). Si no hay límites de trabajo en proceso, no estaríamos haciendo Kanban.

   Limitar el trabajo en proceso (WIP) significa que un sistema de arrastre (pull) se aplica sobre partes o sobre todo el flujo de trabajo. Estableciendo un número máximo de elementos por etapa que aseguren que una tarjeta se "arrastra" al siguiente paso sólo cuando hay capacidad disponible. Tales restricciones muestran rápidamente las áreas problemáticas en el flujo para que puedan ser identificadas y resueltas.

3. **Gestionar el flujo**

   La idea de implementar un sistema Kanban es crear un flujo continuo e ininterrumpido. Por flujo nos referimos al movimiento de elementos de trabajo a través del proceso de producción. Lo que interesa es la velocidad y la continuidad del movimiento.

4. **Hacer las políticas explícitas (Fomentar la visibilidad)**

   No se puede mejorar algo que no se entiende. Esta es la razón por la cual el proceso debe estar bien definido, publicado y promovido.

5. **Circuitos de retroalimentación**

   Para que el cambio positivo ocurra, tenga éxito y sea duradero, se necesita hacer reuniones regulares para la transferencia de conocimiento (circuitos de retroalimentación).

   También existen las reuniones para la revisión de entrega de servicios, la revisión de operaciones y la revisión de riesgos.

6. **Mejorar colaborando (usando modelos y el método científico)**

   La forma de lograr la mejora continua y el cambio sostenible dentro de una organización se consigue a través de la visión compartida para un futuro mejor y la comprensión colectiva de los problemas que deben superarse.

Como comentamos anteriormente, la metodología empleada para el desarrollo del portal responde al método KANBAN y la herramienta elegida para llevarlo adelante fue Trello[9]. Consideramos que fue la mejor forma que encontramos para trabajar junto con el equipo de "La Justa" de manera coordinada y asincrónica, teniendo en cuenta que el portal fue

[9] Trello es una herramienta de gestión que tiene un perfil gratuito, es flexible y sirve para gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas y tener una visibilidad de principio a fin.

desarrollado íntegramente durante la pandemia COVID-19 y bajo la disposición del ASPO. En términos prácticos, el equipo de “La Justa” nos anotaba funcionalidades que creían necesarias y reportaban los errores que iban encontrando y a la vez que les permitía tener una idea de los avances del proyecto más allá de las reuniones periódicas que íbamos teniendo. La figura 12 permite observar cómo los usuarios cargaban nuevas tarjetas en la columna de “Errores o cambios vitales”, “Sugerencias - Cambios” o en “otros para el futuro etc” mientras que nosotros movíamos esas tarjetas a la columna “Done” una vez el desarrollo era terminado o el bug era resuelto.

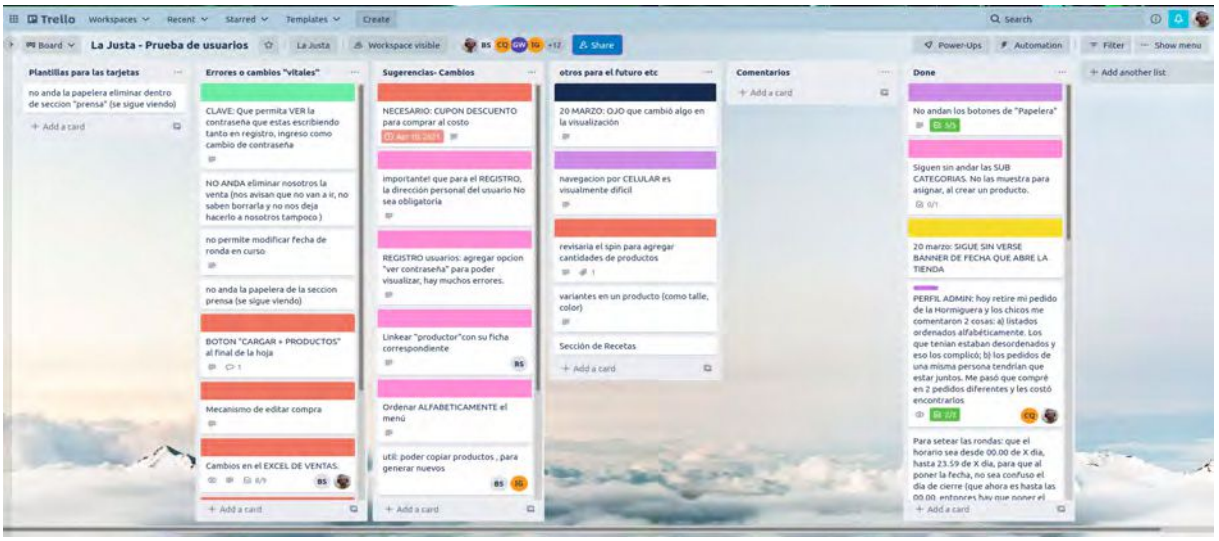


Figura 12: Ejemplo de Trello board con usuarios de “La Justa”

ARQUITECTURA REST

Para llevar a cabo la conexión entre el frontend y el backend nos decidimos por utilizar API REST (REpresentational State Transfer) o API RESTful, que es una interfaz de programación de aplicaciones (API o API web) que se ajusta a la arquitectura REST. Brevemente describiremos qué es una API, qué es una API RESTful y es una arquitectura REST.

¿Qué es una API? ¿Qué es una API RESTful?

Cuando hablamos de una API nos referimos al conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas. Una API establece de qué manera un módulo de un software se comunica o interactúa con otro para cumplir una o varias funciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta). Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima. En otras palabras, las APIs permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el

sistema comprenda la solicitud y la cumpla. Es la mediadora entre los usuarios o clientes y los recursos o servicios web que se quieren obtener. De esta manera las organizaciones pueden compartir recursos e información mientras conservan la seguridad, el control y la autenticación, lo cual les permite definir la información y contenido al cual puede acceder cada usuario. En este sentido, una API puede ser privada para el uso interno de la organización, abiertas sólo para cierto grupo de clientes, o públicas para que cualquier desarrollador pueda consultar e interactuar con ellas o crear sus propias API para que lo hagan. Como ejemplo de este estilo, tenemos las tarjetas de crédito, VISA, MasterCard, American Express, o cualquier otra pasarela de pago; como ejemplo MercadoPago, expone un conjunto de endpoints[10] que cualquier otra aplicación puede consumir y hacer uso de su servicio. Esto nos permite integrar fácilmente un sistema desarrollado por otros en nuestro sistema.

Otra característica importante y ventajosa de las APIs es que no se necesita saber cómo se recibe el recurso, de dónde proviene, de qué manera está implementado, ni el lenguaje utilizado por el servicio que nos da la respuesta.

En particular en este trabajo de Tesis desarrollamos una API RESTFul cuya interacción entre el cliente y el servidor se realiza mediante el protocolo HTTP. La solicitud de información es a través de una solicitud HTTP y la respuesta puede tener distintos formatos, por ejemplo JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. Actualmente el formato JSON es el más popular y utilizado por su simplicidad, dado que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje. JSON permite una comunicación fluida y uniformizada sin errores de conversión de datos, soportando incluso la transferencia de datos BLOB (Binary Long Object) es decir formatos multimedia.

Arquitectura REST: Breve descripción

El informático Roy Fielding es el creador de REST o de lo que se conoce como transferencia de representación de estado. REST no es un protocolo ni un estándar, REST define un estilo de arquitectura o lineamientos para el diseño de sistemas distribuídos poco acoplados como son los sistemas web. REST no impone reglas de implementación solo lineamientos arquitectónicos dando libertades en cuánto a la implementación.

REST establece 6 lineamientos arquitectónicos que caracterizan a una API RESTful:

- **Interfaz uniforme:** se cuenta con una interfaz uniforme para los elementos que permite que la información se transfiera de forma estandarizada. A la vez, se establece el cumplimiento de las siguiente condiciones:
  - Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente.

[10] URL accesible desde internet que retorna HTML, XML, Texto plano o JSON. Se utiliza en el armado de sistemas web para comunicar los navegadores con los servidores o servidores entre si.

- Los recursos pueden ser manipulados por el cliente a través de la representación que reciben, ya que esta contiene suficiente información para hacerlo.
- Los mensajes auto descriptivos devueltos al cliente deben contener la información necesaria para describir cómo éste debe procesarla.
- Debe contar con hipertexto o hipermedios, lo que significa que cuando el cliente acceda a algún recurso, debe poder usar hipervínculos para encontrar todas las demás acciones disponibles en ese momento para utilizar.

- **Cliente-servidor:** se dispone de una arquitectura compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
- **Sin estado:** la comunicación entre el cliente y el servidor debe ser sin estado, lo que implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de las llamadas y respuestas son independiente y está desconectada del resto.
- **Almacenable en caché:** los datos se almacenan en caché y optimizan las interacciones entre el cliente y el servidor.
- **Sistema en capas:** que organiza en jerarquías cada tipo de servidor (los encargados y responsables de la seguridad, del balanceo de carga, etc.) que participan en la recuperación de la información solicitada, siendo invisibles para el cliente.
- **Código bajo demanda (opcional):** puede contar con la capacidad para enviar códigos ejecutables del servidor al cliente cuando se lo solicite, extendiendo la funcionalidad del cliente.

Como se puede observar un concepto clave en una API RESTFul es el de recurso. En palabras de Roy Fielding (2000): *“La abstracción clave de la información en REST es un recurso. Cualquier información que pueda ser nombrada puede ser un recurso: un documento o imagen, un servicio temporal (por ejemplo, "el tiempo de hoy en Los Ángeles"), una colección de otros recursos, un objeto no virtual (por ejemplo, una persona), etc.”.*

A su vez, sobre esos recursos podemos realizar acciones diferenciadas a través de métodos HTTP distintos, que por otro lado resulta interesante observar que siempre estuvieron disponibles pero escasamente utilizados en el desarrollo de sistemas web. A estos métodos también se los conoce como verbos HTTP y sus funciones son las siguientes:

- **GET:** se usa para operaciones de consulta de datos, es posible consultar listados de recursos o un recurso determinado.
- **POST:** se usa para agregar recursos.
- **PUT:** se usa para realizar modificaciones sobre recursos.
- **PATCH:** se usa para realizar modificaciones parciales en un recurso
- **DELETE:** se usa para eliminar un recurso determinado.

Otro elemento que resulta interesante comentar es acerca de los encabezados y los parámetros HTTP, tanto en los métodos de la solicitud como de la respuesta de la API de RESTful, dado que estos contienen información de identificación importante con respecto a

los metadatos, la autorización, el identificador uniforme de recursos (URI), el almacenamiento en caché, las cookies y otros elementos.

Una característica diferenciadora de las API RESTFul respecto de tecnologías de servicios web clásicos como SOAP[11], es que son rápidas y livianas, escalan fácilmente, por ello se las considera adecuadas para el desarrollo de aplicaciones de Internet de las cosas (IoT) y de aplicaciones móviles.

TECNOLOGÍAS ADOPTADAS PARA EL DESARROLLO DEL BACKEND

A continuación describiremos las tecnologías que utilizamos para el desarrollo del backend, específicamente Spring Boot como framework de servicios RESTFul e inyección de dependencias, Kotlin como lenguaje de programación, MariaDB como base de datos relacional, Gradle como manejador de dependencias, Flyway como versionador de base de datos, Docker y Gitlab Actions. También, describiremos brevemente algunas de las tecnologías evaluadas durante el proceso de selección de las mismas, como DropWizard, Maven, MySQL o PostgreSQL, Liquibase o Java.

FRAMEWORK DE SERVICIOS RESTFUL: SPRING BOOT

Podemos afirmar que Spring Boot es uno de los frameworks de desarrollo web en Java más utilizados en la industria de software hoy en día (SimilarTech Ltd.©, 2013-2022). Se caracteriza por su robustez, el soporte de sistemas de amplia demanda concurrente de usuarios, es seguro, se acopla fácilmente a distintos frameworks y distintas tecnologías, tiene un servidor integrado que permite que las aplicaciones puedan ejecutarse de manera serverless[12] y ofrece una multiplicidad de módulos que lo hacen adaptarse perfectamente a la necesidad de nuestras aplicaciones, entre ellos:

- **Spring Security:** para el manejo de seguridad por sesión, token, filtros a nuestros servicios y demás.
- **Spring Data:** para poder exponer directamente nuestro modelo y así no tener que generar controllers, servicios y repositorios por cada elemento que queremos hacer un CRUD (del inglés Create, Read, Update, Delete). Como inconvenientes podemos encontrar la falta de seguridad en los endpoints[13], dado que los endpoints generados no utilizan spring security y pueden ser accedidos por cualquier request (petición), y la escasa facilidad para agregar lógica a los endpoints dado que todo lo genera automáticamente el módulo de Spring Data. Sin embargo, para los endpoints que solo guardan el objeto que nos llega por petición y que se encuentran detrás de un gateway en microservicios[14], este módulo es ideal, dado que nos deja exponer un CRUD completo sin necesidad de escribir una línea de código.

[11] Es un protocolo que define el envío de XML entre dos servicios para poder comunicarse entre ellos.
[12] serverless: es la posibilidad de ejecutar una aplicación sin la necesidad de desplegarla en un servidor. en Java por ejemplo es común empaquetar nuestra aplicación en un WAR que luego es desplegado en un servidor Tomcat (entre otros). La idea de serverless es empaquetar nuestra app en un JAR y que el Tomcat esté embebido en el mismo, entonces con solo ejecutar el JAR tendríamos nuestro servidor corriendo sin problemas en cualquier máquina virtual.
[13] URL accesible desde internet que retorna HTML, XML, Texto plano o JSON.
[14] un gateway nos permite redirigir cualquier request HTTP a otro servidor, funcionando como punto de entrada a la aplicación

- **Spring Web:** este módulo contiene las librerías de Jersey[15] para el manejo de RESTful; también contiene la librería Jackson[16] para convertir objetos a formato JSON o viceversa. Este módulo es muy importante en el desarrollo de aplicaciones web.
- **Spring Test:** contiene todo lo necesario para testear una aplicación web, desde JUnit[17] hasta un MOCKMvc[18] que nos permite simular peticiones a nuestra capa de servicios REST sin necesidad de levantar todo el contexto de Spring.
- **Spring Cloud:** si bien este módulo no fue utilizado en el desarrollo del portal de “La Justa”, ofrece todo lo necesario para realizar desarrollo de microservicios, desde “service discovery” para detectar automáticamente servicios en nuestra aplicación, hasta “load balancer” para poder tener más de un servicio que realice la misma tarea e ir balanceando la carga entre los mismos y así atender todas las peticiones de los usuarios sin sobrecargar ningún servicio, pasando por configuraciones remotas, gateways y demás servicios.

Spring Boot no es un producto, sino un conjunto integrado de distintos productos o módulos, y esto es ideal a la hora de desarrollar un sistema en poco tiempo, porque, entre otras cosas, evita lidiar con múltiples dependencias de librerías. En este sentido, al elegir Spring Boot como motor de nuestro backend, agregando una sola dependencia a nuestro manejador de dependencias, Maven[19] o Gradle[20], permite que Spring Boot se encargue de descargar todas las librerías necesarias y en su versión correcta.

Desde nuestro punto de vista, Spring Boot constituye un elemento clave a la hora de desarrollar aplicaciones en Java. Intentaremos explicarlo con un ejemplo o escenario: es una práctica muy común desarrollar aplicaciones web Java (backend) y tener problemas de dependencias, por ejemplo al agregar en nuestro POM.xml[21] el conector de base de datos, en su versión elegida, junto con el mapeador de objetos usado, en su versión elegida, nos damos cuenta que tenemos un error en el conector de base de datos porque el mapeador que utilizamos es de una versión muy actual (moderno) o muy desactualizada y no coindice con la esperada. Entonces, si bien la librería es la misma, para nuestro conector de base de datos que el mapeador se encuentre en una versión, cuando se espera otra, es un problema que da miles de dolores de cabeza a los desarrolladores en relación a las dependencias de librerías. Spring Boot resuelve este problema de manera muy sencilla para el desarrollador, en lugar de tener que agregar el conector de base de datos, el mapeador y cualquier otro elemento relacionado a la persistencia de datos, en el POM.xml, simplemente le agregamos el módulo de persistencia a nuestro proyecto y el resto lo resuelve el framework. En la figura

---

[15] https://github.com/eclipse-ee4j/jersey Jersey es un framework web que nos permite implementar API REST.
[16] Librería de Java que nos permite convertir objetos a JSON y JSON a objeto.
[17] https://junit.org/junit5/
[18] Librería que nos permite ejecutar llamadas HTTP en nuestros tests y simular el flujo de nuestra aplicación con el fin de realizar testeos de integración.
[19] Explicación de Maven más adelante en el documento, aqui
[20] Explicación de Gradle más adelante en el documento, aqui
[21] Archivo fundamental en el desarrollo con Maven, este archivo contiene toda la información de las dependencias del proyecto, así como información sobre el tipo de buildeo que se va a utilizar.

13 se puede observar la pantalla de configuración de Spring Boot, en la que se configuran todos los detalles de nuestra aplicación, por ejemplo el lenguaje a utilizar, el framework para manejo de dependencias, la versión de Java que queremos usar (recordemos que tanto Groovy como Kotlin corren sobre Java) y las dependencias que podemos agregar de antemano al proyecto. Spring Web es un ejemplo de lo antes mencionado, posee las dependencias de Jersey, pero nosotros no agregamos Jersey explícitamente, en su lugar agregamos el módulo Spring Web, que ya lo contiene.

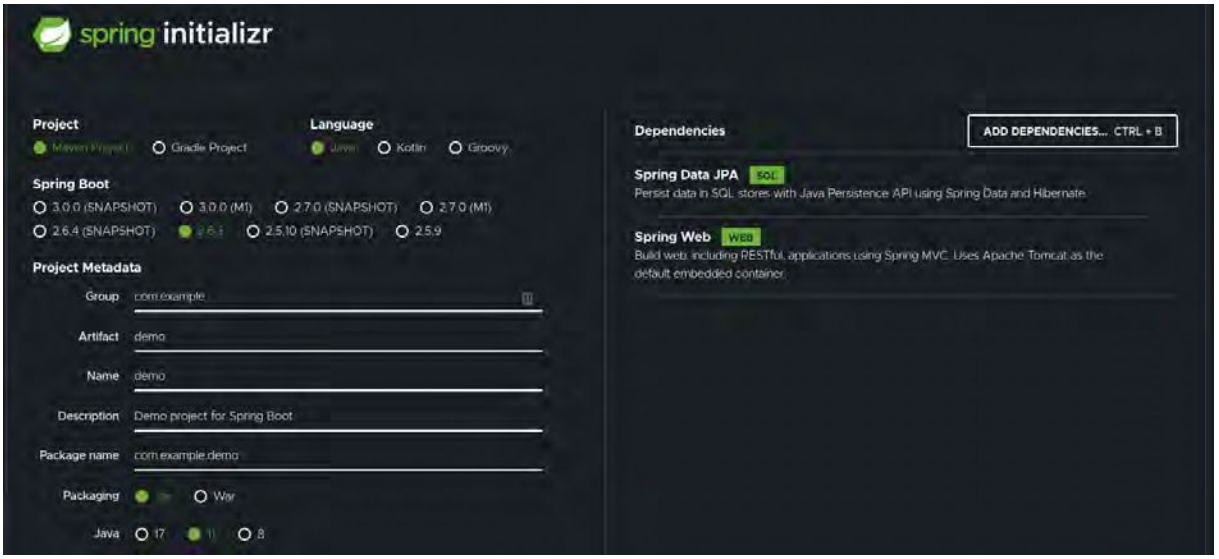


Figura 13: Spring initializer página principal

Como se describió previamente con Spring Boot no es necesario asegurar que cada librería esté en la versión correcta y compatible con el resto de elementos de la aplicación, dado que esto lo maneja automáticamente e internamente. Cada elemento necesario para el desarrollo web, como por ejemplo las librerías de Jakarta que utiliza el Tomcat[22] y que tenemos embebido, está en la dependencia de ese módulo, en su correcta versión, haciendo que el kickoff[23] y puesta en marcha de un proyecto web con Spring Boot, sea rápido y sencillo. Sin mencionar que existe una amplia variedad de herramientas de desarrollo web como Spring Initializr[24], clientes CLI (Command Line Interface) que podemos descargar e instalar en nuestros PCs, o los mismos IDEs o entornos de desarrollo, que nos ofrecen múltiples y variadas plantillas de proyectos.

Como punto negativo sobre Spring Boot podemos señalar su complejidad, posee muchas librerías y dependencias, esto hace que la curva de aprendizaje de Spring Boot sea muy alta, además de que muchas veces el framework oculta la implementación de sus funcionalidades, haciendo que encontrar un problema sea bastante difícil. Spring Boot hoy en día se utiliza en proyectos de gran envergadura donde la seguridad y el tráfico entre usuarios es alto o hasta crítico. Para desarrollos más chicos o del tipo start up[25] los

[22] https://tomcat.apache.org/ servidor Java para aplicaciones web hechas en Java.
[23] kickoff o primera reunión de un nuevo proyecto.
[24] https://start.spring.io/
[25] Son compañías chicas de bajo presupuesto pero con una gran idea a desarrollar.

desarrolladores prefieren otro tipo de framework, algunos los vamos a describir más adelante dado que fueron evaluadas como alternativas.

Análisis de Frameworks Web para Desarrollo de API RESTful

Como alternativa a Spring Boot se analizaron distintos frameworks web para la implementación de la API RESTful del portal. Es sabido que Kotlin compila Java y puede hacer uso de cualquiera de los siguientes frameworks que describiremos a continuación.

DropWizard

Dropwizard[26] es un framework web que se diferencia de Spring Boot por el tipo de tecnologías que usa, ofrece un menor número y variedad de librerías y frameworks que Spring Boot. Por ejemplo, la única alternativa para crear servicios REST en Dropwizard es Jersey y para atender servicios HTTP la única opción es Jetty como servidor HTTP embebido. En caso de necesitar o elegir utilizar alguna otra tecnología, se requiere configurar manualmente. En el caso de Spring Boot nos deja seleccionar que servidor vamos a utilizar a la hora de correr nuestra aplicación, Jetty está soportado, pero Tomcat es la opción por defecto. Por otra lado, en Dropwizard disponemos de las librerías Jackson, Logback como framework para el manejo de Logs de la aplicación (la evolución del viejo Log4J).

Como podemos ver, la idea de Dropwizard no dista mucho de Spring Boot, la cantidad limitada de opciones lo convierte en un framework lightweight[27] y mucho más fácil de aprender que Spring Boot. Una de las diferencias más grandes que nos podemos encontrar es que Dropwizard no tiene un framework para inyección de dependencias, donde Spring Boot utiliza Spring[28] como inyector. En Dropwizard tenemos que agregar las dependencias a mano a cada objeto durante el arranque del servicio, o buscar alguna alternativa como puede ser la librería de Google para el manejo de dependencias llamada Guice. Como podemos notar, Dropwizard no tiene nada que envidiarle a Spring Boot, es robusto y se monta sobre tecnologías importantes en el mercado. Fue descartado en nuestra tesina dada la experiencia con la que contamos con Spring Boot, pero sin lugar a dudas, creemos que el mismo desarrollo hecho con Dropwizard hubiese tenido el mismo resultado.

Ktor

En la próxima sección vamos a hablar sobre Kotlin y el por qué nos inclinamos a usar ese lenguaje de programación en lugar de Java. Pero en esta sección nos gustaría mencionar que Kotlin posee un framework para desarrollo web bastante potente, fácil de aprender y de montar llamado Ktor[29]. Dado

[26] https://www.dropwizard.io/en/latest/
[27] Que dispone de menos librerías, haciéndose más fácil de aprender y más liviano al utilizar recursos de la computadora.
[28] https://spring.io/
[29] https://ktor.io/

que Kotlin fue el lenguaje elegido para desarrollar, podríamos haber experimentado en este framework, que posee características muy interesantes, entre las que se destacan el uso de Corrutinas[30], una posibilidad de dormir las funciones que están esperando por una respuesta y despertarlas más tarde cuando el resultado es obtenido, algo muy parecido a las promesas de Angular.

El punto que observamos como negativo de Ktor es la falta de persistencia que maneja, éste solo se avoca a la capa de HTTP y si bien adaptar Hibernate no debería presentar mayores inconvenientes, nos pareció que ya estábamos experimentando demasiado con el lenguaje, como para ponernos a experimentar con el framework. De todas maneras Ktor se encuentra en nuestros radares de tecnologías interesantes para mirar luego del desarrollo del portal de "La Justa" dado todo lo que aprendimos del mismo durante la investigación para esta tesina.

Jhipster

Jhipster[31] Es una plataforma para el desarrollo web que a diferencia de los frameworks web antes mencionados, que solo se centran en el backend, Jhipster nos ofrece la posibilidad de crear un CRUD de manera automática corriendo una serie de comandos. Dichos comandos son interactivos en una interfaz de línea de comando y nos preguntan cosas como las propiedades o atributos que va a tener nuestra clase. Una vez elegido el comando crea todos los archivos de front-end necesarios, así como los archivos de back-end y las conexiones a la base de datos. Como ejemplo de las librerías que maneja la plataforma podemos seleccionar AngularJs[32], React[33] o Vue[34] como framework de desarrollo de front-end, así como Spring Boot (con Java o Kotlin), Micronaut[35], Quarkus[36], NodeJs[37] o .NET.[38] Como se puede apreciar, la Jhipster nos ofrece una flexibilidad muy alta a la hora de seleccionar tecnologías, así como rapidez en el desarrollo, dado que permite crear los archivos de vista HTML, el modelo en JS, el servicio en JS, el controller en Java, el servicio en Java y el repositorio en Java, con solo un comando.

Si bien Jhipster es una herramienta muy útil para el desarrollo cuando los tiempos son cortos, tiene una gran desventaja en el desarrollo y es que su funcionalidad se basa en crear componentes basados en líneas de comandos, quedando nuestros servicios muy genéricos, esto claramente es una desventaja muy grande, porque hablando desde el lado de la experiencia casi ningún proyecto puede adaptarse al framework y muchas veces necesitas

---

[30] Corrutinas: Funcionalidad integrada en Kotlin, que nos permite ejecutar código multithread sin la necesidad de bloquear la ejecución del código.

[31] https://www.jhipster.tech/

[32] https://angularjs.org/

[33] https://reactjs.org/

[34] https://vuejs.org/

[35] https://micronaut.io/

[36] https://quarkus.io/

[37] https://nodejs.org/en/

[38] https://dotnet.microsoft.com/en-us/

soluciones “out of the box”. Esto hizo que Jhipster sea descartado como posible herramienta en el desarrollo de “La Justa”.

LENGUAJE DE PROGRAMACIÓN: KOTLIN

Una pregunta que podríamos hacernos es: teniendo en cuenta que contamos con una amplia experiencia, de años trabajando en Java, y dado que nos decidimos por Spring Boot como framework de desarrollo para el back-end, ¿por qué nos elegimos utilizar un lenguaje particularmente nuevo como Kotlin, en lugar de Java?, la respuesta a esta pregunta es sencilla. Kotlin vio la luz por primera vez en 2011, si bien es un lenguaje con más de 10 años en el mercado, comparado con los más populares como ser Java, Python o Javascript que se encuentran hace más de 30 activos, podemos decir que Kotlin es un lenguaje joven. El mismo nació por parte de un proyecto de la compañía JetBrains[39] cuando se dieron cuenta que mantener código en Java es muy complejo, tratar de entender algo que fue implementado años atrás, posiblemente por un desarrollador que no se encuentra en la empresa y que por más que esté en producción posiblemente no siga las mejores prácticas, es un trabajo difícil. En ese momento se les ocurrió la idea de hacer un lenguaje más simple de comprender y compatible e interoperable con código Java. Esto les iba a ofrecer flexibilidad para migrar los sistemas que se encuentran en Java a Kotlin, sin la necesidad de tener un día D donde se deja atrás Java y se adopta Kotlin. La posibilidad de migrar un sistema por partes, gradualmente y con la facilidad de migrar primero lo simple y chico y luego atacar lo más grande y complejo, es una de las mejores cualidades que ofrece Kotlin. Más tarde, en el año 2019 Google anunció que Kotlin sería el lenguaje oficial para el desarrollo de aplicaciones en Android, impulsándolo increíblemente en el mercado. Durante el 2020 tuvo un auge muy grande en cuanto a su desarrollo, la particularidad que tiene Kotlin de compilar a distintos lenguajes como Java, Javascript, Swift e incluso hoy tiene un prototipo para compilar a código nativo, le da a Kotlin una versatilidad muy grande a la hora de ejecutarse sobre cualquier dispositivo de los que se encuentran en el mercado actualmente.

La pregunta que nos hacemos es la siguiente, ¿cómo es posible migrar un sistema legacy monolítico 100% desarrollado en Java a Kotlin?. Esto se logra gracias a que Kotlin compila a .class (formato de un compilado Java) y una vez compilado nuestro código no es posible distinguir si ese archivo .class fue desarrollado usando Java o Kotlin, y mejor aún, es posible decompilar el archivo .class desarrollado en Kotlin a archivos .java. De esta manera, se logra una flexibilidad enorme en cuanto a compatibilidad entre sistemas, y como mencionamos anteriormente, nos permite migrar por tramos nuestro código legacy.

Kotlin, nació como un lenguaje que ofrece las ventajas de los lenguajes modernos, por un lado se evitan las características de Java consideradas complejas por los desarrolladores,

[39] https://www.jetbrains.com/ empresa que se dedica en mayor medida al desarrollo de IDEs como ser IntelliJ, CLion, PHPStorm, PyCharm entre otros.

como por ejemplo el “boilerplate code”[40] generado por tener miles de getters y setters en una clase POJO[41], y por otro, poco a poco, con los años fue tomando funcionalidades existentes en otros lenguajes como Python o TypeScript agregando a su repertorio de funcionalidades, técnicas más modernas y cómodas para el desarrollador, como ser extensiones, parámetros nombrados, manejo de colecciones, reemplazo de strings, triple comillas para párrafos, entre otros.

Lenguajes de Programación Evaluados

Como describimos anteriormente, con Kotlin es posible compilar a múltiples lenguajes de programación, con lo cual no queda atado solo a desarrollo para dispositivos móviles, dado que puede utilizarse tanto para el front-end como para el back-end, esto lo hace un competidor con el resto de lenguajes del mercado. Nosotros en este punto nos centramos en lenguajes de programación para el desarrollo del backend.

Java

Una de los lenguajes mencionadas es Java, que si bien nuestra experiencia podría haber impulsado el desarrollo de esta Tesina en Java, el deseo de experimentar con un nuevo lenguaje inclinó la balanza del lado de Kotlin. Por otro lado, el poder utilizar Spring Boot fácilmente sin necesidad de adaptarlo fue un plus, y sin lugar a duda, la posibilidad de poner en producción código Kotlin de manera serverless con un Tomcat embebido, nos ayudó mucho a la hora de desplegar nuestra aplicación, dado que contábamos con el conocimiento necesario para hacerlo. Podemos afirmar, a partir de la experiencia en este trabajo de Tesis, que desplegar una aplicación Kotlin y una Java, es exactamente igual dado que una vez generado el archivo de despliegue (.jar) ambos se tratan de la misma manera.

JavaScript

Javascript, o mejor dicho NodeJS, estaba entre las posibilidades. Contar con un sistema funcionando con NodeJS es rápido y simple, Javascript es un lenguaje bien conocido, y TypeScript es la sintaxis utilizada en el frontend, esto haría muy fácil entender el código entre back y front dados que estarian escritos en el mismo lenguaje. Sin embargo, fue descartado porque el desarrollo del backend implicaba un conocimiento avanzado de estos frameworks, como pueden ser NodeJS backend propiamente dicho, Webpack para transpilar[42] el proyecto, NPM para el manejo de dependencias, NVM para manejar las distintas versiones de Node en nuestras

[40] Son piezas de código que se repiten una y otra vez aportando nada al producto final, un ejemplo claro son los getters y setters de toda clase Java.
[41] Plain Old Java Object, clase Java que solo contiene atributos, getters y setters, sin nada de lógica de negocio incluida.
[42] convertir código en TypeScript a código Javascript entendible por un navegador.

máquinas locales y el servidor, todas está tecnologías hacen que transitar la curva de aprendizaje consuma mucho tiempo, y mencionamos en varias ocasiones que los tiempos nos apremiaban durante todo el desarrollo del portal. Si bien el portal de “La Justa” se podría haber desarrollado en NodeJS, el aprendizaje para desarrollo de los servicios, el soporte y manejo de las distintas funcionalidades y la puesta en producción presentaban un desafío incompatible con los tiempos.

PHP

En sus comienzos, las primeras pruebas de concepto del portal se desarrollaron con PHP. Hasta la incorporación de Francisco al equipo, se evaluó la posibilidad de desarrollar el backend del portal en PHP. Esta idea se descartó dada la experiencia de Francisco con sistemas Java, la velocidad de desarrollo de la que podíamos disponer, y como mencionamos antes la facilidad de poner en producción y mantener un sistema desarrollado en Java, que finalmente fue en Kotlin.

MOTOR DE BASES DE DATOS: MARIADB

Para la capa de persistencia nos propusimos usar MariaDB, la cual es un desarrollo creado en 2009 por una comunidad que hizo un fork del proyecto de MySQL pero lo preparó con fines de uso corporativo. En 2010 Oracle compra MySQL y desde ese momento MariaDB tomó su lugar como una alternativa open source para el almacenamiento de datos. MariaDB es una base de datos relacional, con la cual interactuamos usando Hibernate, esto no nos da lugar a poder decir mucho acerca de su uso, dado que delegamos casi toda esa funcionalidad a la herramienta, pero podemos mencionar que en lugar de tener una base de datos dedicada (RDS de AWS ofrece ese servicio), lo que hicimos en esta Tesina, fue instalar en un servidor cloud de Donweb[43], una instancia del servidor y cliente CLI de MariaDB, montados sobre un LInux Ubuntu 20.04 LTS.

La configuración más difícil fue habilitar el acceso exclusivo del contenedor docker donde corre el backend (podemos ver la descripción de docker aquí, donde se explica qué es y para qué se usa en la industria) así como ingresar desde un cliente remoto de nuestras computadoras. Para conseguir esto, ejecutamos las queries de GRANT PRIVILEGES con el usuario de “La Justa” sobre los distintos entornos desde los cuales nos íbamos a conectar:el local, para el usuario de “La Justa”, el remoto para el mismo usuario desde una máquina con IP de Argentina, y otro, remoto también, para el mismo usuario de “La Justa”, pero esta vez la IP de la cual se conectará es el contenedor docker. En la figura 14 se puede observar los distintos usuarios con acceso a la base de datos de “La Justa”, los primeros son para

[43] Proveedor de servicios web, como hostings o clouding: https://donweb.com/es-ar/

acceder solamente desde el servidor, el tercero es para hacer pruebas remotas desde nuestras computadoras y el último es el servidor de back-end.

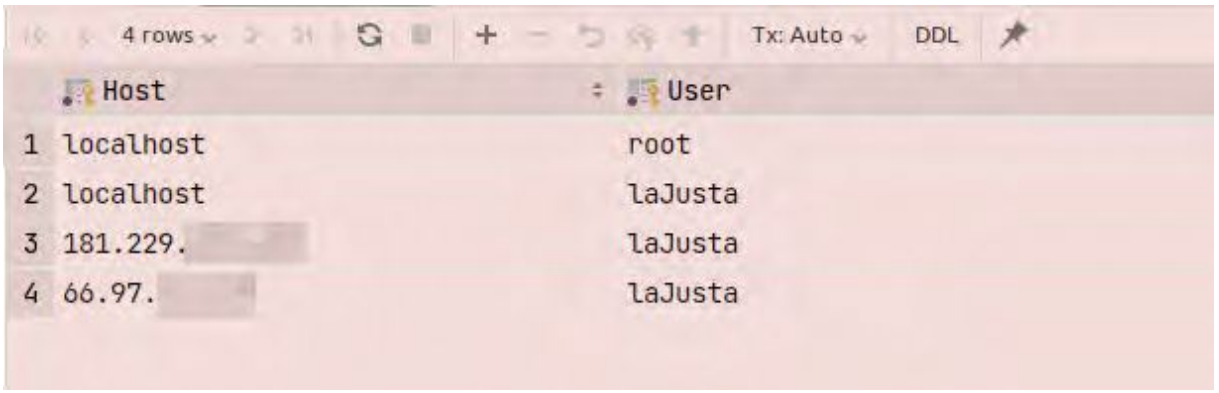


Figura 14: Perfiles de usuarios con acceso a la base de datos

El establecimiento de diferentes perfiles de usuario se relaciona con aspectos de seguridad del portal, que nos permite evitar conexiones desde afuera del servidor hacia la base de datos, dado que el servidor de MariaDB no permite el acceso a toda conexión que no provenga de esas IP por más que el usuario y la password sean correctos. Pero tenemos otro problema, si el puerto queda expuesto es muy probable que alguien trate de acceder a él, si bien la base de datos va a rebotar todo acceso remoto que no provenga de dichas IP, es muy probable que con suerte y usando algoritmos de “fuerza bruta” se termine dando con la IP y la password, es por eso que decidimos cerrar el puerto desde el firewall que nos ofrece Donweb. Una vez que sabemos que no vamos a utilizar más la base de datos de producción para reportes o algún dump, cerramos ese puerto en el firewall de Donweb y de este modo limitamos el acceso a cualquier usuario malicioso. La figura 15 nos muestra la configuración del firewall con el puerto 3306 abierto solo a conexiones que provengan de la IP 181.229.xxx.xxx, y la figura 16 muestra que por seguridad removemos esa conexión una vez terminadas las pruebas.

Figura 15: Datos de la configuración del Firewall del servidor de DonWeb habilitando la conexión a MariaDB

Figura 16: Datos de la configuración del firewall del servidor de DonWeb con la conexión a MariaDB cerrada

En síntesis con estas medidas de protección, mitigamos el acceso al servidor de MariaDB desde fuera del servidor, es decir si alguien intenta hacerlo, encontrará obstáculos para conectarse.

Con respecto al hosting de la base de datos, tenemos que hablar de un aspecto importante que es la alocación de recursos, sabemos que la base de datos es un recurso crítico en todo desarrollo de software, una alta demanda de consultas concurrentes hacen que la aplicación se comporte de manera lenta, unos minutos de caída por falta de recursos podría hacer que un cliente pierda dinero. Con lo cual, tenemos que asegurarnos una asignación adecuada de procesador y memoria a nuestra máquina virtual, a modo de ofrecer el servicio necesario. No queríamos arriesgarnos a ofrecer una mala experiencia de usuario a los consumidores, para ello el servicio contratado es de 2 Cores de CPU y 4 GB de RAM. En la figura 17 podemos ver el consumo de CPU y que el mismo no supera el 70%, e incluso en una semana de uso intensivo hubo un pico de consumo, pero generalmente se mantiene por debajo del 50%.

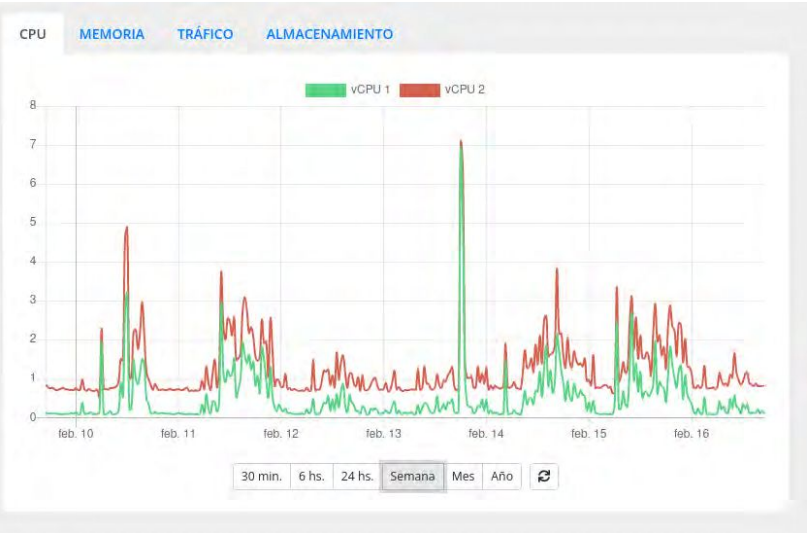


Figura 17: Consumo de CPU en el servidor

La figura 18 nos muestra el consumo de memoria RAM, rondando los 512 MB de consumo en toda una semana, manteniéndose muy por debajo de la memoria disponible.

Figura 18: Consumo de memoria RAM en el servidor

La figura 19 nos muestra el uso del almacenamiento SSD (Solid State Drive) que tiene del servidor, en el mismo se guardan fotos, la base de datos y los logs de las aplicaciones. Actualmente se ocupan 7 GB.

Figura 19: Consumo de almacenamiento SSD en el servidor

Dos cosas que nos gustaría destacar en base al cloud donde tenemos corriendo la base de datos, es que DonWeb agregó la posibilidad de esconder nuestro cloud en una red privada y así no exponer nada a Internet, como se muestra en la figura 20. Esto no solo es una característica interesante, sino que es casi obligatoria en un desarrollo web, dado que al tener la base de datos en una red privada, se niega todo tipo de acceso al mismo por medio de la red pública de Internet. En caso de querer ingresar al servidor de base de datos, debemos conectarnos primero al servidor de backend y de ahí acceder a la base, esto permite agregar más seguridad a nuestra base de datos, escondiéndola de posibles escaneos maliciosos.

Figura 20: Configuración de acceso al servidor, posibilidad de bloquear el acceso público

Como punto final, posiblemente nos podríamos preguntar dado que tenemos nuestra aplicación de front-end y back-end 100% dockerizada: ¿por qué no dockerizar la base de datos con volúmenes virtuales?, logrando de esta manera tener nuestra información persistida en disco y con la posibilidad de levantar la base dentro de un contenedor. Aunque en la teoría suena bien, la realidad indica otra cosa, por un lado los profesionales del área de operaciones no recomiendan tener la base de datos dockerizada en producción, dada la sobrecarga que Docker aplica sobre sus contenedores, esto estresa a la base de datos, que como dijimos anteriormente es el recurso más crítico dentro de un desarrollo web. A partir de nuestra experiencia como desarrolladores, podemos afirmar que lo más adecuado es tener la base de datos directamente instalada sobre el sistema operativo.

Motores de Base de Datos Analizados

PostgreSQL

Otra base de datos relacional, potente, muy utilizada en la industria y open source, es PostgreSQL. Sin lugar a dudas una muy buena alternativa a MariaDB. A nuestro entender, no existe una razón técnica para descartar Postgres, fue más un gusto personal dado que ambos tenemos basta experiencia con MariaDB, motivo que inclinó la balanza.

SQLite

SQLite es una plataforma open source, orientada a tener una base relacional de poco tamaño, se almacena en un archivo, es “human friendly” con lo cual podemos leer los datos con cualquier editor de texto. Aunque no tiene la potencia de MariaDB en cuanto al desarrollo de una aplicación web productiva, funciona perfectamente en desarrollos más chicos, además al ser un archivo, la portabilidad y escalabilidad es inimaginable, sin embargo no se adecúa para proyectos grandes. Si bien los archivos donde se guardan las tablas se pueden partir y hacer archivos más chicos, al final si la tabla crece mucho, los tiempos de respuesta empeoran.

Oracle DB

Oracle DB es posiblemente la base de datos más potente de todas las evaluadas incluyendo MariaDB, soporta tablas con millones de datos y responde en cuestión de segundos. Muchas de las características de Oracle DB están disponibles en todas las demás bases de datos, como ser Materialized Views, Stored Procedures y Triggers, pero su potencia es la diferencia del resto. Es ampliamente usada en aplicaciones de gran envergadura, ubicando a Oracle DB como una de las bases de datos más rápidas y fiables del mercado. La razón más importante por la cual fue descartada es que no es open source y además tiene un costo de licencia bastante alto, y pese a que su potencia es la más alta en el mercado, el portal de “La Justa” puede, y de hecho actualmente lo hace, funcionar perfectamente con alternativas open source como es el caso de MariaDB. Oracle DB encaja perfecto en sistemas grandes con mucha demanda de datos y accesos concurrentes a la base, si pudiésemos asignar un tamaño al tipo de problema donde las bases son usadas, podríamos afirmar que Oracle DB es perfecta para grandes o enormes sistemas, mientras que MariaDB se adapta mejor a pequeños, medianos y grandes. Para cerrar la idea, el portal de “La Justa” no requiere mucha potencia de base de datos, con lo cual MariaDB es una clara ganadora.

¿Qué problemas resuelven las herramientas de versionado de base de datos?

Antes de describir las herramientas de versionado, introduciremos los problemas que enfrenta la industria de desarrollo de software al no contar con bases de datos versionadas y cómo las herramientas de versionado cumplen el cometido de resolver estos problemas.

Hoy por hoy el desarrollo de software trata de representar el problema a resolver en pequeñas piezas fáciles de comprender, fáciles de completar y fáciles de entregar, dicho esto, una de las principales estrategias de la industria del software es dividir los problemas en problemas más chicos y versionarlos. Esto se lleva a cabo en casi todas las áreas, primero el problema se divide en tareas que son etiquetadas dependiendo de su urgencia, su facilidad o su dependencia con otras tareas, estas tareas son conocidas como “historias de usuario”, más tarde dichas tareas son completadas por los desarrolladores en orden de prioridades. El código por su parte también se divide en porciones más chicas que se completan y entregan como un todo, una estrategia comúnmente usada es la de agrupar las porciones de código que los desarrolladores entregan con las historias de usuario que mencionamos en el párrafo anterior, una vez hecho esto es muy fácil saber qué cambios se realizaron para qué tarea y en qué momento. Este enfoque de trabajo, por más sencillo que parezca, no siempre es implementado y las consecuencias de no hacerlo son bastante críticas dado que, no saber dónde buscar el error en el código implica tiempo perdido en análisis.

Ya mencionamos al problema y al código como elementos que se particionan en piezas más chicas y fáciles de abordar, pero ¿qué pasa con la base de datos?, ¿cómo son manejados

los cambios en ambientes donde el versionado de base de datos no es una opción?. Muchos procesos de desarrollo de software involucran un paso en el cual un administrador de base de datos ejecuta una serie de scripts sobre las mismas, impactando los cambios en algún orden aleatorio, sin conocimiento del negocio ni de lo que se está intentando aplicar en ese momento. Por más especializada que sea la persona encargada de realizar la tarea, no deja de ser un proceso manual y propenso a errores. El versionado de base de datos propone solucionar los problemas de ejecución de un script nuevo, dado que esto se realiza de forma automática en todos los ambientes (desarrollo, QA y producción). Para ello, la propuesta consiste en: a) versionar los scripts de base de datos para darle orden, es decir el script versión 1 corre antes que el versión 2; b) reducir la cantidad de código en cada script, lo ideal es que cada archivo solo contenga 1 script, y c) garantizar que el flujo sea el mismo en todos los ambientes, es decir si un script se ejecutó bien en desarrollo o QA debe de ejecutarse bien en producción.

HERRAMIENTA DE VERSIONADO DE BASES DE DATOS: FLYWAY

Es la herramienta de versionado de bases de datos usada en el desarrollo del portal “La Justa”. Se trata de una herramienta open source con licencia Apache 2.

Flyway nos permite modificar la base de datos con código, almacenar ese código en los repositorios y ejecutarlo de forma automática para así poder realizar cambios sin riesgo de errores. Dichos repositorios nos ayudan a versionar nuestras entregas, podemos crear “branches de releases” (como por ejemplo: release_1_0, release_sprint_1, release_version_1) que posteriormente son desplegados en un ambiente de testeo y, una vez terminadas todas las pruebas pertinentes, son puestas en producción de manera segura, en caso de fallar algo, se puede desplegar una versión anterior y todo quedaría igual que antes.

Flyway funciona de una manera muy simple, gracias a su integración con Maven podemos correr todos los cambios desde la raíz del proyecto sin ningún problema, o podemos instalar el cliente CLI de Flyway y ejecutar los scripts desde línea de comandos.

Para poder comenzar a ejecutar scripts solo debemos crear una carpeta en resources/ con el nombre db.migration/, dentro de ella ubicamos otra carpeta llamada migration/ y listo, comenzamos a crear scripts .sql en esa carpeta migration/. La figura 21 describe la estructura actual del proyecto de “La Justa”.

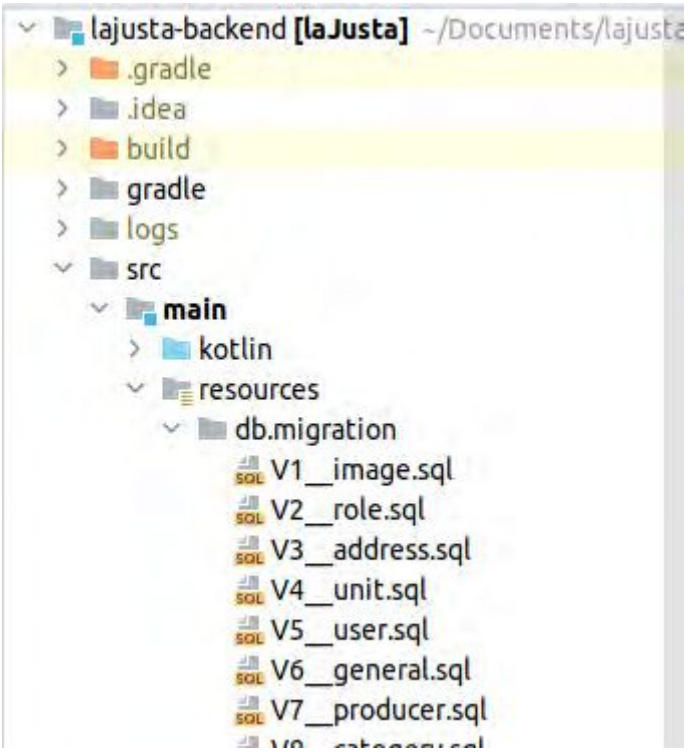


Figura 21: Estructura de archivos de migración de Flyway

Un detalle importante a tener en cuenta es que Flyway es incremental, eso quiere decir que debemos comenzar nuestros scripts con el nombre Vx__nombre.sql donde x es el número de orden de ejecución que tiene ese script, esto nos deja ordenar la ejecución de scripts y evitar fallos originados por alguna tabla que depende de otra que aún no fue creada. Otro detalle muy importante a tener en cuenta es que así como dijimos que Flyway es incremental, el mismo no acepta cambios sobre scripts que ya fueron ejecutados, eso quiere decir que si queremos cambiar algún detalle de un script ya ejecutado, debemos crear un nuevo script, porque toda modificación va a resultar en error. En la figura 22 podemos observar que Flyway crea una tabla en nuestra base de datos y mantiene un registro de los scripts que fueron ejecutados hasta el momento, además, posee una columna llamada checksum en la cual guarda información relacionada con el contenido de los scripts ya ejecutados, de esta manera cualquier cambio va a ser detectado (porque los checksum serían distintos) y presentados como error al usuario. Nuevamente, reiteramos, que Flyway maneja los cambios de manera incremental, por ende si deseamos cambiar el script ejecutado en V3__address.sql, debemos crear uno nuevo con dichos cambios en lugar de modificar este script. En la figura 22 también podemos observar que actualmente nuestra base de datos ejecutó 40 scripts diferentes, eso puede significar que está en la versión 41. En caso de encontrar algún problema, podríamos eliminar la base de datos, y crear una nueva, pero esta vez en lugar de correr los 40 scripts, ejecutaríamos sólo el 29 o 30, quedando esta nueva base de datos en la versión 30 o 31 respectivamente, esto es así porque no hacemos uso de la versión enterprise de la librería, en caso de tener la versión paga podemos correr un rollback que bajaría la versión de nuestra base de datos a un

estado anterior. Esta es una de las características más fuertes de Flyway que nos permite crear bases de datos posicionadas en cualquier momento del desarrollo con solo un comando. Vale mencionar que la versión paga de esta herramienta también permite volver una versión avanzada a una anterior sobre la misma base de datos sin la necesidad de eliminarla primero.

	installed_rank	version	description	script
1	40	41	alter cart product table	V41__alter_cart_product_table.sql
2	39	40	alter cart table	V40__alter_cart_table.sql
3	38	39	add role	V39__add_role.sql
4	37	38	alter balance table	V38__alter_balance_table.sql
5	36	37	alter cart table	V37__alter_cart_table.sql
6	35	36	news	V36__news.sql
7	34	35	alter cart	V35__alter_cart.sql
8	33	34	producer tag producer	V34__producer_tag_producer.sql
9	32	33	producer tag	V33__producer_tag.sql
10	31	32	producer product cataloge	V32__producer_product_cataloge.sql
11	30	31	alter table add general id	V31__alter_table_add_general_id.sql
12	29	30	reset password	V30__reset_password.sql
13	28	29	alter tables	V29__alter_tables.sql

Figura 22: Muestra todas las migraciones que fueron ejecutadas en la base de datos

Herramientas de versionado de bases de datos evaluadas

Liquibase

Liquibase[44] es un proyecto open source para manejar el versionado de base de datos, la idea de esta herramienta es poder tener una carpeta en el proyecto donde estén alojados todos los scripts SQL y un archivo XML que nos permite controlar el orden de ejecución de dichos scripts. En la figura 23 podemos observar el XML que controla las migraciones de Liquibase de haber sido usado para el proyecto.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
 xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
       http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd">
   <include file="src/main/resources/db/migration/V1_image.sql"/>
   <include file="src/main/resources/db/migration/V2_role.sql" />
   <include file="src/main/resources/db/migration/V3_address.sql"/>
   <include file="src/main/resources/db/migration/V4_unit.sql"/>
   <include file="src/main/resources/db/migration/V5_user.sql"/>
   <include file="src/main/resources/db/migration/V6_image.sql"/>
   <include file="src/main/resources/db/migration/V7_producer.sql"/>
</databaseChangeLog>
```

Figura 23: XML que controla la ejecución de scripts SQL en Liquibase.

[44] https://www.liquibase.org/

El archivo XML puede ser fácilmente versionado en Git y ofrecer la habilidad de correr la cantidad de scripts que necesitamos, sin ejecutar demás ni de menos en nuestras migraciones. Liquibase también ofrece la posibilidad de hacer rollback al poner la query que deseamos correr en caso de querer hacerlo, por ejemplo si agregamos una tabla nueva, lo más lógico es que nuestro rollback scripts borre dicha tabla, entonces, en caso de necesitar bajar de una versión superior a una previa o dos previas, corremos el rollback y listo, los scripts que se ejecutan dejan a nuestra base de datos en un punto estable sin demasiado esfuerzo. Además Liquibase nos ofrece distintas formas de ser ejecutado, una podría ser por medio de un CLI que nos deja correr los comandos en nuestra base de datos desde cualquier consola, o integrado en nuestro proyecto en un IDE. Este segundo enfoque corre como un ejecutable que se descarga usando Maven o Gradle y nos permite correr cualquier migración nueva en nuestra base con solo levantar el proyecto. Flyway es más liviano y fácil de adaptar, y con los tiempos acotados que teníamos, esto lo puso por encima de Liquibase como el ganador.

Herramienta de mapeado de objetos a bases de datos relacionales: Hibernate

Hibernate es el ORM (Object–Relational Mapping) más famoso de Java, nos permite escribir en una base de datos relacional un objeto y sus dependencias, también agrega un lenguaje de consulta propio llamado HQL donde podemos escribir en modo de query SQL pero utilizando los objetos dentro de las queries para facilitar su uso. Entre todas sus buenas características Hibernate nos provee la facilidad de crear la base de datos de nuestro proyecto a partir del modelo de objetos del dominio escrito en Java. Dicho en otras palabras, si borramos la base de datos y ejecutamos Hibernate con la opción de crear la base de datos, éste nos crearía automáticamente todas las tablas con los atributos de las clases del modelo, además la integración a nivel proyecto lo hace super fácil de usar, solo basta con arrancar el backend y la base de datos se crea automáticamente.

Si bien las ventajas que ofrece Hibernate aporta al proyecto en cuanto a versionado de la base de datos, también tiene problemas que nos hicieron descartarlo. Un ejemplo es el tema que no podemos realizar inserts para tablas de control, como por ejemplo Categorías o Estados, esto hace muy molesto el tema de tener que cargar todo manualmente una vez hecho un drop (eliminado) de la base de datos y ejecutar una nueva versión. Además tengamos en cuenta que hacer un rollback (reversión) en producción es prácticamente imposible, dado que para ello debemos primero hacer un dump (volcado), y perderíamos todos los datos cargados hasta ese momento. Para una temprana etapa del desarrollo, cuando podemos eliminar todo y crear la base desde cero, Hibernate es un buen aliado, pero no es conveniente su uso en sistemas maduros donde los datos son importantes.

TECNOLOGÍAS ADOPTADAS PARA EL DESARROLLO DEL FRONTEND

Se realizó un estudio de las alternativas existentes para desarrollar el frontend priorizando que el mismo sea “web responsive” para que se adapte a las pantallas de celulares y tablets, para ello se necesitó de una tecnología que sea capaz de crear una interfaz de usuario amigable y a su vez comunicarse con el back-end.

Como es sabido las aplicaciones web se clasifican en aplicaciones web estáticas y aplicaciones web dinámicas. Las primeras se caracterizan por la ausencia de funcionalidades frente a la posibilidad de que el cliente altere el contenido, es decir, son básicamente páginas informativas que tienen sus secciones y contenido fijo, mientras que las webs dinámicas son páginas en las que su contenido es fácilmente y frecuentemente modificado. Para el desarrollo del front-end del portal de “La Justa” se optó por la tecnología SPA (Single page application) debido a su rapidez y a que ofrece una comunicación con el servidor muy fluida, generando una experiencia de usuario satisfactoria, en la que se cuenta con una sola página pero diferentes vistas (partes de nuestra web), que se van intercambiando, dando la sensación de que visitamos varias páginas, aunque en realidad estemos en la misma página pero con diferente vista.

Del análisis acerca de las tecnologías disponibles para el desarrollo del front-end se concluyó que Angular 9 era la más adecuada dado que se trata de la evolución de AngularJS, es modular y escalable, adaptándose a nuestras necesidades.

El diseño de Angular adopta el estándar de los componentes web. Se trata de un conjunto de APIs que permiten crear nuevas etiquetas HTML personalizadas, reutilizables y auto-contenidas, que luego pueden utilizarse en otras páginas y aplicaciones web. Estos componentes personalizados funcionan en navegadores modernos y con cualquier biblioteca o framework de JavaScript que funcione con HTML.

Además, Angular 9, es la tecnología con la que estamos más familiarizados en el ámbito laboral dado que actualmente es uno de los frameworks más utilizados del cual se dispone de mucha información en Internet.

Framework para el desarrollo del frontend: Angular 2

Angular 2 es un framework desarrollado y mantenido por Google, es de código abierto, y para el desarrollo utiliza TypeScript[45]. Al momento de buildear[46] la aplicación, TypeScript se transpila, traduciendo el código a JavaScript y de esta manera es soportado por los navegadores. Angular 2, es la evolución de AngularJS, vale aclarar que estos frameworks no son retro compatibles, esto significa que un sistema desarrollado en AngularJS no puede ser ejecutado en Angular 2. Otra de las

[45] TypeScript: lenguaje de código abierto, orientado a la programación orientada a objetos, muy similar a JavaScript.

[46] En la jerga de desarrollo de software, el término buildear significa realizar una versión ejecutable del código, en este caso para los navegadores.

diferencias respecto a su antecesor es que Angular 2 está más orientado a dispositivos móviles mientras que AngularJS no se hizo para soportar este tipo de dispositivos.

Angular 2 responde al paradigma Single Page Application (SPA) o aplicación de página única, es decir es una aplicación web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como si fuera una aplicación de escritorio. En un SPA todos los códigos de HTML, TypeScript y CSS se cargan una sola vez o los recursos necesarios se cargan dinámicamente cuando lo requiera la página, normalmente como respuesta a las acciones del usuario (Flanagan D, 2006).

Dado que Angular 2 es un framework, ofrece muchas más funcionalidades que una simple librería. Con otros software similares como React, lo más común es tener que utilizar varias bibliotecas de terceros a la hora de desarrollar una aplicación, instalando paquetes adicionales para hacer el routing (enrutamiento, se describirá más adelante) es decir definir la ruta de navegación, para la gestión de dependencias, para realizar llamadas a APIs REST, para hacer el testing, etc

En la figura 24, se describen los elementos fundamentales que componen un front-end en Angular 2 y la relación entre los mismos.

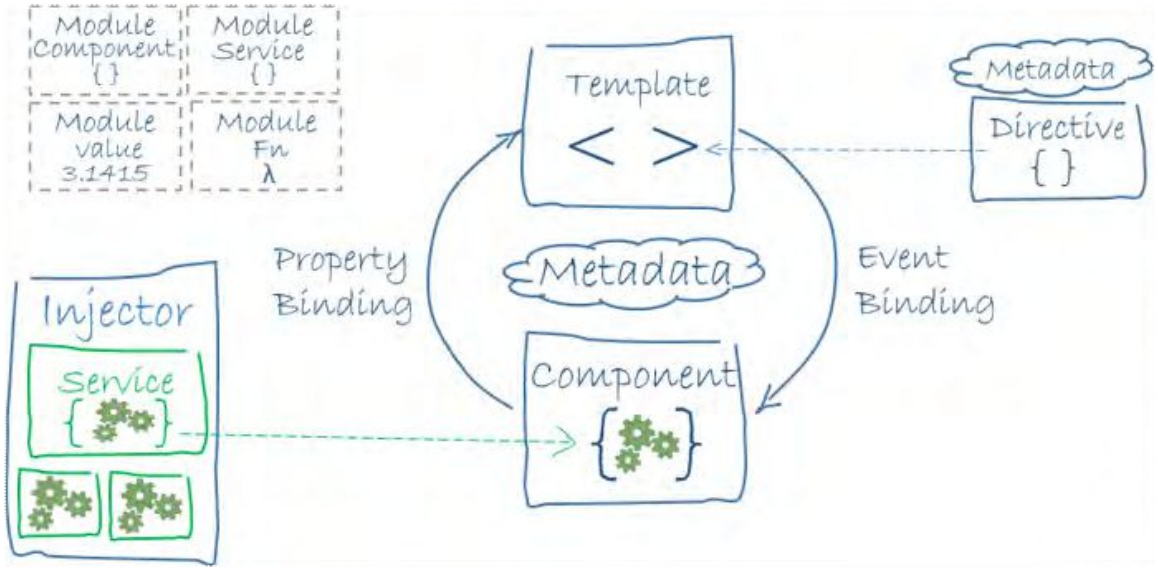


Figura 24: Diagrama de relación de elementos que componen una aplicación front-end desarrollada en Angular 2[47]. Extraído de https://angular.io/guide/architecture#whats-next

El lenguaje de Angular 2 es TypeScript: aunque se puede programar en ECMAScript[48] puro, el equipo de Angular estableció como lenguaje TypeScript, y la documentación y los ejemplos se encuentran en este lenguaje, ofreciendo varias ventajas, como la consistencia en la documentación. Por ejemplo, ES6 (o sea, ECMAScript 2015) ofrece varias formas a la hora de declarar un objeto, lo cual puede ser confuso. Con TypeScript esto no pasa, y toda la

[47] Resumiendo el diagrama, un componente y una plantilla definen una vista en Angular, el componente cuenta con un decorador que agrega los metadatos, en los que se encuentra incluida la plantilla. Mientras que las directivas junto con el enlace markup en la plantilla modifican las vistas basadas en datos y la lógica del programa, El inyector de dependencias proporciona los servicios a un componente, un ejemplo de estos es en servicio de enrutamiento que permite definir la navegación entre las vistas

[48] Es una especificación de lenguaje de programación y actualmente está aceptado como el estándar

sintaxis y la manera de hacer las cosas en el código es la misma, lo que añade coherencia a la información y legibilidad al código. Aunque no es obligatorio el uso de TypeScript, el equipo de desarrollo de Angular recomienda su adopción, buscando un mejor y más fácil mantenimiento de las aplicaciones.

Las componentes en Angular 2: como se describió previamente, Angular 2 está orientado a componentes, es decir se agregan directivas que extienden la funcionalidad del HTML usando para ello una nueva sintaxis y de esa manera es posible usar lógica que será ejecutada en el DOM (Document Object Model, modelo de objetos del documento). Las componentes Angular son una porción de código que es posible reutilizar en otros proyectos de Angular con muy poco esfuerzo. Cada componente se define mediante una **clase que contiene datos y lógica de la aplicación**, y está asociada con una **plantilla HTML** que define una vista en la que se va a mostrar. La plantilla HTML combina el HTML con los markup de Angular, que son los encargados de conectar los datos de la aplicación con el DOM y que, a su vez, pueden modificar los elementos HTML antes de que estos se muestran. Un componente, en términos simples, es un elemento que encapsula código reutilizable.

Para mantener una consistencia entre la información que se visualiza y los datos de la aplicación, utiliza data binding (enlace de datos) en dos sentidos (bidireccional), lo que significa que los cambios realizados en cualquiera de los dos lados, se verá reflejado en el otro.

Pipes: Angular 2, a su vez, cuenta con pipes, un tipo de directiva para las planillas que mejoran la experiencia del usuario mediante la transformación de valores para mostrar. Por ejemplo, se usan para mostrar fechas o valores de monedas. Existen algunos pipes por defecto para transformaciones comunes, y también podemos definir nuestros propios pipes.

NgModules: las aplicaciones en Angular 2 son modulares, el framework cuenta con su propio sistema de modularidad llamado NgModules que son contenedores encargados de organizar el código de la aplicación. Estos pueden contener componentes, servicios y otros archivos con código, también pueden importar la funcionalidad que se exporta desde otro NgModules y exportar la funcionalidad seleccionada para que la utilicen otros NgModules. Cada aplicación tiene al menos una clase NgModule llamado convencionalmente AppModule (módulo raíz) y se encuentra en un archivo llamado app.module.ts que se encarga de iniciar la aplicación conectando una jerarquía de componentes con el DOM. El código de la figura 25 permite visualizar el AppModule del front-end del portal de “La Justa”, el cual contiene el componente principal de la aplicación llamado AppComponent e importa varios módulos.

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
                ToastrModule.forRoot({
      positionClass: "toast-bottom-right",
      closeButton:true
    }),
    MDBBootstrapModule.forRoot(),
    FormsModule,
    SharedModule,
    CommonModule,
    MaterialModule
  ],
  providers: [
    {
      provide: LOCALE_ID,
      useValue: "es_AR"
    }],
  bootstrap: [AppComponent],
  schemas: [ NO_ERRORS_SCHEMA ]
})
export class AppModule { }
```

Figura 25: AppModule del front-end de “La Justa”

Servicios: Angular 2 también cuenta con una estructura separada de los componentes con el acceso a los datos llamada servicio. Estos pueden contener datos estáticos, datos que vengan desde una API o backend o datos de websockets entre otros. Debe ser una clase con un propósito limitado y bien definido. Angular hace una distinción entre los componentes y los servicios para aumentar la modularidad y la reutilización.

Conceptualmente, los componentes se encargan de la experiencia del usuario, mediando entre la vista y la lógica de la aplicación y delegan ciertas tareas a los servicios, como obtener por ejemplo datos desde un servidor o validar la información ingresada por un usuario. Los servicios son inyectables, lo que hace que estén disponibles para cualquier componente. Estos principios facilitan la integración de la lógica de la aplicación en los servicios y hace que dichos servicios estén disponibles para los componentes a través de inyección de dependencia.

Enrutamiento: Angular 2 ofrece la facilidad de enrutamiento, que es un servicio que permite definir una ruta de navegación entre los diferentes estados de la aplicación y ver su jerarquía. Estas pueden variar al clickear en un botón, hacer click en un enlace o ingresar una url en la barra de direcciones del navegador. Cuando un usuario realiza una acción, por ejemplo click en un enlace, que por defecto cargaría una nueva página en el navegador, el enrutador intercepta el comportamiento del navegador y muestra u oculta las jerarquías de vista. El enrutador registra cualquier actividad en el historial del navegador, para que los botones de retroceso y avance del navegador también funcionen.

A partir de Angular 2, comenzaron a salir versionados consecutivos, manteniendo la mismas características y agregando mejoras continuas, al momento de realizar el desarrollo de esta

tesina la versión 9 era la más nueva, ofrece multi idiomas y utiliza la última versión de Typescript.

Angular Cli

Angular 2 ofrece una herramienta llamada Angular Cli que permite crear, inicializar, depurar y publicar desde la línea de comandos. Además, Angular Cli es muy potente, versátil y sencilla de utilizar, simplificando el desarrollo y brindando una rápida generación de archivos y estructura de componentes. En nuestro caso la versión utilizada fue la 9.0.5.

Angular y Angular CLi dependen de las funcionalidades y características que proporcionan las bibliotecas disponibles en los paquetes npm (Node Package Manager). Por ello, para poder disponer de estos paquetes, es necesario instalar NodeJs que es un entorno de ejecución de JavaScript orientado a eventos asíncronos y está diseñado para crear aplicaciones escalables. Cuando usamos Node.js, éste nos provee la posibilidad de instalar módulos nuevos ya que al ser un sistema fuertemente modular viene prácticamente vacío. Así que, para la mayoría de las operaciones de nuestra Tesina fue necesario instalar módulos adicionales. Esta operación se realiza de forma muy sencilla con la herramienta npm (Node Package Manager).

Una vez instalado Angular Cli, generamos el front-end de nuestro proyecto con el siguiente comando:

- ng new laJusta-frontend

Con el comando anterior se crea e inicializa automáticamente la aplicación Angular, nuestro front-end: se agrega un nuevo espacio de trabajo en una carpeta con el nombre **laJusta-frontend**, como se visualiza en la figura 26, se crea una carpeta interna llamada **src** que es donde se va a alojar el código fuente y varios archivos mas en el que se destaca el llamado package.json que contiene toda la información de los paquetes instalados, por defecto, para el funcionamiento de la aplicación. Pero para algunas funcionalidades requeridas se necesita instalar paquetes adicionales, siendo uno de los más importante los paquetes para manejar la parte responsive y los distintos tamaños en los que se va a visualizar la aplicación, el estilo de los distintos componentes, la reproducción de videos, el manejo de JWT (JSON Web Token) y la librería de los mapas.

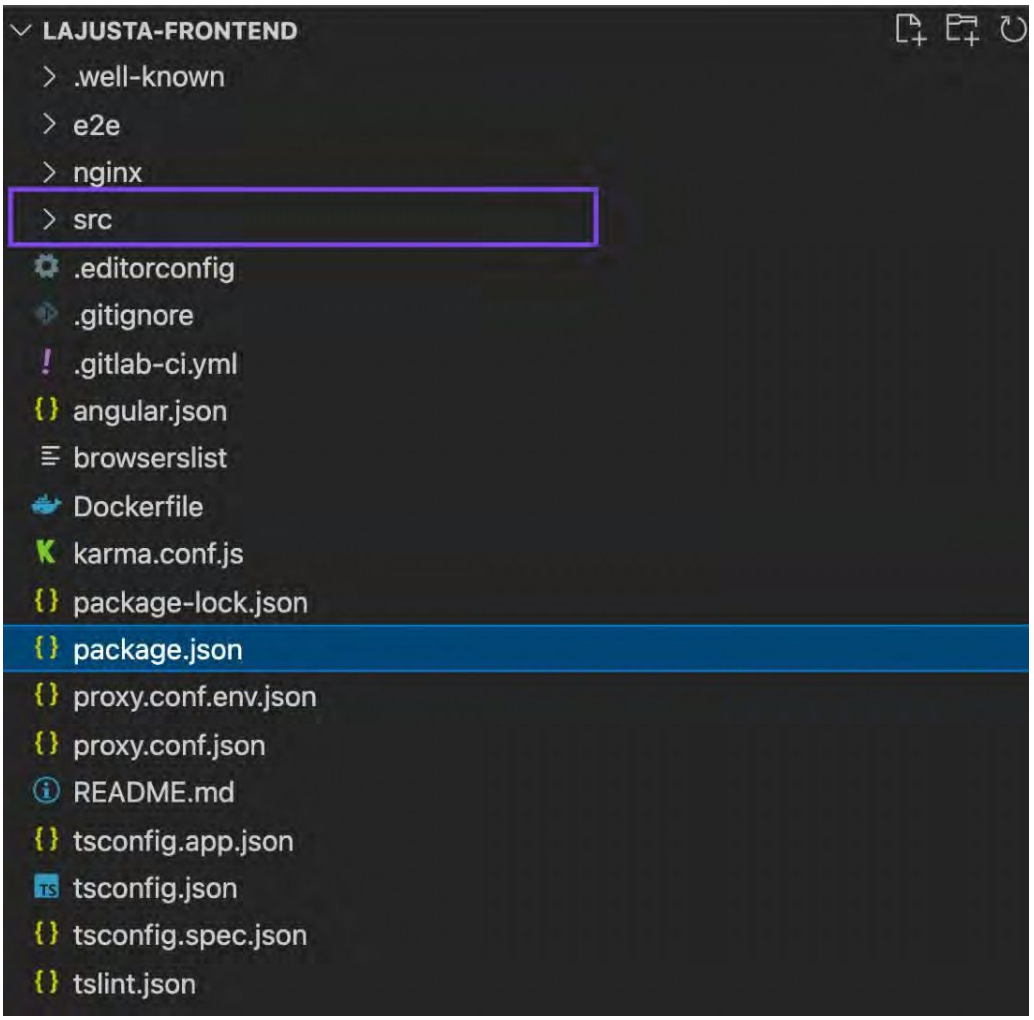


Figura 26: Estructura del proyecto frontend

Frameworks analizados para el desarrollo del front-end

React

Se trata de una librería JavaScript de código abierto, declarativa, eficiente y flexible para construir interfaces de usuario en una SPA. React está mantenida por Facebook y una comunidad de desarrolladores. Al igual que Angular está basada en componentes, crea componentes encapsulados que manejan su propio estado. A diferencia de Angular, la lógica de las mismas está escrita en JavaScript y no en plantillas, esto hace que se puedan pasar datos de forma sencilla a través de la aplicación y mantener el estado fuera del DOM. React está pensado para construir la interfaz de una aplicación en que los datos cambian todo el tiempo al estar apuntado a las vistas. Para ello cuenta con un método render que se encarga de retornar una descripción de lo que se quiere ver en la pantalla y React se encarga de tomar dicha descripción (generalmente es un elemento de React) y mostrar el resultado. La mayoría de desarrolladores de React usan un lenguaje especial de renderizado llamado JSX (JavaScript XML) que facilita la escritura de estas estructuras. JSX mezcla JavaScript con HTML, siendo luego traducido a JavaScript puro sin alterar su semántica. Distinta es la forma en la que

Angular genera el Javascript puro para que éste sea interpretado por los navegadores web, ya que no cuenta con un método como el render que inserta los elementos que se quieren ver en pantalla, sino que estos son llamados generalmente como tags de HTML haciendo referencia a las componentes que están compuestas por plantillas que contienen código en Typescript.

Es importante señalar que React es una librería y no un framework, a diferencia de Angular 2 que cuenta con un ecosistema para el desarrollo en el que además de componentes, cuenta con servicios, enrutamientos y módulos entre otros elementos.

Por otro lado, al ser una librería React es muy utilizada por la forma de trabajo y rapidez en la que la información se actualiza. En nuestro caso, para el desarrollo del portal de "La Justa", la adopción de React hubiese implicado agregar un gran número de librerías adicionales y de terceros, como por ejemplo: Redux[49] que ayuda en la gestión de estado del front-end de la aplicación web o React Router DOM[50] que es una librería para crear la navegación de una SPA en React

Vue.js

Es un framework progresivo, open source de JavaScript, que permite construir interfaces de usuarios de una forma muy sencilla. Vue.js fue lanzado en el año 2014 y puede utilizarse importando la librería principal de Vue.js mediante un CDN (Content Delivery Network o Red de Distribución de Contenido en español). Fue creado por Evan You, ex empleado de Google e integrante del equipo de desarrolladores de Angular. Aunque inicialmente fue pensado para ser una biblioteca personal, la comunidad hizo que el proyecto creciera a un ritmo impresionante, posicionándolo hoy en día como uno de los frameworks web más populares, junto con Angular y React. Al igual que Angular trabaja con componentes. A diferencia de otros frameworks monolíticos, como Angular o Ember, Vue.js está diseñado desde cero para ser utilizado incrementalmente. La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, con Vue.js es posible implementar SPA muy eficientes cuando se utiliza en combinación con herramientas modernas y librerías de apoyo, como vue-router para el enrutamiento o vuex para el manejo de estado. Vue.js comparte muchas similitudes con Angular JS 1.5 (una de las primeras versiones de Angular), en su página web principal (VueJS©, 2014-2022) podemos encontrar las similitudes y diferencias entre estas dos librerías, como ser la sintaxis, pero por otro lado como diferencias tenemos la simplicidad de Vue.js, la libertad que le da al programador a la hora de modularizar una aplicación, el tipo de bindeo (vinculación) que hace entre las variables mientras que Angular es bidireccional, en Vue.js se enfoca más en un solo sentido. En términos de performance, AngularJS 1.5 es

[49] https://redux.js.org/
[50] https://v5.reactrouter.com/

un framework con ya sus años en el mercado y que fue reemplazado y mejorado por Angular 2, por otro lado Vue.js es un framework vigente y que sigue recibiendo mejoras constantemente, por lo tanto es lógico que su performance es superior a la de AngularJS.
En nuestro proyecto, Vue.js fue descartado porque no contamos con experiencia en el framework , nos sentimos más seguros con Angular, sin embargo entendemos que podría haber sido una gran opción al ser un framework progresivo, partiendo de una librería central que va agregando solo lo necesario, lo que hace que sea más liviano que Angular. Esta última característica fue muy valorada al momento de elegir el framework para el frontend, pero apostamos por la experiencia, dado que como venimos comentando era prioritario contar con un portal en producción rápidamente.

Manejo de autenticación y permisos

En el front-end se desarrolló un servicio de autenticación para manejar el login, el logout y para chequear si el usuario está autenticado y tiene permisos para ciertas acciones o accesos a ciertas páginas. Para llevar a cabo estas acciones y verificar los permisos de acceso, se usaron las siguientes tecnologías: tokens JSON Web Token (JWT), interceptores (JwtInterceptor) y guards, descriptas a continuación.

Token JWT

Los tokens web JSON son un estándar abierto para crear token de acceso que permiten propagar la identidad y los privilegios entre dos partes. Por ejemplo, cuando enviamos una solicitud de inicio de sesión al portal, el backend nos enviaría un token que indica que se ha establecido una conexión segura entre el frontend y el backend. Con cada solicitud que el frontend envía al backend, es necesario adjuntar dicho token. El token puede contener diferentes tipos de información y también puede caducar. La información que contiene nuestro token, aparte de los datos del usuario logueado, son el rol y si el token está activo, es decir, no caducó al momento de realizar una petición al servidor. Con el rol manejamos los permisos, actualmente contamos con dos roles activos en el proyecto de “La Justa” que vienen codificados en el token, el usuario consumidor, que es el que puede realizar compras y el usuario administrador que además de poder comprar, puede acceder a la sección de configuración del sitio web. También hay un rol que no necesita del token que es el usuario anónimo, éste sólo tiene acceso a la página principal, a ver el catálogo y a armar un carrito, pero al momento de realizar una compra se le va a solicitar que se autentique o se registre para poder efectivizar la compra, dado que las compras pueden ser realizadas sólo por usuarios registrados. Si un usuario autenticado quisiera realizar cualquier acción y el token estuviese caducado, el servidor responderá con un error y se le solicitará al usuario que vuelva a autenticarse.

JwtInterceptor

De acuerdo con la documentación de Angular: “Un interceptor maneja una HttpRequest o HttpResponse antes de pasar al siguiente interceptor en la cadena, llamando a next.handle

(transformReq)". Entonces, lo que hace un Interceptor básicamente es "escuchar" una solicitud o una respuesta y transformar los datos o producir un efecto secundario. En nuestro caso, usamos interceptores para verificar que un usuario se encuentre logueado al realizar algunas acciones como comprar o validar que el token de autenticación no haya caducado. Las figuras 27 y 28 son ejemplos de usos de interceptores en el front-end del portal.

Interceptor HttpRequest

En el frontend del portal de "La Justa" se implementaron interceptores de petición, como se describe en la figura 27, que escucha peticiones HttpRequest y adjunta el token obtenido desde el servicio de autenticación, luego de verificar que haya un usuario logueado para agregarlo al encabezado de la solicitud.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor, HttpResponse } from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';
import { tap, map } from 'rxjs/operators';


@Injectable()
export class JwtInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        // add authorization header with jwt token if available
        if (this.authenticationService.isLoggedIn()) {
            request = request.clone({
                setHeaders: {
                    Authorization: `Bearer ${this.authenticationService.currentUserValue.value}`,
                    'Set-Cookie': 'HttpOnly;Secure;SameSite=Strict'
                }
            });
        }

        return next.handle(request).pipe(
            map((event: HttpEvent<any>) => {
              if (event instanceof HttpResponse) {
                // do stuff with response and headers you want
                event.clone({
                    headers: request.headers.set('Set-Cookie', 'HttpOnly;Secure;SameSite=Strict')
                });
              }
              return event;
            })
          );
    }
}
```

Figura 27: Interceptor HttpRequest: escucha peticiones y adjunta el token

Entonces, este interceptor obtiene el token del usuario registrado (this.authenticationService.currentUserValue.value) y lo adjunta al encabezado de la solicitud. De esta manera, el backend puede validar la solicitud y enviar al frontend el resultado.

Interceptor HttpResponse

El segundo interceptor, figura 28, escuchará cualquier respuesta entrante y luego responderá en consecuencia. Supongamos que el token que ha generado el backend caduca, por lo que, aunque enviamos la solicitud y adjuntamos correctamente el token en el

encabezado de la llamada, el backend nos trae un error 401 (usuario no autorizado). En este caso, deberíamos detectar este error y cerrar la sesión del usuario automáticamente.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { AuthService } from '../services/auth.service';


@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(request).pipe(catchError(err => {
            if (err.status === 401) {
                // auto logout if 401 response returned from api
                this.authenticationService.logout();
                //location.reload(true);
            }
            if (err.status === 440) {
                console.log('token');
                return;
            }
            console.log('error', err.error.message           );
            const error = err.error.message || err.statusText;
            return throwError(error);
        }))
    }
}
```

Figura 28: Interceptor HttpResponse: recibe la respuesta del servidor que evalúa si está autenticado

Los interceptores son compartidos en toda la aplicación, por ello se definieron en el **Módulo SharedModule**, como se muestra en la figura 29.

```
    providers: [
        AuthService,
        AuthGuard,
        AdminGaurd,
        FormBuilder,
        CurrencyPipe,
        DatePipe,
        KeyValuePipe,
        UpperCasePipe,
        LoaderService,
        { provide: HTTP_INTERCEPTORS, useClass: LoaderInterceptor, multi: true },
        { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
        { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }
    ]
})
export class SharedModule { }
```

Figura 29: SharedModule: para uso de interceptores

También implementamos guards, que son un mecanismo que nos provee Angular para permitir o no el acceso a las distintas partes del sitio de acuerdo al tipo de usuario.

Guards

Como hemos descripto anteriormente, diferentes usuarios tendrán diferentes accesos en la aplicación por eso implementamos la interfaz **canActivate** desde @angular/router que decidirá si se puede acceder a una ruta o no.

AdminGuard

En la figura 30 se describe la implementación del AdminGuard, el cual se encarga de verificar si el usuario tiene permiso para acceder al Panel de configuración como administrador. Para este caso, el backend guarda el rol en el token, lo que significa que el valor de nuestro token será usuario (consumidor) o administrador.

```
import { Injectable } from "@angular/core";
import { Router, CanActivate } from "@angular/router";
import { AuthService } from "../services/auth.service";

@Injectable()
export class AdminGaurd implements CanActivate {
  constructor(private router: Router, private authService: AuthService) {}

  canActivate() {
    if (this.authService.isLoggedIn() && this.authService.isAdmin()) {
      return true;
    }
    this.router.navigate(["no-access"]);
    return false;
  }
}
```

Figura 30: Admin Guard.

Para que los accesos funcionen de acuerdo a los permisos, es necesario agregar el **canActivate** al módulo de enrutamiento. Como se muestra en la figura 31, el acceso a la url de config, en el archivo app-routing.module, está supeditado al tipo de usuario, es decir, solo puede acceder a dicha URL siendo administrador.

```
const routes: Routes = [
  {
    path: 'config',
    canActivate: [AdminGaurd],
    loadChildren: () => import('./configuration/configuration.module')
      .then(m => m.ConfigurationModule),
  },
  {
    path: 'shop', loadChildren:() => {return ShopModule;}
  /*  loadChildren: () => import('./shop/shop.module')
      .then(m => m.ShopModule)*/
  },
  { path: '', redirectTo: 'shop', pathMatch: 'full' },
        { path: 'no-access', component: NoAccessComponent },
        { path: '**', component: PageNotFoundComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { useHash: false, scrollPositionRestoration: 'enabled' })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Figura 31: Ejemplo configuración de accesos con Guard.

TECNOLOGÍAS DE DESPLIEGUE Y MANEJO DE CÓDIGO

Herramienta adoptada para el manejo de dependencias: GRADLE

En todo desarrollo de software, ya sea móvil, web o de escritorio, es necesario el manejo de dependencias así como el poder manejar la compilación. Compilar un archivo con un compilador como Java es simple, con solo correr el javac y pasarle los parámetros necesarios es suficiente, pero la complejidad se da cuando tenemos más de un archivo Java, los proyectos medianos como el de “La Justa” llegan a tener cientos, y todos ellos comienzan a utilizar librerías de terceras partes[51]. En estos casos necesitamos de alguna herramienta que nos permita automatizar la compilación de nuestros archivos fuente sin la necesidad de preocuparnos por el uso de estas librerías y saber que nuestro ejecutable resultante no necesita ninguna dependencia extra para ser ejecutado. En estos casos es donde herramientas como Gradle[52] entra en escena.

Gradle tiene 2 tareas principales, la primera es descargar todas las dependencias que tenemos en nuestro proyecto, y esto se logra con definir en el apartado de **dependencies** cuales son los proyectos que queremos utilizar, como vemos en la figura 32.

```
dependencies {
  implementation("org.springframework.boot:spring-boot-starter-web")
  implementation("org.springframework.boot:spring-boot-starter-data-jpa")
  implementation("org.springframework.boot:spring-boot-starter-jdbc")
  implementation("org.springframework.boot:spring-boot-starter-security")
  implementation("io.jsonwebtoken:jjwt:0.9.0")
  implementation("io.springfox:springfox-swagger2:3.0.0")
  implementation("io.springfox:springfox-swagger-ui:3.0.0")
  implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
  implementation("org.flywaydb:flyway-core")
  implementation("org.jetbrains.kotlin:kotlin-reflect")
  implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
  implementation("com.google.guava:guava:20.0")
  implementation("org.mariadb.jdbc:mariadb-java-client")
  implementation("commons-io:commons-io:2.7")
  implementation("javax.mail:javax.mail-api:1.6.2")
  implementation("com.sun.mail:javax.mail:1.6.2")
  implementation("log4j:log4j:1.2.17")
  implementation("org.apache.poi:poi:3.15")
  implementation("org.apache.poi:poi-ooxml:3.15")
  implementation("org.apache.poi:ooxml-schemas:1.3")
  implementation("org.apache.pdfbox:pdfbox:2.0.21")
  implementation("com.github.dhorions:boxable:1.6")
  testImplementation("org.springframework.boot:spring-boot-starter-test")
  {
    exclude(group = "org.junit.vintage", module = "junit-vintage-engine")
  }
}
```

Figura 32: Listado de dependencias de Gradle.

[51] Este tipo de librerías son las que desarrolla otra persona o equipo y se empaquetan de tal modo que es posible descargar archivo JAR de esa librería y hacer uso de toda la funcionalidad que trae.

[52] https://gradle.org/

Gradle busca y descarga los proyectos, bajo el apartado **dependencies**, en sus versiones JAR y los ubica en una carpeta **/.gradle** dentro de la raíz del proyecto. Los archivos son descargados del repositorio Maven[53], el cual tiene infinidad de librerías para descargar. Una vez descargados, Gradle además ofrece la posibilidad de compilar nuestra aplicación y generar el archivo JAR que más tarde va a ser ejecutado. Todo esto siguiendo la configuración que se indica en el archivo **build.gradle**; la figura 33 muestra cómo configurar Gradle para adaptar nuestro JAR a distintas versiones, o cambiarle el nombre.

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
  id("org.springframework.boot") version "2.3.0.RELEASE"
  id("io.spring.dependency-management") version "1.0.9.RELEASE"
  kotlin("jvm") version "1.3.72"
  kotlin("plugin.spring") version "1.3.72"
  kotlin("plugin.jpa") version "1.3.72"
}

group = "edu.unlp.la.justa"
version = "0.0.1-SNAPSHOT"
java.sourceCompatibility = JavaVersion.VERSION_1_8

repositories {
  mavenCentral()
}

dependencies {
  // Dependencias
}

tasks.withType<Test> {
  useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
  kotlinOptions {
     freeCompilerArgs = listOf("-Xjsr305=strict")
     jvmTarget = "1.8"
  }
}
```

Figura 33: Configuración de Gradle del proyecto de “La Justa”

Como podemos observar, nuestro archivo de configuración dice que Gradle va a generar un JAR llamado **edu.unlp.la.justa-0.0.1-SNAPSHOT.jar**, que la versión mínima de la JVM requerida para ejecutar este proyecto es la 1.8 y que va a descargar todas las dependencias del repositorio central de Maven. Esto es posible porque dichos repositorios son públicos y cualquier herramienta puede hacer uso de ellos.

Gradle es una herramienta fundamental para el desarrollo de cualquier tipo de aplicación, dado que todos los IDEs soportan Gradle y saben cómo importar el proyecto, esto nos da la

[53] https://mvnrepository.com/

libertad de no quedar atados a un IDE en particular, además de todas las ventajas antes mencionadas en el manejo de dependencias y la compilación.

Herramientas evaluadas para el manejo de dependencias

Maven

Maven[54] es un producto de Apache, y si bien cumple la misma función que Gradle, básicamente su única diferencia es que el POM se escribe con XML y el gradle.build con JSON. Maven se comenzó a utilizar mucho antes que Gradle y actualmente está vigente.

Al igual que Gradle, Maven tiene un pequeño ejecutor que se puede integrar a nuestro proyecto, de esta manera con solo agregar un ejecutable de Maven llamado mvnw, podemos integrar nuestra aplicación sin la necesidad de tener instalado Maven en cada una de las PCs de los desarrolladores o bien de los sistemas de integración continua. Este ejecutable puede verse como un Maven embebido en la aplicación, haciendo super portable a Maven y permitiendo ejecutar todo el flujo de descarga de dependencias o armado de nuestro proyecto con solo seleccionar el código del repositorio. Esto aunque parezca una tontería es importante porque reduce los tiempos de "set-up"[55] de un nuevo integrante, reduciendo la cantidad de herramientas que debe instalar en el proyecto.

La decisión de no utilizar Maven fue que Gradle nos permite escribir el gradle.build con código Kotlin, lo cual nos pareció divertido de experimentar. Pero a decir verdad el proyecto podría haber sido desarrollado usando Maven sin ningún problema, no existe ninguna característica técnica que vuelva mejor a Gradle por sobre Maven.

Herramienta adoptada para el despliegue: GitLab

Para el versionado de código utilizamosde GitLab[56], como toda herramienta de Git, nos permite crear *branches*, *commitear* y *pushear* nuestro código al repositorio sin agregar mayores beneficios que otras plataformas, entonces, ¿por qué la elegimos? GitLab tiene características que no están directamente relacionadas con un repositorio de Git y que inclinaron la balanza a su favor, como puede ser el ejemplo de los *pipelines* de los que nos ocuparemos en la siguiente sección. Gracias a que existen los pipelines podemos automatizar tareas tales como la compilación de código, creación de imagen Docker, subida de esta imagen a la nube, descarga en el servidor, borrado y creación de contenedores basándonos en las nuevas imágenes. Todo esto simplemente mergeando nuestros cambios a un branch en particular al que llamamos **producción.**

Otra de las características que nos ofrece GitLab que se debatió, es la facilidad de utilizar un "container registry", el cual funciona como un repositorio privado de imágenes Docker. Esto

---

[54] https://maven.apache.org/

[55] set-up significa tener todo el entorno local listo y preparado para desarrollar.

[56] https://gitlab.com/

en su momento fue una alternativa a docker hub[57], pero finalmente la dejamos de lado dado que los lenguajes y tecnologías que usamos son open source, entonces ¿por qué poner nuestro producto final en un servidor privativo en lugar de tenerlo en un repositorio público y abierto?.

Actualmente utilizamos 2 de las funcionalidades que ofrece GitLab, el repositorio de código y las actions que nos permiten desplegar código sin intervención humana, sin embargo GitLab no es solo un repositorio, sino un ecosistema donde podemos delegar muchas de las tareas que si bien pueden hacerse con otras herramientas, algunas de éstas son pagas o difíciles de instalar/usar.

Por último, nuestro repositorio cuenta con 2 branches: **master** y **producción**, uno de los cuales tiene los cambios compartidos (master), mientras que el otro tiene los cambios que están subidos a producción, así como alguna que otra configuración (por ejemplo swagger deshabilitado). Siempre que comenzamos con una nueva funcionalidad, se crea un branch a partir de **producción**, se trabaja, se hace un merge contra master para compartirlo con otros desarrolladores y una vez que se está seguro de que la funcionalidad está terminada, entonces se *mergea* contra el branch **producción** de nuevo, disparando así algunos procesos de compilación y subida al servidor que vamos a tratar en la siguiente sección. En la figura 34 se describe el flujo de desarrollo en un branch llamado "**feature/nuevo_carrito_de_compras**" (paso 1) que más tarde se mergea con el branch **master** (paso 2) y finalmente con el branch **producción** (paso 3) y así la funcionalidad estará disponible en el portal de "La Justa".

[57] https://hub.docker.com/

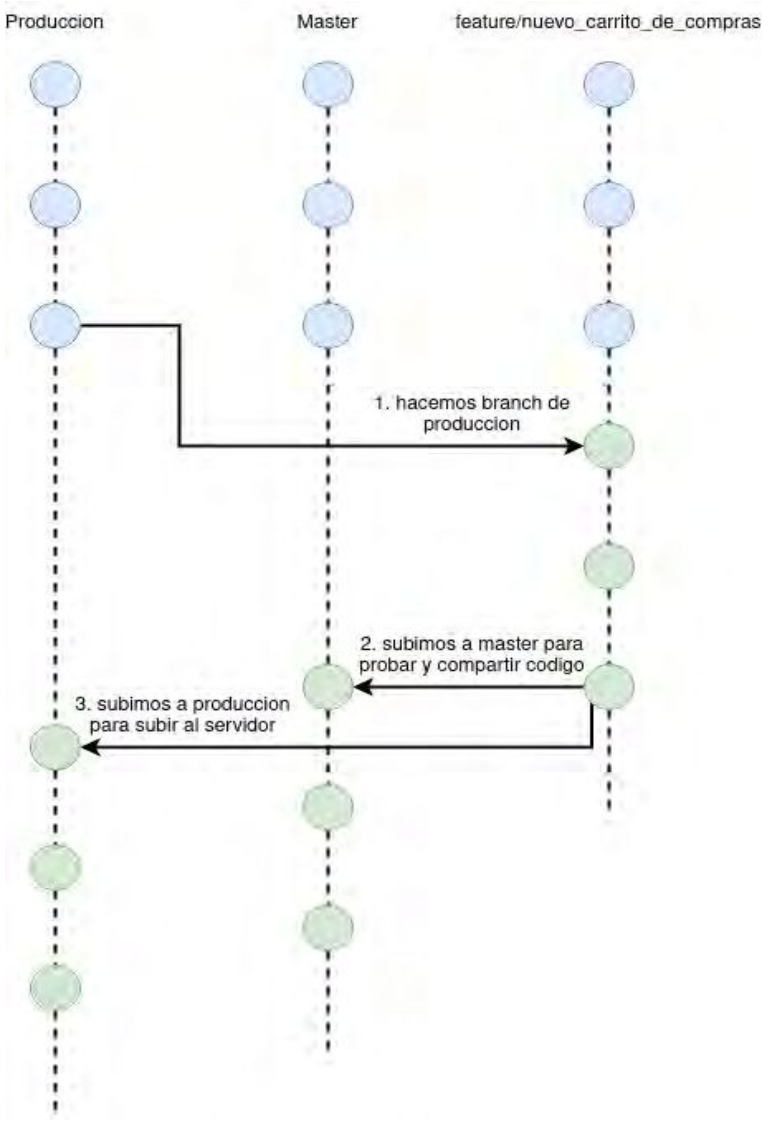


Figura 34: Flujo de branches en GIT

GitLab PIPELINES

Actualmente la industria del desarrollo de software invierte mucho dinero en reducir los errores producto de la intervención humana. Cuanto más repetitiva es una tarea, cuantas más personas involucradas trabajan sobre el mismo proceso, o bien, cuando no existe un proceso definido para lograr un objetivo en común, es donde aparecen los errores. Los humanos podemos equivocarnos por distintas razones: podemos desconocer un paso importante en un deploy, podemos desconocer la herramienta que estamos utilizando, sus estándares y buenas prácticas, podemos confiarnos en que una tarea que realizamos todos los meses del mismo modo va a salir siempre bien dado que “lo hacemos todos los meses de la misma manera” o bien, el stress y las horas de trabajo pueden jugarnos una mala pasada. Pero, ¿cuáles son los errores a los que nos referimos?, una puesta en producción o un deploy en un ambiente de desarrollo o testing involucra varios pasos, los más comunes que podemos destacar son el deploy de nuestro código en un servidor (o en caso de que nuestro componente sea serverless, el arranque del mismo) y la ejecución de todos los

scripts en la base de datos correspondiente. Estos dos son los pasos más comunes en la gran mayoría de los escenarios del mercado, pero existen otros no tan frecuentes: imaginen el correcto seteo de herramientas como ser colas de mensajes o un bucket para poder almacenar imágenes estáticas, otro punto podría ser tener más de un servidor corriendo detrás de un balanceador de carga, replicar el mismo deploy en muchas instancias. Estos son algunos de los problemas a los que nos podemos enfrentar a la hora de subir nuestro código a un ambiente de producción. Además, estos problemas podrían ser divididos en otros posibles problemas dentro de su categoría. En nuestro proyecto de Tesina podemos identificar los siguientes problemas en las diferentes capas de la arquitectura propuesta, front-end, back-end, bases de datos::

Problemas en la puesta en el despliegue del Front-End

Un error muy común es deployar una versión más nueva en front-end pero mantener una vieja en el back-end, generando una pobre experiencia para el usuario, al que nunca le dejan de aparecer mensajes de error en la pantalla. Otro punto propenso a error se encuentra a la hora de minificar y *uglificar*[58] el proyecto para su próxima subida al servidor. Tenemos que estar atentos que la URL del backend que estamos queriendo consumir sea la del ambiente correspondiente, nada peor que testear funcionalidad en un ambiente de prueba para más tarde darnos cuenta de que nuestro servidor de front-end está apuntando a producción, y todos los cambios que hicimos en realidad fueron hechos en producción.

Problemas en la puesta en el despliegue del Back-End

En los servicios de back-end los problemas con los que nos podemos encontrar son aún más complejos. Así como en el front-end podemos subir al servidor una versión más moderna y desactualizar el sitio web, en el back-end puede suceder algo similar con la base de datos. Por ejemplo, es posible realizar cambios en una entidad modelo de nuestro código y olvidarnos de correr los scripts correspondientes en la base de datos para agregar dichas columnas en la tabla correspondiente, resultando de esta manera en que nuestros servicios de back-end nunca arranquen. Otro problema muy común es asegurarnos que nuestro back-end está apuntando a la base de datos del ambiente en cuestión. Otro problema que podemos encontrarnos en el back-end de “La Justa” es apuntar a un servidor de imágenes diferente, el servidor de imagenes que usamos nosotros nos permite almacenar fotos de los productos y productores, apuntar a un servidor de prueba en producción implicaría mostrar fotos que no son las reales en el sitio a los usuarios finales, algo que claramente no queremos hacer.

Por último, pero no por eso menos importante, existe un problema en común entre los dos ambientes, tanto en front-end como en back-end. Trabajando con una herramienta de versionado de código es muy probable que a la hora de subir nuestros cambios a algun

[58] uglificar: es un término utilizado en la compilación de aplicaciones a JavaScript, su idea principal es poder cambiar el nombre de las variables por términos más difíciles de leer, esto tiene como objetivo dificultar la lectura del código JavaScript para evitar su modificación.

servidor nos encontremos con funcionalidad en progreso, esto presenta un gran problema, dado que podemos exponer un menú que todavía no está diseñado, un ruteo a una vista del front-end que no existe, o servicios de back-end que no van a ser consumidos pero aún pueden ser referenciados por alguna herramienta que ejecute llamas HTTP, como SOAP-UI o Postman.

Problemas en la puesta en el despliegue del Base de datos

Por último los problemas con la base de datos, suelen ser pocos pero recurrentes. La base de datos debe mantenerse actualizada en todo momento y con la habilidad y versatilidad suficiente como para poder realizar un rollback en cualquier momento que sea necesario.
Dicho esto y habiendo señalado algunos de los posibles errores a los que podríamos enfrentarnos en una actualización de código en los servidores de “La Justa”, llegamos a la conclusión que el proceso de deployment es uno de los más críticos en el ciclo de vida de un sitio web y en general es uno de los lugares donde estamos más vulnerables a los errores, ¿cuáles son las prácticas que actualmente adopta la industria de desarrollo de software para mitigarlos?
Hoy en día existe una amplia gama de herramientas para automatizar procesos repetitivos. Como mencionamos antes, necesitamos poder subir nuestros cambios a producción de la misma manera una y otra vez, actualizar la base de datos antes de poner en marcha un servidor back-end, mantener todas las credenciales y URL de los distintos componentes consumidos por nuestros servicios. Todo esto es delegado a herramientas de integración continua y delivery continuo. Una de las más conocida es Jenkins, con 10 años en la industria, también existen otras alternativas quizás más modernas y más simples, como ser GoCD, Code Pipeline de AWS, CircleCI, GitHub actions y la elegida para este proyecto: GitLab pipelines. En la figura 35 podemos ver el resultado de la ejecución de un pipeline en GitLab, la salida de consola es redirigida y podemos verla desde la Web.

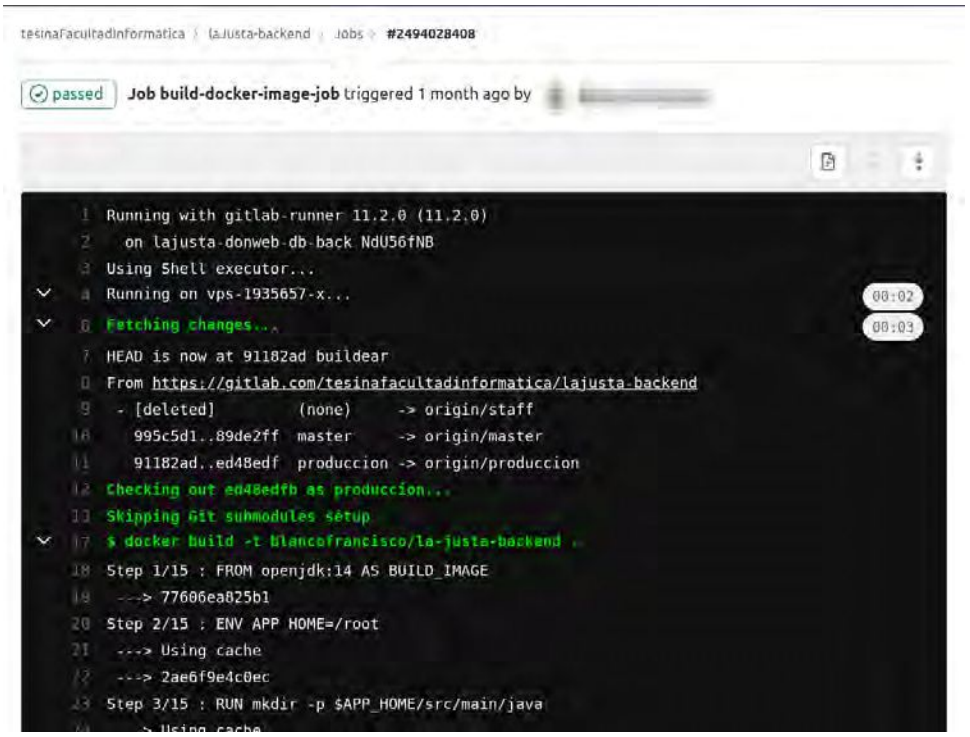


Figura 35: ejecución de stage de un pipeline para compilar el backend usando docker

ANÁLISIS DE HERRAMIENTAS DE INTEGRACIÓN CONTINUA

GitLab Runner

Así como CircleCI, Gitlab Runner es un “task runner”. La idea principal de estas herramientas es instalar un “runner”[59] en el servidor dedicado a integración continua, luego a partir del archivo de configuración **gitlab-ci.yam** se indican todos los pasos que se deben ejecutar.

Por otra parte, en la UI de Gitlab, es posible observar todas las ejecuciones de nuestros pipelines[60], como así cada una en más detalle. En la figura 36 se puede muestra el historial de ejecución de los distintos pipelines, así como su estado. Cuando alguno falla se avisa por email a los desarrolladores. En la figura 37 se muestra más en detalle el estado de cada uno de esos pipelines ejecutados.

[59] Un ejemplo de qué es y cómo instalar un runner para GitLab, está disponible en esta url:ps://docs.gitlab.com/runner/

[60] En la industria nombramos pipeline a una serie de acciones que se ejecutan una detrás de la otra siempre en el mismo orden, dependiendo unas de otras y trabajando en conjunto con el único fin de tener la tarea terminada, por separado no funcionan dado que son acciones muy chicas, pero en el conjunto nos permite automatizar procesos repetitivos.

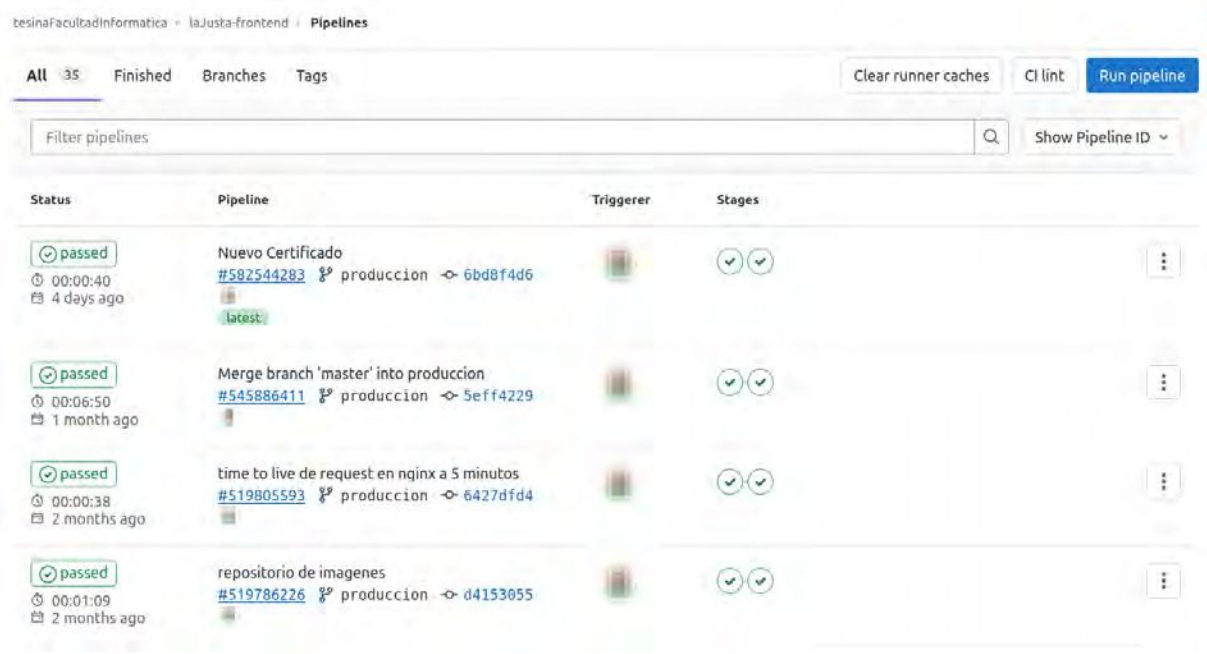


Figura 36: Imagen que nos muestra el panel de control de gitlab pipelines

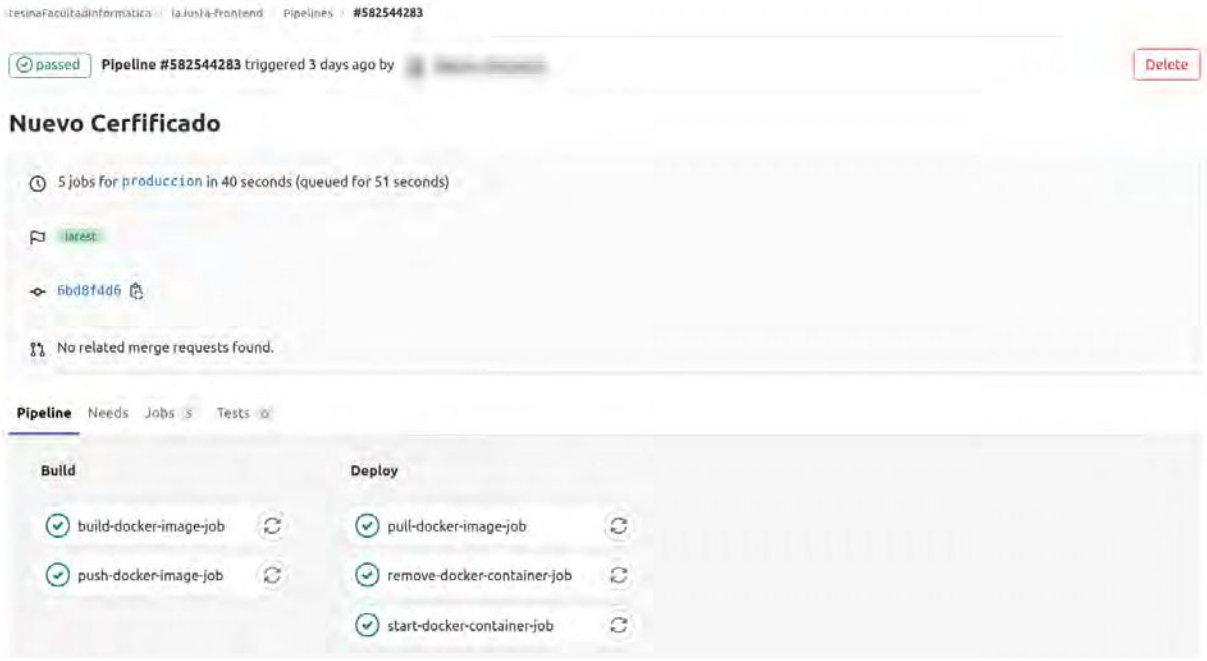


Figura 37: Detalle de la ejecución de un solo pipeline

Es importante destacar que GitLab ofrece runners comunes listos para usar, algunos están optimizados para Python, otros para Node, Doker, Maven, etc, lo único que se necesita hacer es buscar el nombre y agregarlos en **tags.** Esta es una facilidad excelente, porque nos permite tener un flujo de build, test y deploy completo, que además no requiere de pago para su uso.

Para cerrar, compartimos un gráfico que ilustra la manera en que el portal de “La Justa” es deployado en producción, mediante una nueva imagen Docker de back-end y front-end mergeando una historia al branch de producción. GitLab detecta el cambio e inmediatamente dispara la creación de la imagen Docker correspondiente en la máquina virtual del hosting (DonWeb), luego la sube a docker hub, por último el “runner” que se

encuentra instalado en la máquina virtual donde tenemos los servicios, descarga la nueva imagen, mata al contenedor actualmente corriendo, y crea uno nuevo con la nueva imagen, todo esto es automático y sin la necesidad de intervención humana. En la figura 38 se describen los pasos que se llevan a cabo luego de mergear al branch producción.

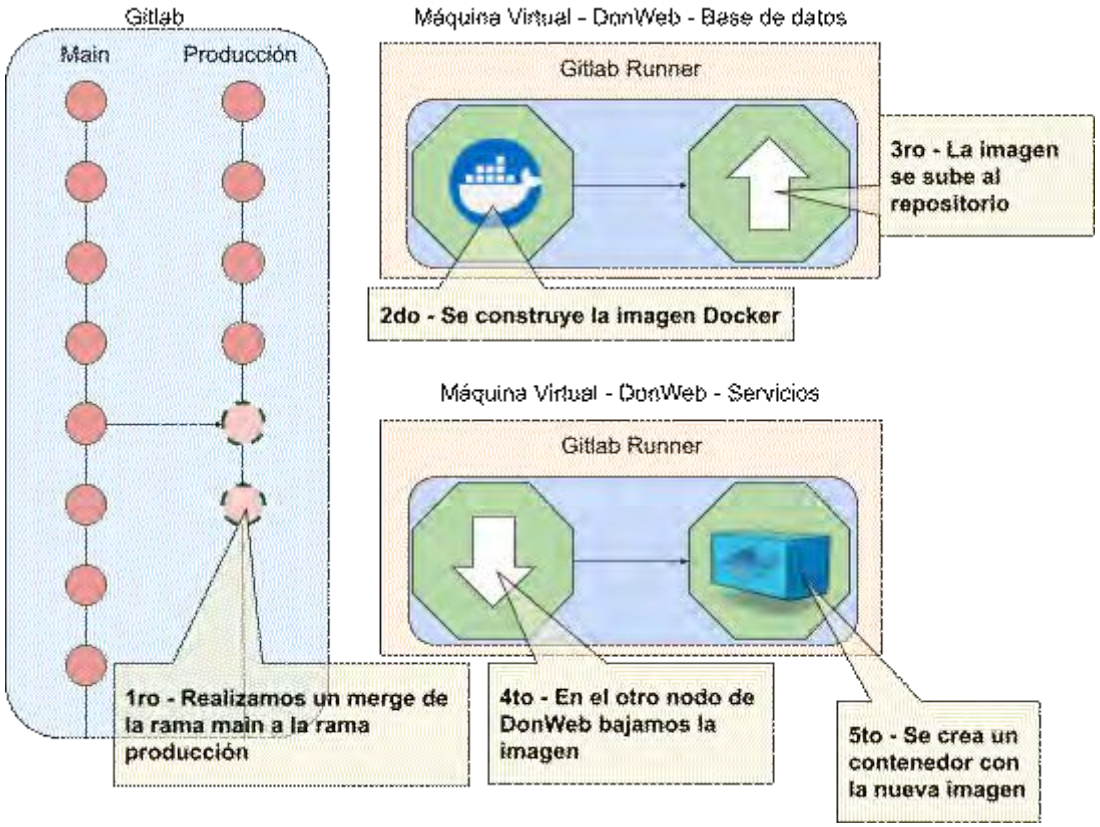


Figura 38: Orden de ejecución de cada tarea una vez se realiza el merge al branch producción

Como conclusión podemos decir que los pipelines no solo nos ayudan a automatizar tareas las cuales son propensas a errores, también nos permiten “mantener a los desarrolladores lejos de los servidores”, ¿Qué significa esto?, en general los equipos de OPS (Operaciones) son bastante reacios a la hora de compartir o brindar accesos a los servidores. Es muy posible que un mal comando usado por un desarrollador en la base de datos de testing o desarrollo pueda detener el trabajo del día de 1, 2 o 5 equipos. Es fundamentalmente que por este motivo, los profesionales de OPS no dan acceso a los servidores y se manejan con sistemas de gestión de “tickets”, que pueden tardar 1 o 2 días en ser resueltos. Con las herramientas de despliegue contínuo es posible desplegar el código en un ambiente con solo mergear los cambios en Git. Esto significa que un desarrollador puede desplegar su código sin conocer el servidor físico en donde se va a desplegar, la URL de dicho servidor, el hosting, o incluso sin tener una sola credencial de acceso.

Github Actions

Comparte mucha similitud con Gitlab pipelines, esta herramienta es también un task runner que podemos configurar para ejecutar distintas acciones luego de un merge o commit en una rama de git en particular. Extremadamente útil a la hora de automatizar tareas, pero un poco más joven que la herramienta que seleccionamos, eso fue un determinante para descartarla dado la madurez de Gitlab pipelines. Github actions fue lanzado en el 2018, es extremadamente potente, soporta la compilación en múltiples sistemas operativos como Windows, Linux y Mac y por sobre todas las cosas se encuentra al tope de las tecnologías a experimentar en un futuro cercano.

Jenkins

Con fecha de lanzamiento en el 2011, Jenkins es una de las soluciones enterprise para el problema de la integración continua de las grandes y medianas empresas. Es un proyecto Open Source[61] realizado en Java y adoptado ampliamente en el mercado. Cuenta con más de 100 plugins que se ajustan a cualquier problema y herramienta, ya sea que nuestra herramienta de buildeo y gestión de paquetes sea Gradle o Maven. ¿Por qué no fue elegido para manejar las puestas en producción de “La Justa”?, simple, Jenkins es un proyecto robusto, confiable y con muchos años en uso, pero la instalación, puesta en marcha y configuración de los plugins necesarios es una tarea bastante ardua y compleja para nuestro problema. Además, Jenkins es pesado y necesitaría una máquina virtual propia donde correr, esto implicaría un costo extra e innecesario para el equipo de “La Justa”.

GoCD

GoCD[62] es otra herramienta bastante similar a Jenkins, es un proyecto open source también realizado en Java que cuenta de 2 partes: los runners, los cuales se instalan en diferentes máquinas y son los responsables de ejecutar los distintos pasos del deploy, y el orquestador, el cual ofrece la UI, se encarga de crear, administrar y disparar todos procesos de deployment. GoCD apunta a ser mas liviano, simple y fácil de adoptar que Jenkins. Si bien GoCD cuenta con un buen número de plugins, es posible utilizarlo sin necesidad de ninguno dado que es posible conectarse a cualquier repositorio GIT, descargar el código, correr los pasos de deploy y notificar por email cualquier fallo con la versión base de GoCD.

[61] https://www.jenkins.io/
[62] https://www.gocd.org/

Al igual que Jenkins, instalar el orquestador y al menos un runner implica un gasto de recursos innecesario, por esta razón tampoco fue seleccionado como herramienta encargada del deployment de “La Justa”.

CircleCi

De acá en adelante nos adentramos más en herramientas del tipo “step runners” más que en herramientas de plugins como lo son Jenkins y GoCD. Este tipo de sitios nos ofrecen la posibilidad de ejecutar una seguidilla de pasos en distintos “runners” provistos por la plataforma, donde podemos descargar el código de GitLab, buildear nuestra nueva versión de Angular o Kotlin, crear el container de Docker y subirla a producción dondequiera que sea, esto sin la necesidad de instalar ningún orquestador, porque la UI es web. Es suficiente con instalar los “runners” en nuestro servidor. Entonces, CircleCI[63] es una herramienta superadora, sin embargo nuestra falta de experiencia en su uso y y los tiempos disponibles, en tanto se necesitaba una solución que estuviese en producción lo más rápido posible, nos llevó a descartarla para “La Justa”.

AWS CodePipeline

Al igual que CircleCI, Code Pipeline es una herramienta cloud que nos permite realizar “steps execution” para descargar el código de GitLab, compilarlo, testearlo y subirlo a cualquier servidor[64]. Es bastante simple de configurar y podemos tener rápidamente nuestro proceso de despliegue integrado a esta herramienta.El problema que detectamos es que está bastante acoplado a la infraestructura de AWS (Amazon Web Services) dicho en otras palabras, es muy fácil de integrar si nuestro servidor corre en una instancia de AWS. Además es una herramienta paga que pertenece al ecosistema de AWS. Estas cuestiones fueron determinantes para descartarla.

DOCKER

Si bien el uso de Docker no era necesario para el tipo de proyecto que estamos utilizando, o dicho de otra manera, un servidor con Nginx instalado hubiese cumplido la misma función, utilizarlo nos dio un conjunto de ventajas que mejoran increíblemente los procesos de despliegue. ¿Qué es Docker?, ¿qué nos ofrece?.

Docker es una herramienta de virtualización de aplicaciones dentro de contenedores, que permite agrupar un elemento de software en una unidad, dicha unidad puede ser compartida

[63] https://circleci.com/

[64] https://aws.amazon.com/codepipeline/

en distintas máquinas y aun así correr sin necesidad de elementos extra, dado que todas sus dependencias se encuentran en el mismo contenedor. Otra de las características de Docker es ser multiplataforma, es decir, un contenedor creado en Windows puede ser ejecutado sin ningún tipo de problema en Linux o Mac.

A diferencia de otras herramientas de contenedores, Docker es “application driven”, esto significa que si nosotros creamos un contenedor basado en un sistema operativo ya sea Arch Linux, Ubuntu o Mint, pero no le especificamos ninguna aplicación para ejecutar dicho contenedor va a finalizar su ejecución al poco tiempo de haberse creado, la razón es que todos los contenedores necesitan de una aplicación para ejecutar, si no, no tiene sentido tenerlos utilizando memoria y CPU solo para correr el sistema operativo de la imagen que usamos para crearlos. Podemos marcar esta funcionalidad como una gran diferencia con las máquinas virtuales, donde es posible tener el sistema operativo en ejecución de modo ocioso.

¿Para qué usamos Docker en el portal de “La Justa”?. Básicamente se delega en Docker las siguientes funciones: buildeo de front-end y back-end, ejecución de los servidores de imagen, back-end y front-end.

Por otro lado, la base de datos se encuentra corriendo instalada en la máquina virtual, no en un contenedor. En nuestra experiencia, si bien existen miles de contenedores MariaDB para poder probar y utilizar, virtualizar la base de datos para un ambiente productivo es una mala práctica. La base de datos es un recurso crítico y si bien actualmente la demanda es baja, consideramos que a futuro, es muy probable, que crezca la demanda del portal y una base de datos virtualizada en un contenedor puede terminar siendo un cuello de botella difícil de solucionar. Además, ejecutar una base de datos dockerizada para desarrollo es una práctica común, dado que le ahorra al programador el dolor de cabeza de buscar e instalar la versión correcta, instalar el dump de la base de datos, los usuarios y permisos necesarios y la estructura deseada por la aplicación, con una imagen Docker podemos hacer todo esto cuando ejecutamos nuestro contenedor por primera vez.

Compilación y empaquetado del Front y Back

Nosotros teníamos un conjunto de ideas en mente, la primera era poder evitar que el despliegue sea realizado por los desarrolladores (para evitar errores) y segundo, sabíamos que una vez terminado el desarrollo iba a quedar en manos del staff de “La Justa” u otro equipo, entonces no queríamos compartir las credenciales con los demás desarrolladores hasta no estar seguros que ellos iban a ser nuestro reemplazo. Docker si bien no es indispensable fue una herramienta fundamental para cumplir estos requisitos, una vez la imagen es generada por un nodo de manera automática, podemos subirla a docker hub[65] para que pueda ser instalada en el servidor posteriormente. Otra de las ventajas en cuanto al despliegue de contenedores es el tema del rollback, si algo sale mal, simplemente utilizamos la imagen anterior que todavía se encuentra instalada en el servidor, gastando 0

[65] https://hub.docker.com/

tiempo en generar algo nuevo y permitiéndonos trabajar en una solución al problema que encontramos en ese momento. Por último, nos gustaría resaltar la facilidad que tiene Docker de crear variables de entorno dentro del sistema operativo, por ejemplo, nosotros tenemos parametrizadas varias variables del back-end en variables de entorno del sistema operativo, gracias a esto podemos utilizar la misma imagen Docker pero cambiando las variables para apuntar nuestro back-end a producción o desarrollo así querramos.

Como conclusión, Docker no es solo una herramienta de moda en estos días que fue utilizada en La Justa solo por eso, si no, que es innegable que nos facilita mucho la puesta en producción de código al mismo tiempo que disminuye algún posible fallo en el deploy, nos garantiza que nuestro código puede ser compilado en un ambiente nuevo sin la necesidad de depender de los desarrolladores y por último nos ahorra tiempo de mantenimiento en producción, porque gracias a la feature de restart que tiene, si un contenedor se detiene por alguna razón, es automáticamente reiniciado sin necesidad de intervención humana.

NGINX

Nginx[66] Es un servidor web open source de software libre que nos permite realizar muchas cosas, entre las cuales se destaca, servidor HTTP, reverse proxy, load balancer o caché HTTP. En el proyecto de “La Justa” Nginx es utilizado como servidor web para Angular, como reverse proxy para el back-end y como servidor de recursos estáticos para todas las imágenes del frontend.

Cuando mencionamos que funciona como reverse proxy lo que queremos decir es todos las peticiones son hechas contra Nginx. El mismo funciona como una puerta de entrada única a la aplicación, de este modo todos las peticiones que tienen un determinado patrón en la URL son direccionados a la carpeta de Angular, a la carpeta donde des imágenes o al servidor back-end. La figura 39 describe el flujo de datos entre los clientes, el servidor y los recursos. Esto no solo ofrece flexibilidad a la hora de desplegar la aplicación en producción, sino que también ofrece seguridad, dado que Nginx es el único punto por donde pasan todas las peticiones de la aplicación. pudiendo cerrar el resto de puertos y exponiendo solo el 443 y el 80 (el puerto 80 lo único que hace es redireccionar la petición al puerto 443).

[66] https://www.nginx.com

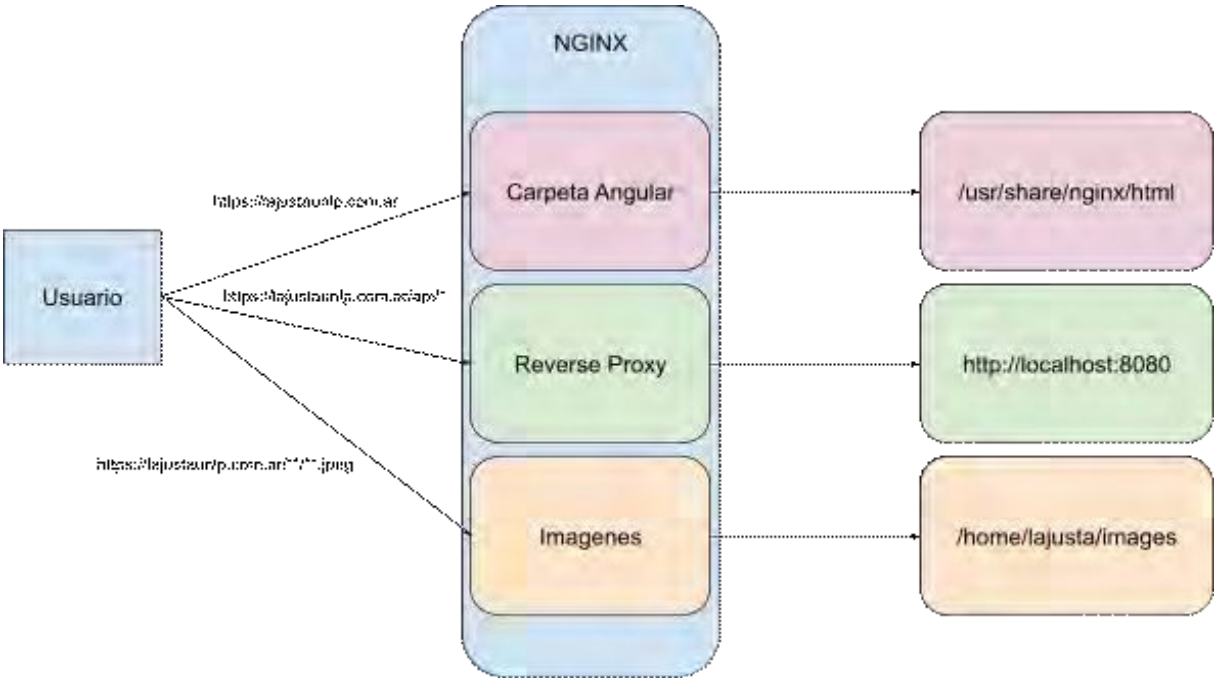


Figura 39: Muestra cómo se redireccionan los request dependiendo del request path

La configuración usada para el reverse proxy, se muestra en la figura 40: el servidor de HTML y el back-end, en su path **/api** redirige a una IP interna con puerto 8080, un tomcat corriendo dentro de la red; todo lo que termina en .jpeg, jpg, JPEG, PNG o JPG es redireccionado a la carpeta **/www/media/images** dentro de una imagen docker (como es un volumen virtual, este apunta a /home/lajusta/images) y por último, cualquier otra petición es redirigida a la carpeta donde tenemos instalado Angular, dentro del docker sería **/usr/share/nginx/html,** path donde copiamos los recursos estáticos de Angular al crear la imagen Docker.

```
server {
        listen  443 ssl;
        server_name https://lajustaunlp.com.ar;
        ssl_certificate     /root/lajustaunlp.com.ar.crt;
        ssl_certificate_key /root/lajustaunlp.com.ar.key;
        access_log /var/log/nginx/reverse-access.log;
        error_log /var/log/nginx/reverse-error.log;
        sendfile_max_chunk 512k;

        location / {
            root /usr/share/nginx/html;
            index index.html;
            proxy_max_temp_file_size 0;
            try_files $uri $uri/ /index.html @proxy;
        }
        location /api {
            proxy_pass http://66.97.**.***:8080;
        }
        location @proxy {
            proxy_pass http://66.97.**.***:80;
        }
        location ~ \.(jpeg|jpg|png|JPEG|PNG|JPG) {
            autoindex on;
            root /www/media/images;
        }
}
```

Figura 40: Configuración reverse proxy

Por último y como una configuración separada tenemos un redireccionamiento, como se puede ver en la figura 41, en el que toda petición que llega por HTTP es redireccionada al puerto seguro de HTTPS y de este modo no permitir, bajo ningún punto, que algún usuario acceda al portal sin certificado SSH.

```
server {
        listen 80;
        server_name www.lajustaunlp.com.ar lajustaunlp.com.ar;
        return 301 https://lajustaunlp.com.ar$request_uri;

        location / {
            return 301 https://lajustaunlp.com.ar$request_uri;
        }
}
```

Figura 41: Configuración redireccion request Http

Análisis de alternativas a servidor web: Apache

Una posible alternativa a Nginx es Apache server, que ofrece funcionalidad para servir recursos HTTP o estáticos, para configurar el reverse proxy, conocido en Apache como Proxy, para redireccionar el flujo al back-end sin necesidad de exponerlo. Apache server es una tecnología que se usa en la industria de producción de software, desde hace más tiempo que Nginx, está fuertemente testeado y probado. La razón por la cual nos inclinamos por Nginx es la necesidad de utilizar el servidor dentro de una imagen Docker, dado que es más sencillo poder arrancar la imagen de manera local, con los parámetros locales y luego usar la misma imagen pero cambiando las configuraciones para levantar el servidor en producción. Apache no ofrece una imagen Docker por defecto, y la única forma de lograr esto es creando nuestra propia imagen y aunque no se trata de algo complejo , los tiempos apremiaban y Nginx está excelentemente documentado, con lo cual decidimos usar y al mismo tiempo aprender a manejar Nginx por sobre Apache.

En algún proyecto futuro posiblemente elijamos Apache server dado que es uno de los mejores de la industria, y como mencionamos antes, si bien no ofrece una imagen Docker por defecto, su creación no es compleja, solo que lleva tiempo.

CAPÍTULO 6 - TESTING Y PUESTA EN PRODUCCIÓN

EVALUACIÓN DE SERVIDORES

Una de las partes más difíciles de decidir a la hora de poner nuestro código en producción fue el servidor, evaluamos varias propuestas como ser AWS, GCP, Digital Ocean, Azure como posibles plataformas internacionales donde desplegar el servidor de “La Justa”, asi como tambien evaluamos propuestas locales como ser Iplan, y el finalmente el elegido fue DonWeb. El principal problema a la hora de elegir el servidor fue el costo del servicios: las plataformas internacionales están atadas al dólar, y con un país cuya moneda es tan volátil, apostar a una plataforma internacional puede dejarnos en un futuro cercano sin la posibilidad de afrontar el gasto y tener que dar de baja el servidor. Los servidores internacionales ofrecen numerosas ventajas, fundamentalmente en lo referido a base de datos, entre ellos servicios como RDS (Relational Database Service) que permiten crear automáticamente una base de datos MySQL o MariaDB o Postgres y poder extraer todo el potencial de ellas. En nuestro caso, la base de datos de “La Justa” fue configurada mayormente mediante procesos manuales, como ser las IP que tienen acceso a la plataforma o la cantidad de memoria que se utiliza para cache. RDS resuelve eso con solo contratar el servicio, además de proveer backups diarios sin costo extra.

Se contrató en DonWeb 2 servicios cloud de 2 procesadores y 4 GB de RAM cada uno, en uno está instalada la base de datos MariaDB mientras que en el otro están instaladas las imágenes Docker del front-end y back-end, como se muestra en la figura 42.

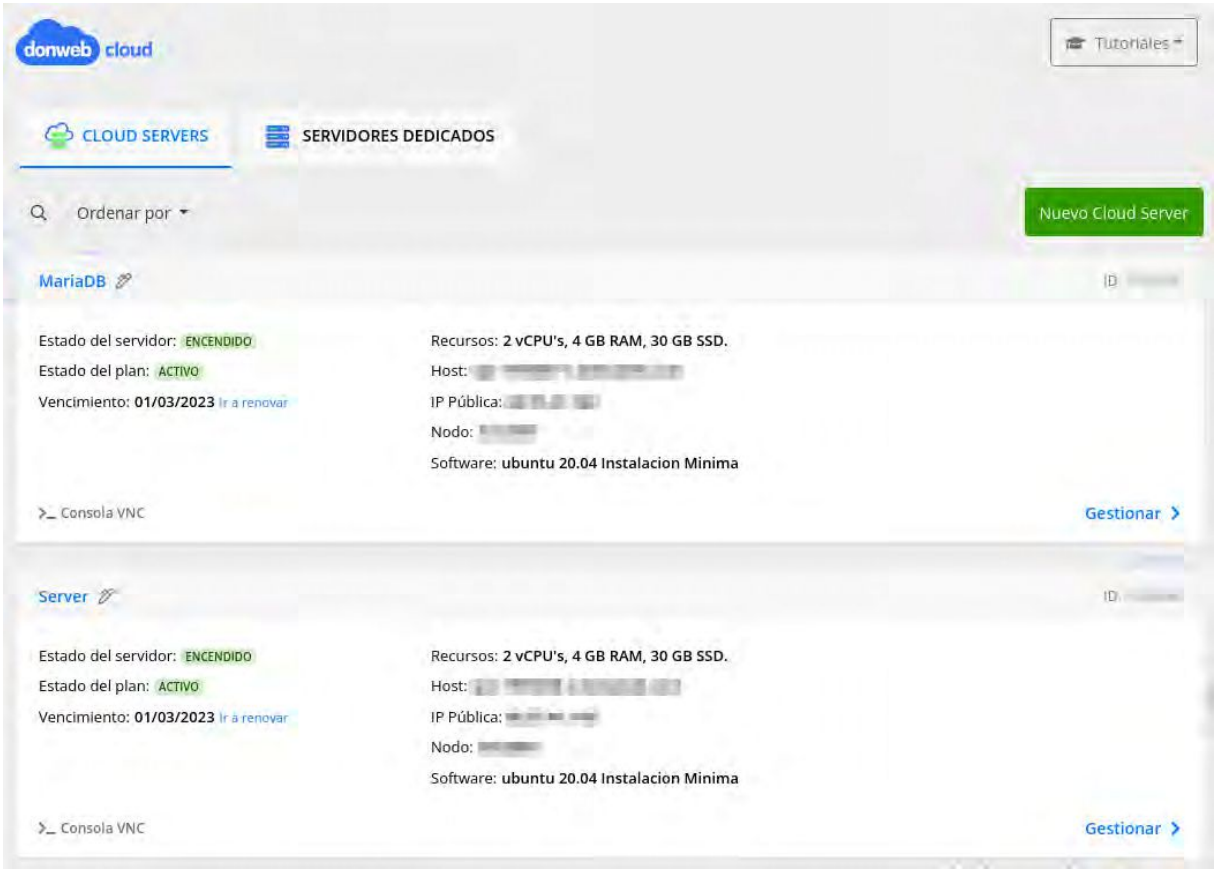


Figura 42: Home de la página de DonWeb

Por otra parte, no solo el costo del servicio de hosting fue lo que inclinó la balanza hacia DonWeb, sino también la posibilidad de disponer de un servicio de firewall para evitar conexiones no deseadas o abrir sólo los puertos que deseamos en nuestros servidores, monitorear la cantidad de recursos consumidos por cada servidor o disponer de un fácil acceso a los mismos.

MONITOREO DE USO

Como mencionamos antes, uno de los principales monitoreos que usamos es el manejo de memoria, procesamiento y tráfico de datos que nos ofrece DonWeb, como muestra las figura 43, figura 44 y figura 45. Esto fue de gran . ayuda a la hora de decidir dónde íbamos a colocar los ”runners” de GitLab dado que consumen mucha memoria. El procedimiento consiste en disparar un pipeline y monitorear que los valores de memoria y procesamiento se mantengan en un rango normal.

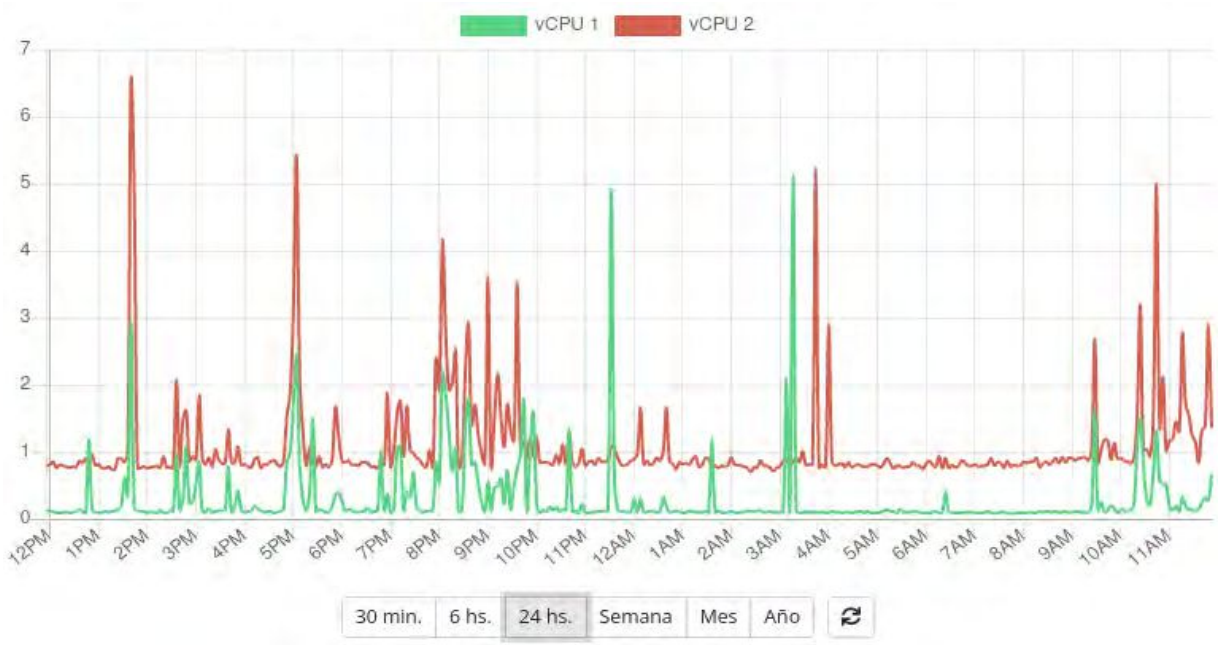


Figura 43: Capturas de pantallas de los monitores de DonWeb, Procesamiento

Figura 44: Capturas de pantallas de los monitores de DonWeb, Memoria RAM

Figura 45: Capturas de pantallas de los monitores de DonWeb, Red

Uno de los trabajos a futuro podría ser agregar alguna herramienta como Splunk o NewRelic para monitorear los logs en un solo lugar. Esto permitirá centralizar la información de los logs, disponer de la misma en un fácil acceso Web y preparar alertas que avisen de puntos críticos en donde los recursos se vean comprometidos. Por el momento, por el tamaño de datos que manejamos, la cantidad de instancias de back-end y front-end, o la cantidad de usuarios concurrentes que tenemos en el portal, las herramientas mencionadas previamente resultan suficientes.

UAT TESTING

Una vez que el desarrollo del back-end y del front-end fueron terminados, se desplegó el portal en AWS, en una máquina free tier que cuenta con una t3.micro de 0.5 GB de RAM y 1 procesador. En este primer despliegue de testeo, se pudo evaluar el comportamiento del sitio con los diferentes perfiles de usuario, advertir cambios de UI y ajustes en la arquitectura. Hasta ese momento lo que hacíamos era almacenar las imágenes en la base de datos en una columna BLOW, luego convertirlas en un string base64 y enviarlas dentro de los JSON de respuesta. Esto claramente no fue una buena solución, dado que listados largos como por ejemplo el de productos, al principio podían llegar a pesar 5 MB y tardar hasta 10 segundos en renderizarse. Este comportamiento fue detectado en esta primera instancia de testing, de la que participaron los usuarios de perfil administrador de “La Justa” y un grupo reducido de consumidores que fueron convocados para tal fin. Luego de hacer los cambios pertinentes, se logró mejorar la latencia entre el front-end y el back-end, respondiendo en tan solo 100 ms todas las transacciones y cargando las imágenes con Nginx como servidor de recursos estáticos.

Para poder reportar los problemas, arreglarlos y seguir la pista de lo que se iba completando y lo que faltaba, se le dio acceso a un grupo de consumidores y usuarios administradores de “La Justa” a una pizarra de Trello donde se escribía cada cambio en el portal, bug resuelto y bug encontrado. Esto resultó de mucha ayuda dado que permitió priorizar bugs, asignar la resolución a quien corresponda, determinar si eran del front-end, back-end o del equipo de “La Justa”, y ayudó a organizarnos. La figura 46 muestra una captura del Trello interno utilizado para organizar las tareas.

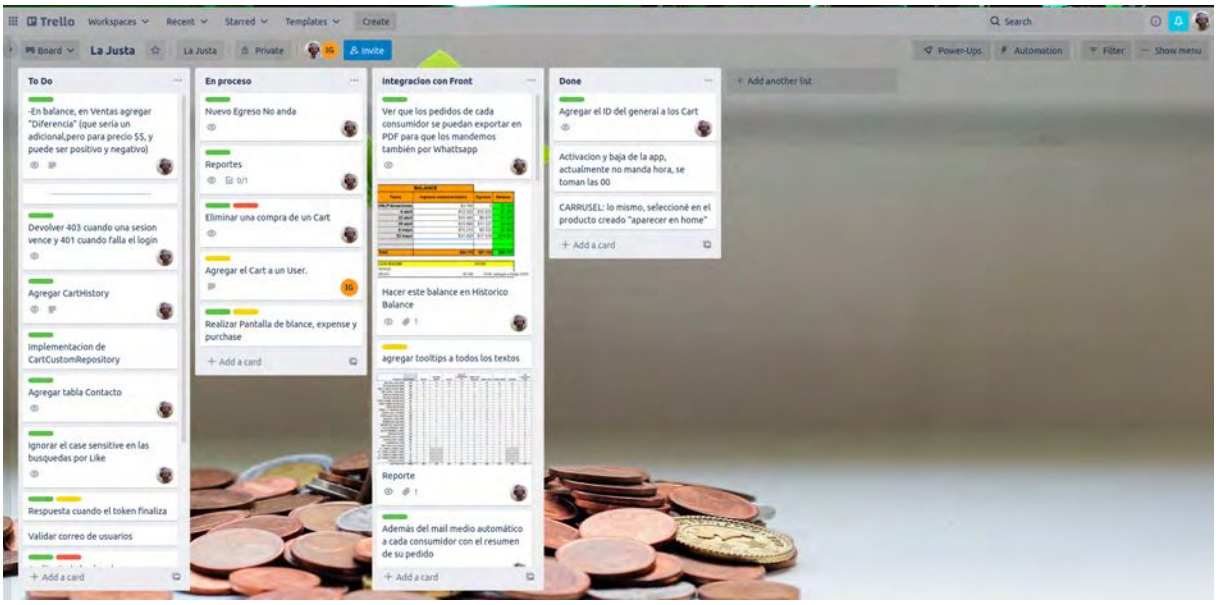


Figura 46: Captura de Trello interno

CAPÍTULO 7 - CONCLUSIONES Y TRABAJOS A FUTURO

CONCLUSIONES

Desde el punto de vista personal, ambos tesistas trabajamos en la industria de desarrollo de software desde hace más de 10 años, pasando por distintas compañías y usando distintas tecnologías. Esta experiencia acumulada nos permite volcar en la tesina todo lo aprendido en cuanto a procesos de desarrollo y tecnologías. Por otro lado, los años de estudio en la facultad nos permitieron evaluar nuevos lenguajes, nuevas herramientas y procesos, para poder aplicarlos, siendo sumamente enriquecedor dado que pudimos aportar al desarrollo de esta tesina, ejemplo de ellos son Kotlin, Kanban o GitLab Actions.

Desde el análisis de requerimientos, la puesta en producción e incluso el manejo de los tiempos acortados por la pandemia fue todo un desafío. Acostumbrados a trabajar en equipos grandes de 5 a 8 personas, de golpe veíamos que solamente eramos 2 y que teníamos que cubrir todos los roles de las distintas áreas: analistas funcionales, scrum masters, project managers de nosotros mismos o devops, fueron unos de los tantos “trajes” que nos probamos a lo largo de la tesis. El camino fue divertido y desafiante, como mencionamos antes pudimos experimentar con tecnologías y herramientas que en el día a día laboral no podemos hacerlo, ya sea por el entorno o los clientes y fundamentalmente por las demandas y dinámicas específicas de un trabajo, esto no es posible. El desarrollo de esta tesina, nos permitió comprender los diferentes puntos de vista de cada uno de los engranajes que componen el desarrollo de software, tomar nuestras propias decisiones en diálogo con nuestros directores y la organización adoptante del portal, “La Justa”.

En este proceso pudimos entregar un portal de compra virtual hecho con tecnologías actuales, abiertas, libres, competitivas, que no tienen nada que envidiarle a otros sitios de compras del mercado. Este proyecto a su vez está abierto en GitLab para que todo el que desee lo pueda descargar, modificar o mejorar, agregando más funcionalidad de la que ya tiene el portal. Esta es una de las grandes ventajas del software libre, de código abierto, mientras haya un desarrollador que suba cambios, el proyecto nunca es abandonado, dado que no pertenece a un grupo de personas o a una institución, es de todo el mundo.

Por otro lado, la naturaleza del “negocio” y del problema fue un desafío, al iniciar el proyecto se analizaron herramientas que podían adaptarse a las demandas del equipo de “La Justa” y, para nuestra sorpresa, encontramos bastantes. Sin embargo, éstas no resolvían los problemas planteados por “La Justa” en su totalidad, esto nos llevó a plantear un desarrollo desde cero, y así fue como lo hicimos durante la pandemia global de COVID-19 y en menos de 6 meses se implementó el portal adaptado a las necesidades del equipo de “La Justa”, esto puso el reloj en nuestra contra. Un hecho con el que nos encontramos fue que el equipo de “La Justa” no utilizaba ningún software integrado para la comercialización. En este sentido, se pasó de utilizar planillas excel y formularios de google, aislados, que debían integrarse manualmente, a un portal web totalmente adaptado a sus necesidades,

simplificando las tareas cotidianas. El punto antes mencionado es muy importante dado que una vez que el sistema fue puesto en producción, el staff de “La Justa” nos invitó a charlas virtuales (todavía estábamos en medio de la pandemia) donde se comentaron los problemas que tenían y las ventajas de disponer del portal, que les permitía tener tiempo libre para enfocarse en otros temas o procesos. No solo les facilitó hacer lo mismo que ya venían haciendo de manera automática, repetitiva y confiable, si no que les permitió tener más tiempo libre, tiempo que utilizan hoy en día para pensar mejoras que hacerle al portal, y a la comercialización en general. Para nosotros esto fue un “efecto colateral” dado que nunca lo pensamos, pero fue muy gratificante la primera vez que lo mencionaron.

En el camino pudimos adentrarnos en un modelo de negocio distinto, el de la economía social y solidaria, que nos planteaba desafíos nuevos. Al no centrarse en el lucro, pero sin perder eficiencia y calidad, este modelo requiere que el diseño de las herramientas y la construcción de tecnologías estén co-construidas permanentemente con los sujetos que las van a utilizar, de manera de realmente convertirse en tecnologías sociales que puedan apropiarse por ellos, quienes las van a utilizar, y además requieren transparentar a los sujetos que están atrás, que la plataforma de comercialización tenga sujeto, muestre el trabajo y a los trabajadores que están detrás, tanto de la producción de esos bienes que se comercializan, como así también atrás del servicio de intermediación solidaria y que los consumidores también puedan interaccionar y tener su lugar en el circuito, de tal manera de que estas economías dispongan de herramientas aptas a su cosmovisión de ser economías con el sujeto en el centro, con la vida en el centro, y no ser espacios donde los bienes van y vienen invisibilizando el trabajo y la vida de quienes protagonizan dichos procesos. Creemos haber respondido correctamente con esta realidad, al construir muy de cerca las herramientas con sus protagonistas y al haber pensado estrategias que revelen qué hay detrás del cliente, no solo como estrategia comercial sino de paradigma de construir economías para la vida de las comunidades.

TRABAJOS FUTUROS

Uno de los principales puntos que podemos agregar a la deuda técnica, es el uso de Unit o Integration tests, actualmente el portal cuenta con un proceso de CI corriendo todos los tests que creemos necesarios antes de deployar, pero lamentablemente la cantidad de cobertura de tests es 0, tanto en back como en front. Esto es un problema, porque muchos de los problemas que fueron reportados por los usuarios o que actualmente encontramos en producción podrían haber sido evitados fácilmente con solo tener test de unidad en el portal.

Otro punto a analizar es la frecuencia de los backups de base de datos, cada cuánto los hacemos, cuántos archivos podemos mantener al mismo tiempo antes de comenzar a reutilizar otros backups. Teniendo en cuenta que solo disponemos de 30 GB si no usamos una política de borrado de backups viejos, podemos terminar usando toda la memoria del servidor.

Como mencionamos en el apartado sobre Docker, actualmente para seguir un log de la aplicación es necesario conectarse a la máquina virtual de DonWeb y solicitarle a Docker que nos muestre los logs de aplicación. Esto, si bien en un principio y por la cantidad de elementos que tenemos en la arquitectura no supone un problema, a futuro puede convertirse en algo insostenible. Para ello una herramienta del tipo de manejo de log centralizado ya sea NewRelic o Splunk puede ser de gran ayuda. Con dichas plataformas dispondríamos de todos los logs en un solo lugar, pudiendo filtrar por ambiente, tipo de error, fecha, o bien crear alertas para que nos envíe un email en caso de que un recurso se encuentra comprometido o pasa alguna barrera crítica.

También es posible generar una app mobile y agregar nuevas funcionalidades al portal dado que el modelo se dejó preparado para que así sea, algunas de ellas pueden ser descuentos para algunos usuarios, campañas de venta con promociones, entregas a domicilio y formas de pago digital. También se podrían agregar roles nuevos, como el rol productor.

Otro punto a considerar es el de disponer de distintos entornos de desarrollo en DonWeb, que nos permitan testear con el equipo de “La Justa” nuevas funcionalidades sin poner en riesgo la versión de producción. Por ejemplo, se podrían definir distintos tipos de entornos donde nuestro código va a ser deployado, un ejemplo sería Producción y Desarrollo, ver qué salió mal sin necesidad de tener problemas en Producción. Esta es una interesante herramienta que ofrece GitLab con solo configurar qué entorno pertenece a que branch de GitLab, podemos subir el código a los entornos deseados. El hecho de poder medir, interactuar y ver qué salió mal en un deploy sin la necesidad de conectarse a la máquina virtual por ssh es muy poderoso y definitivamente un “must have”[67] en todo tipo de proyecto de software.

[67] Expresión que significa “esto debe de estar si o si”

REFERENCIAS BIBLIOGRÁFICAS

Abramovich Ana Luz y Vázquez Gonzalo. (2007). Experiencias de la Economía Social y Solidaria en la Argentina. Estudios fronterizos, 8(15), 121-145. Recuperado en 20 de junio de 2020, de https://tinyurl.com/ydbug5c9.

Alcoba, D y Dumrauf, S et al (compiladores). (2011). “Del productor al consumidor. Apuntes para el análisis de las ferias y mercados de la agricultura familiar en Argentina.”. Colección Agricultura Familiar O7 CIPAF. PN Territorios. INTA. Ministerio de Agricultura, y Ganadería y Pesca de la Nación. Buenos Aires.

Angular IO https://docs.angular.lat/docs

Barros, et. al. (2015). citada en: Manzolido, E, Carmona, B., Sendin, B. y Moyano, P., “Ferias de Productores de la agricultura familiar en la Universidad Nacional de La Plata”, en: Errecalde, S. M. (Comp.) (2020). *La economía popular ante la crisis: por la defensa de los derechos y hacia una economía social y ambientalmente sustentable.* Berazategui, Cuadernos de la Economía Social y Solidaria, 2020. ISBN: 978-987-45992-3-0.

Bulich, et. al. (2017). citada en: Manzolido, E, Carmona, B., Sendin, B. y Moyano, P., “Ferias de Productores de la agricultura familiar en la Universidad Nacional de La Plata”, en: Errecalde, S. M. (Comp.) (2020). *La economía popular ante la crisis: por la defensa de los derechos y hacia una economía social y ambientalmente sustentable.* Berazategui, Cuadernos de la Economía Social y Solidaria, 2020. ISBN: 978-987-45992-3-0.

Bauer Christian, King Gavin. (2006). J*ava Persistence with Hibernate: Revised Edition of Hibernate in Action*. Editorial Manning Publications. ISBN: 1932394885.

Burke Bil.l (2013). *RESTful Java with JAX-RS 2.0, 2nd Edition*. Designing and Developing Distributed Web Services. Editorial O'Reilly Media. ISBN: 978-1-44936-134-1.

Caracciolo, M. (2015). “Situación de la institucionalidad de apoyo a la innovación y a la gestión comercial de la agricultura familiar en la Argentina”. IICA. ArgenINTA. Programa Foncyt. Ministerio de Agroindustria.

Coraggio, José Luis. (2007): La economía social desde la periferia. Contribuciones latinoamericanas, UNGS-Altamira, Buenos Aires.

Coraggio, JL. (2010). “Pensar desde la perspectiva de la Economía Social”. En Cittadini, R, Caballero, L, Moricz, M y Mainella, F. (Compiladores). Economía Social y Agricultura Familiar. Hacia la construcción de nuevos paradigmas de intervención. Ediciones INTA. Buenos Aires

Craig Larman. (2002). UML y patrones : una introducción al análisis y diseño orientado a objetos y al proceso unificado

Craig Walls. (2016). Spring boot in Action, Manning

Deux Marzi, Maria Victoria. Vannini, Pablo A. (2016). *Manual de tecnologías abiertas para la gestión de organizaciones de la economía social y solidaria* / Pablo Vannini; María Victoria Deux Marzi. - 1a ed .Ciudad Autónoma de Buenos Aires: Cooperativa de Trabajo gcoop Ltda.; Los Polvorines: Editorial de la Universidad Nacional de General Sarmiento.

Dmitry Jamerov y Svetlana Isakova. (2017). Kotlin in Action, Manning

F. J. García-Peñalvo y A. García-Holgado. (2017-2018). "Análisis orientado a objetos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática.

F. J. García-Peñalvo y A. García-Holgado. (2018). Salamanca, España: Grupo GRIAL, Universidad de Salamanca, [Online]. Disponible en: https://goo.gl/LZ3K6z. doi: 10.5281/zenodo.1181820. (pp. 17-47)

Flanagan David. (2006). JavaScript - The Definitive Guide, 5th ed., O'Reilly, Sebastopol

Jones M. , Bradley J., Sakimura N. (2015). *Request for Comments: 7519*. *JSON Web Token (JWT).* ISSN: 2070-1721. URL: https://tools.ietf.org/html/rfc7519

JWT.IO: https://jwt.io

David J. Anderson. (2011). Kanban: Cambio Evolutivo Exitoso Para su Negocio de Tecnología

One framework. Mobile & desktop (s.f). URL: https://angular.io

Mosse, Luis. (2019). Organizaciones de Intermediación Solidaria en el INTA. en Mercados: diversidad de prácticas Comerciales y de Consumo. Ediciones INTA. ISBN 978-987-521-998-4 Año 2019.

Redhat: https://www.redhat.com/es/topics/api/what-is-a-rest-api

VueJS: https://v2.vuejs.org/v2/guide/comparison.html?redirect=true#AngularJS-Angular-1

Richardson Leonard, Ruby Sam, Amundsen Mike. (2013). *RESTful web APIs, 1st Edition*. Editorial O'Reilly Media. ISBN: 9781449358068.

Roy Thomas. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.

SimilarTech Ltd.© (2013-2022). *Spring Market Share and Web Usage Statistics*. SimilarTech. https://www.similartech.com/technologies/spring

Thomas, H. E., Becerra, L. D. y Picabea, J. F. (2014). *Colaboración, Producción e Innovación: Una Propuesta Analítica y Normativa para el Desarrollo Inclusivo.* Astrolabio, (12). Recuperado de: https://revistas.unc.edu.ar/index.php/astrolabio/article/view/7370

Thomas Hernán, Albornoz María Belén y Picabea Facundo. (2015). Políticas tecnológicas y tecnologías políticas: dinámicas de inclusión, desarrollo e innovación en América Latina. 1ra Edición. Bernal: UNQ. ISBN: 978-987-558-359-7.

Viteri, ML. (2013). “Cambios en el Abastecimiento Hortícola. Una Mirada desde los Mercados Mayoristas del Gran Buenos Aires”. En Viteri, ML, Ghezán, G e Iglesias, D. (editores) “Tomate y lechuga: producción, comercialización y consumo”. Estudios Socioeconómicos de los Sistemas Agroalimentarios y Agroindustriales. Publicación INTA 13: 23-48.

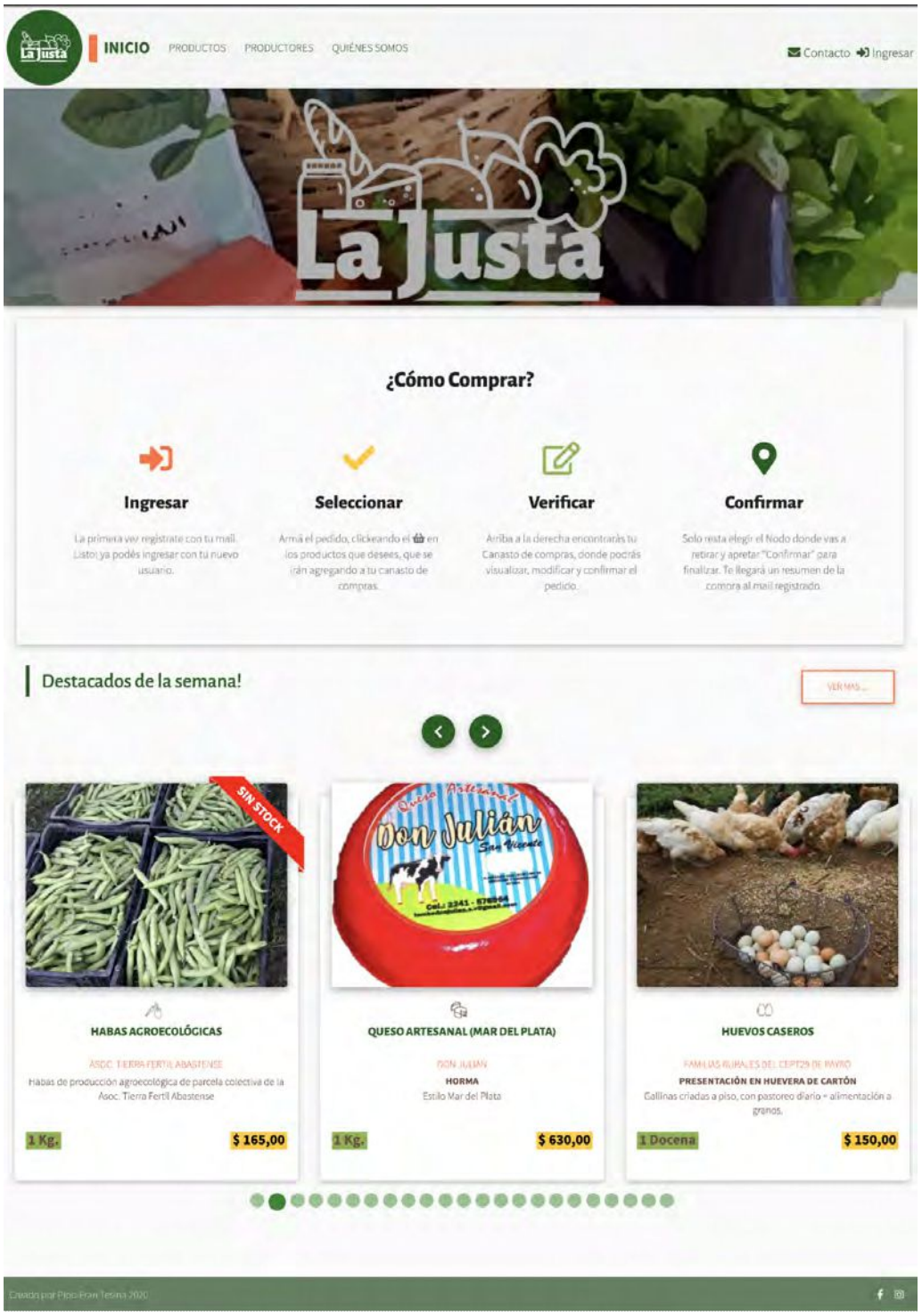
ANEXO

En este anexo se cuenta con la información de las distintas páginas implementadas para los diferentes perfiles de usuario y referencias a los paquetes instalados en el front-end para llevar a cabo el desarrollo.

PÁGINA PRINCIPAL

Esta es la primera página (Anexo figura 1) que se visualiza al ingresar al link https://www.lajustaunlp.com.ar/, a la cual acceden todos los tipos de usuarios, los anónimos, los de tipo consumidor o los administradores. En un principio los usuarios se encuentran todos como anónimos, salvo que estén logueados.

La página principal cuenta con un menú que contiene varios links a las distintas páginas, una canasta de compra (en caso de estar activo el sitio para realizar pedidos) y los botones de contacto e ingresar. También se puede apreciar un banner con slider en caso de haber más de uno, en el que se muestran imágenes con textos configurables, seguido a esto está disponible un instructivo de cómo realizar la compra. Luego se puede apreciar un carrousel con los productos destacados de la semana y un botón “ver mas” que lleva a la sección de productos.



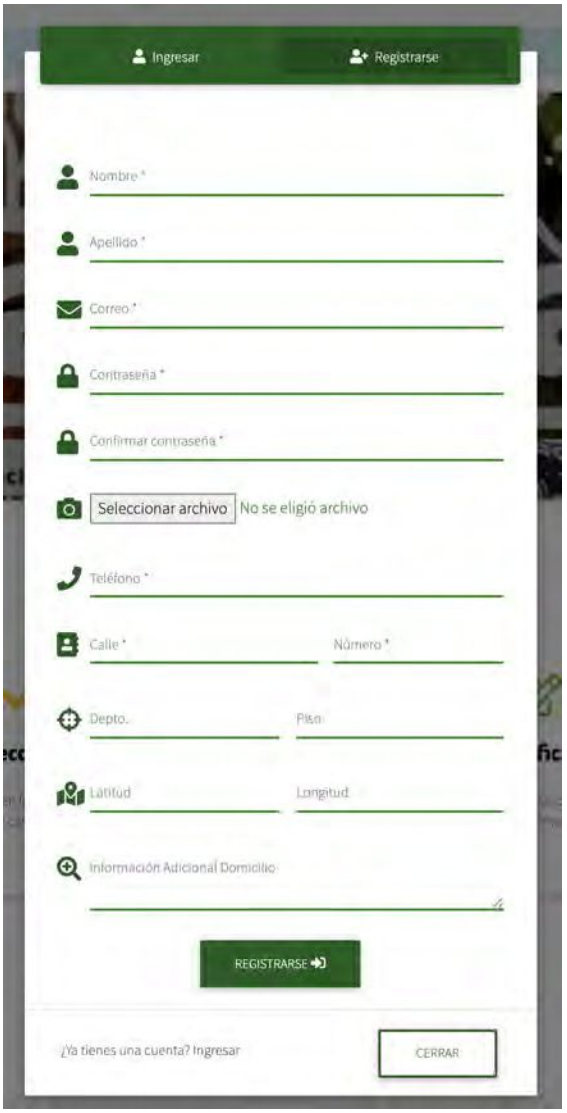
Anexo figura 1: Portal La Justa

REGISTRO Y LOGIN

Realizamos un modal/popup de registro y login con correo y contraseña, y un botón de logout. Tanto para el registro al portal como para ingresar, se comparte un modal que cuenta con ambas opciones, similar a tabs.

Registro

Para registrarse (Anexo figura 2) en el modal compartido debe estar seleccionado dicho tab:



Anexo figura 2: Modal de Registro

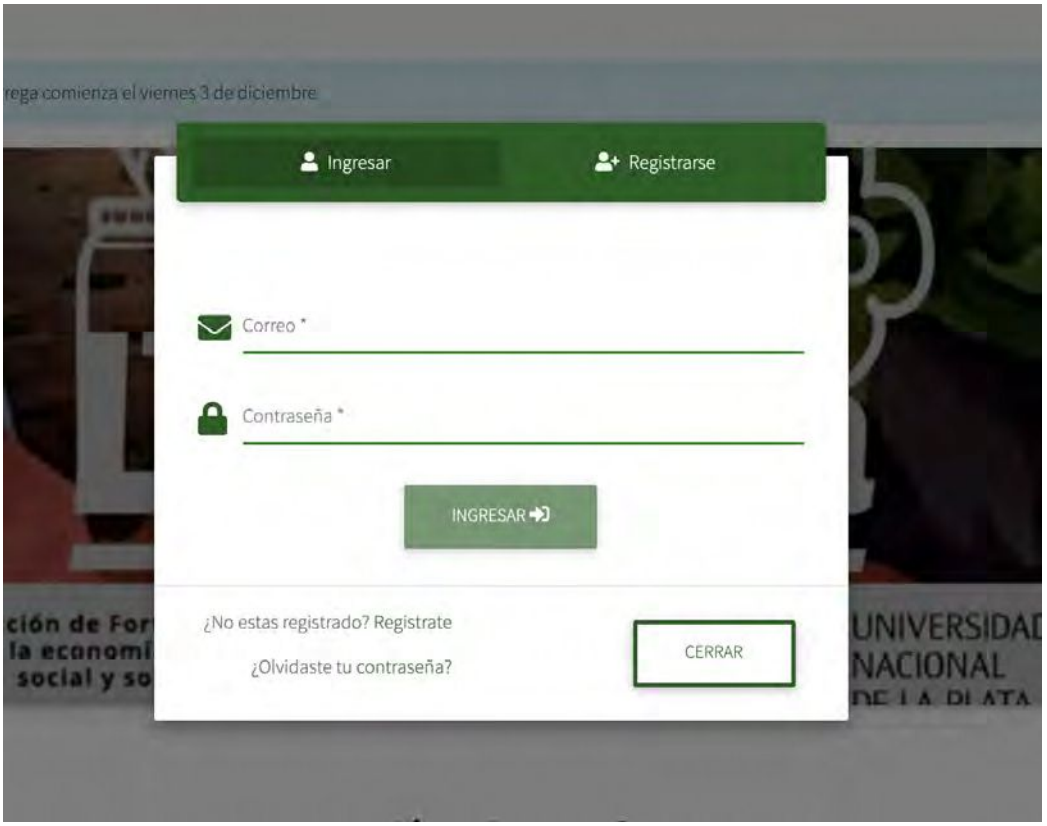
Al registrarse solicita la siguiente información:

- Nombre (obligatorio)
- Apellido (obligatorio)
- Correo (obligatorio)
- Contraseña y Confirmar Contraseña (obligatorio)
- Cargar una imagen si así lo desea el usuario
- Teléfono (obligatorio)
- Calle (obligatorio)
- Número (obligatorio)
- Departamento
- Piso
- Latitud
- Longitud
- Información adicional si es que quiere aclarar algo

Una vez registrado al usuario le va a llegar un link a su correo para validar la cuenta.

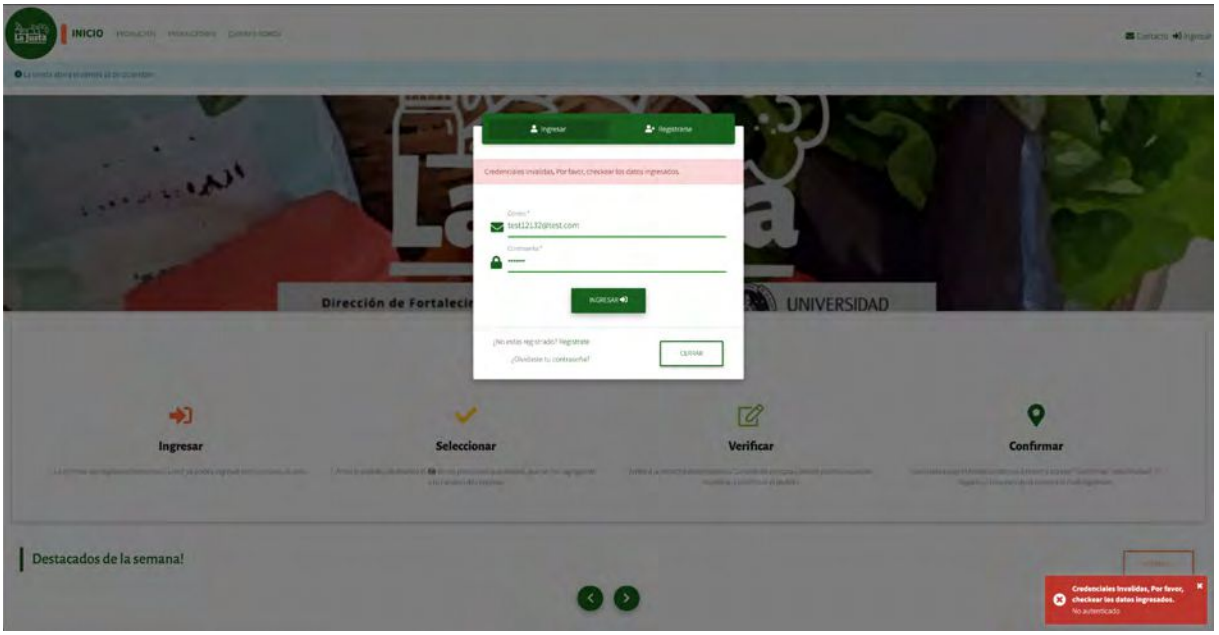
Login

En el caso del logueo, en el modal (Anexo figura 3) debe estar seleccionado el tab que dice “Ingresar” se ingresa con el correo registrado y la contraseña elegida.



Anexo figura 3: Modal de Login

Obviamente, en caso de ser incorrecto el usuario y/o la contraseña, se muestra un mensaje y un cartel que dice “Credenciales inválidas. Por favor, chequear los datos ingresados”(Anexo figura 4).



Anexo figura 4: Modal de Login con error

RECUPERAR CONTRASEÑA

En caso de no recordar la contraseña, hay un link que permite recuperarla al momento de ingresar. Si el correo ingresado se encuentra registrado en el portal, le va a llegar un email con un link para poder ingresar su nueva contraseña (Anexo figura 5).

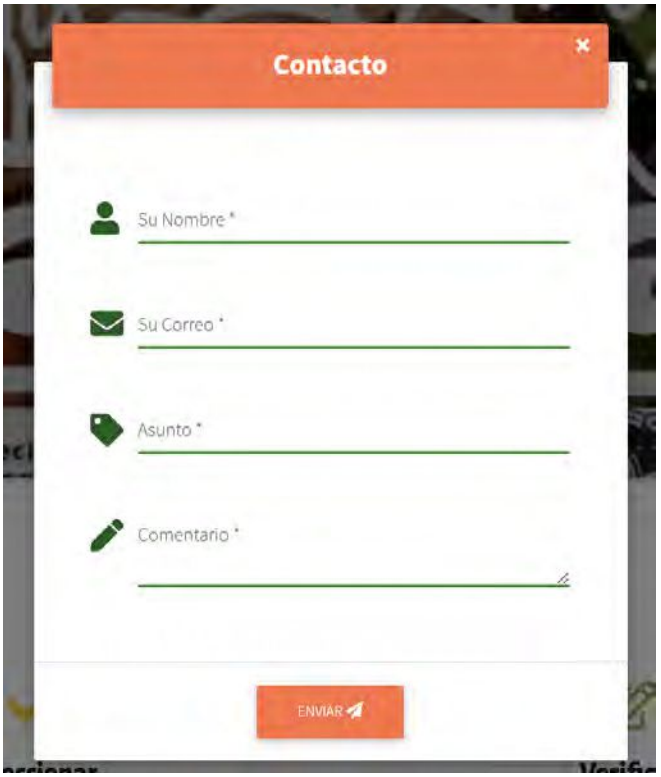
Anexo figura 5: Modal para el recupero de contraseña

Siendo el correo un campo obligatorio

CONTACTO

El sitio cuenta con un modal (Anexo figura 6) para ponerse en contacto con los administradores y cuenta con los siguientes campos obligatorios:

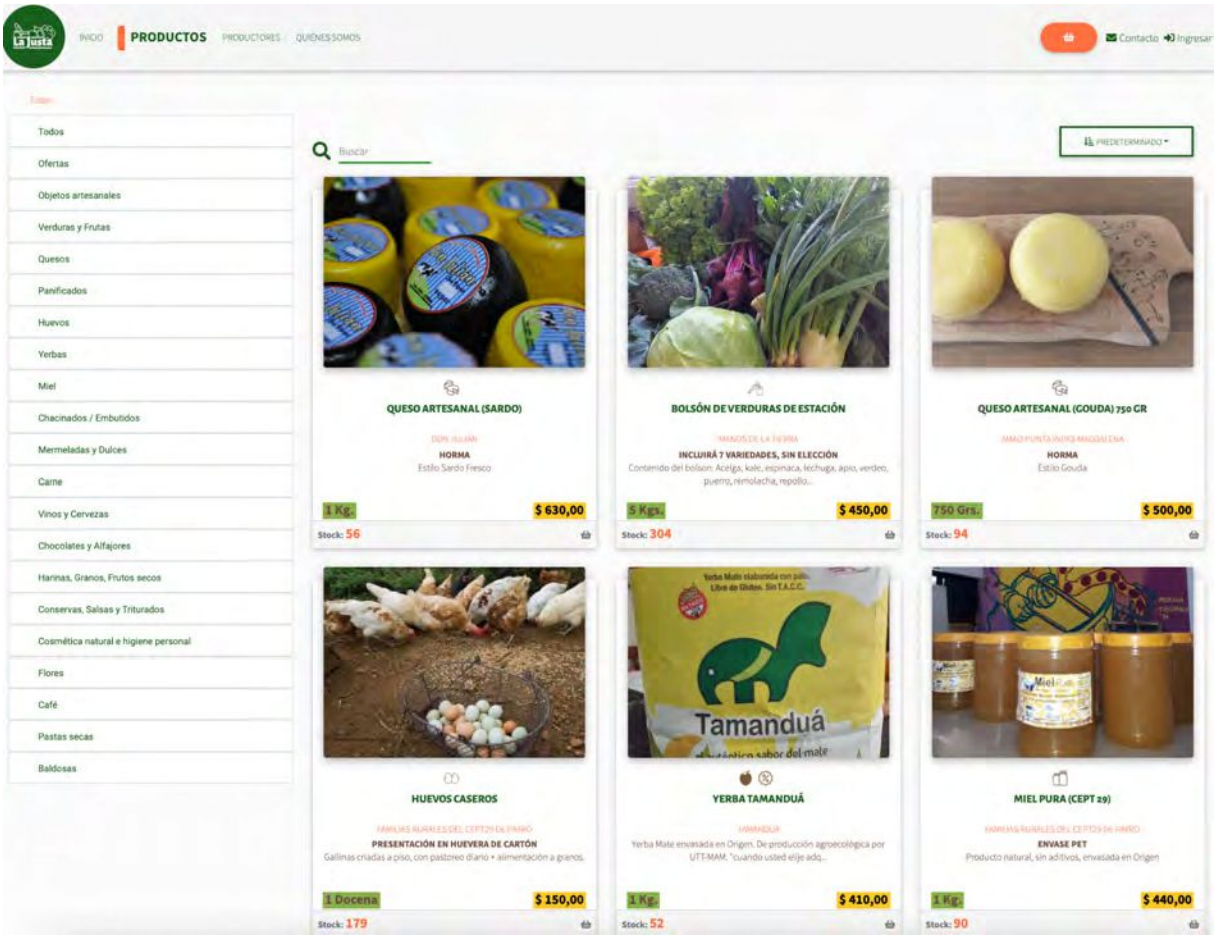
- Nombre
- Correo
- Asunto
- Comentario



Anexo figura 6: Modal de contacto

PÁGINA DE PRODUCTOS

El sitio cuenta con la página de productos (Anexo figura 7) para seleccionar y/o visualizar:



Anexo figura 7: Página de productos

Del lado izquierdo se pueden seleccionar las categorías y del lado derecho se ven todos los productos, los cuales se pueden ordenar de distintas formas, predeterminado, menor precio, mayor precio, nombre ascendente o nombre descendente y también cuenta con un buscador.

Para agregar al carrito, hay que apretar en la canastita del producto y podes modificar la cantidad desde la tarjeta.

TARJETA DE PRODUCTOS

Las tarjetas de los productos (Anexo figura 8) cuentan con una información básica y la posibilidad de agregar a la canasta en caso de estar activa la ronda.

Anexo figura 8: Tarjeta de producto

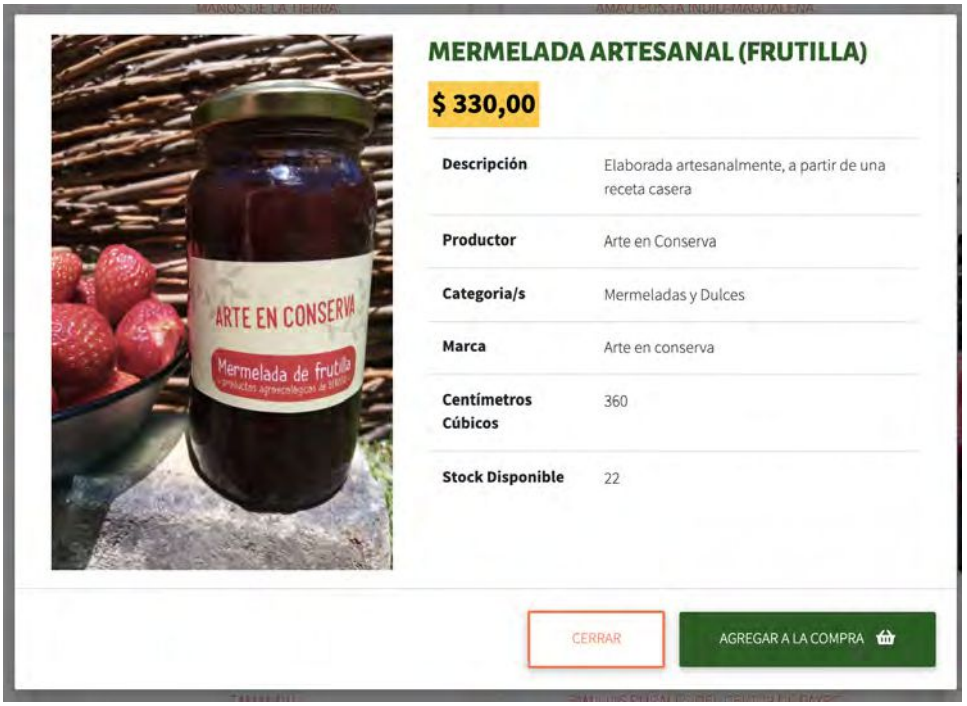
En dicha tarjeta se puede visualizar una imagen, un título, quienes son los productores, descripción, unidad de medida, precio stock y en caso de estar habilitado el sitio para la compra se puede visualizar la canastita.
Si se aprieta en la canasta (Anexo figura 9), se puede visualizar un input que permite agregar más cantidad o quitar el producto de la canasta (aclaración: no se modifica el stock hasta que la compra no es realizada)

Anexo figura 9: Tarjeta de un producto agregado

En caso de quedarse sin stock (Anexo figura 10), no se visualizar la canasta y le aparece una banderita de color rojo.

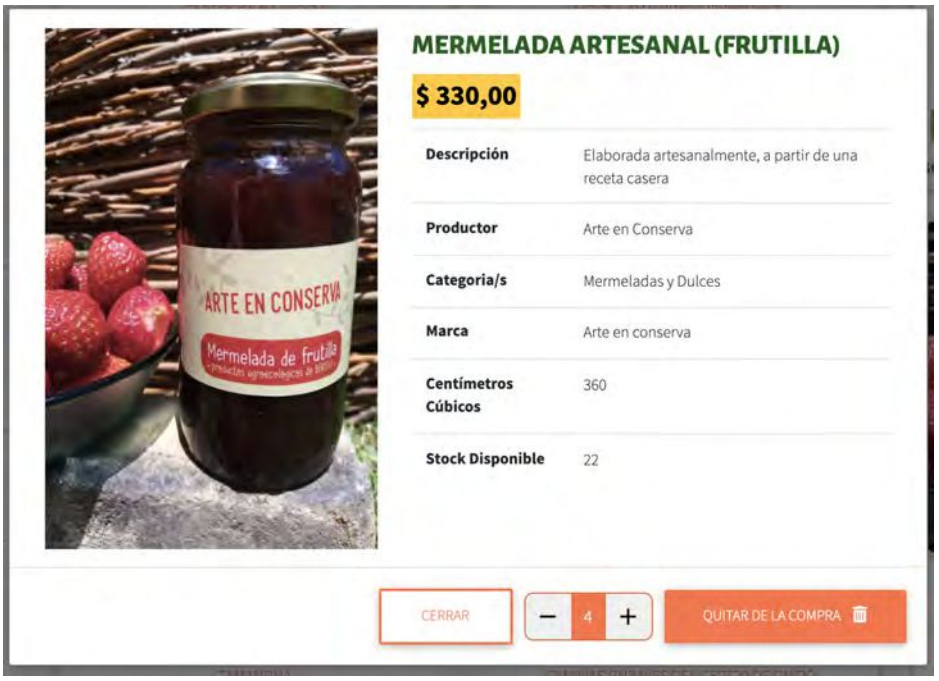
Anexo figura 10: Tarjeta de un producto sin stock

Al clickear sobre la imagen se puede apreciar un modal con otra visual más grande y mayor información (Anexo figura 11)



Anexo figura 11: Modal de un producto

En caso de contar con stock se visualiza el botón para agregar a la canasta como se visualiza en la imagen (Anexo figura 11) y al igual que en las tarjetitas, se puede modificar la cantidad y eliminar de la canasta (Anexo figura 12)



Anexo figura 12: Modal de un producto agregado

Al apretar en la imagen del modal (que puede ser un slider si se cargan 2 o más imágenes) se puede ampliar aún más la imagen del producto.
Al agregar un producto se puede visualizar un cartel (Anexo figura 13) en la parte inferior derecha de la pantalla que informa que el producto fue agregado a la canasta.

Anexo figura 13: Cartel de producto agregado

y en caso de quitar un producto de la canasta (Anexo figura 14), se puede visualizar en el mismo lugar un cartel que nos informa que se eliminó el producto

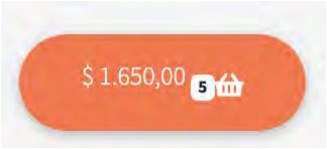
Anexo figura 14: Cartel de producto quitado

CANASTA

Al ingresar al sitio, si se encuentra abierta una ronda para la compra, el mismo va a mostrar un botón de una canasta en el menú de arriba a la derecha

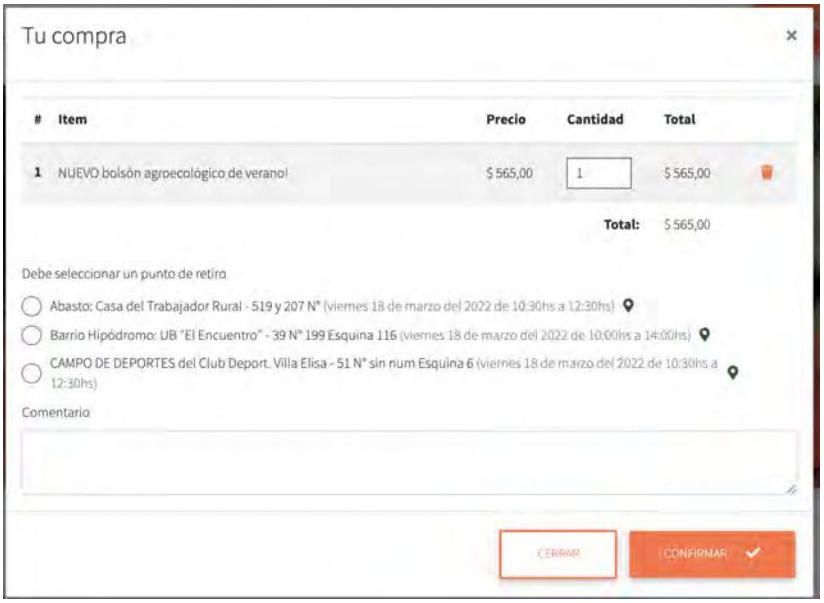
Anexo figura 15: Menú con una ronda abierta para comprar.

En caso de tener algún producto agregado a la canasta, el botón se va a visualizar con el monto a abonar y la cantidad de productos agregados (Anexo figura 16)



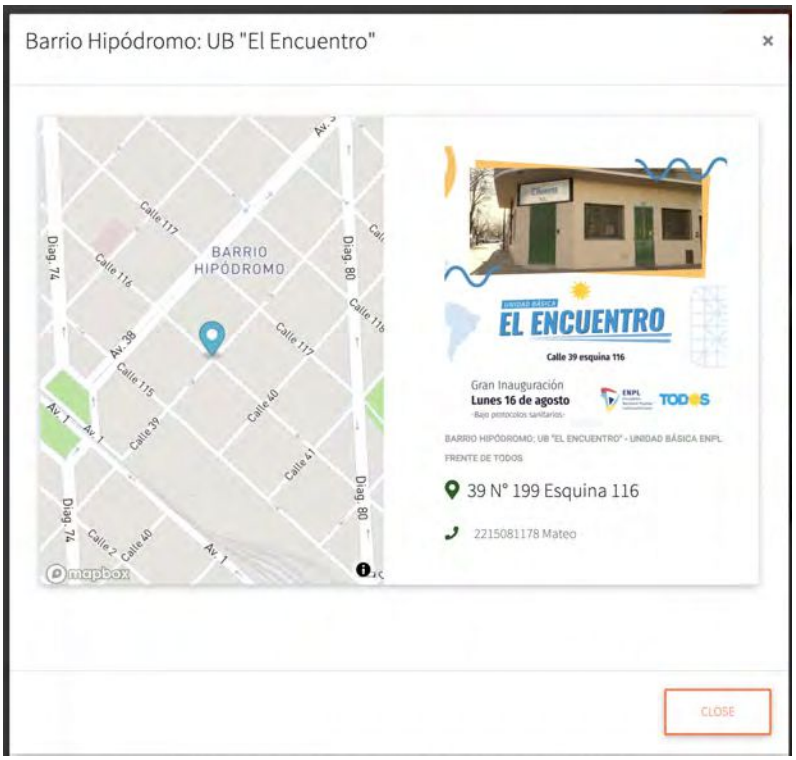
Anexo figura 16: Botón con productos agregados

Al apretar sobre este botón se puede visualizar un modal con la informacion mas detallada de los productos agregados (Anexo figura 17)



Anexo figura 17: Modal de la canasta de compras

Dicho modal cuenta con los productos agregados, y las opciones de los nodos donde se puede ir a buscar los pedidos con días y horarios. Se pueden eliminar productos, cambiar cantidades (como máximo el stock disponible) y agregar un comentario adicional por cualquier cosa.
Al clickear sobre el pin verde que se encuentra al lado de los puntos de entrega, se puede visualizar un modal con la información del nodo (Anexo figura 18).



Anexo figura 18: Modal de un nodo

Que cuenta con información adicional, una imagen del punto de retiro y ubicación en el mapa.

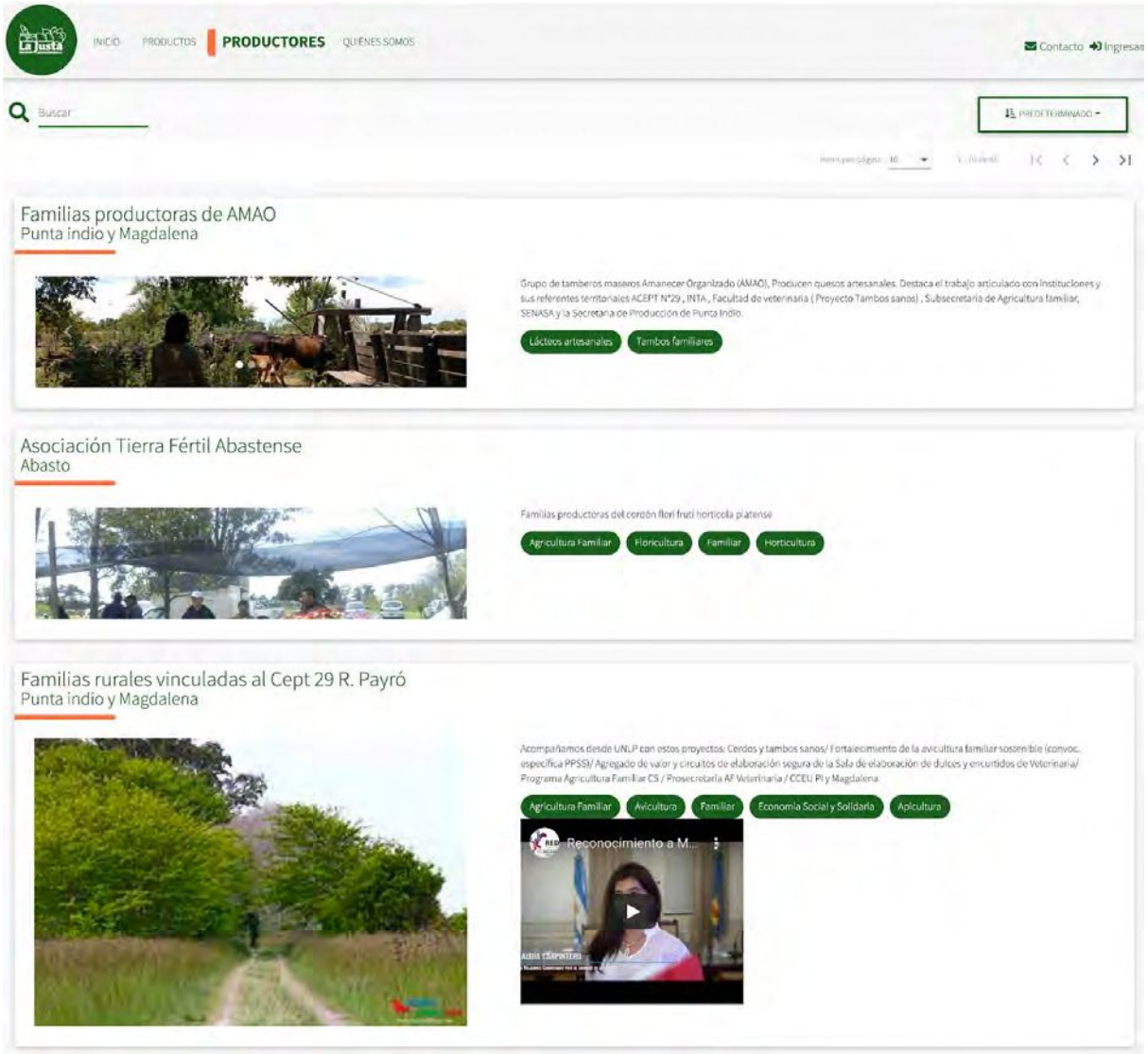
Si el sitio no está abierto para realizar pedidos, no se visualiza la canasta y la página muestra un mensaje debajo del header (Anexo figura 19) que dice cuando se va a abrir la tienda para poder comprar. Ejemplo de mensaje:

- *La tienda abrirá el viernes 10 de diciembre*

Anexo figura 19: Canasta no visible y mensaje de próxima ronda

PRODUCTORES

La página también tiene un apartado para los productores (Anexo figura 20) donde se pueden dar a conocer.

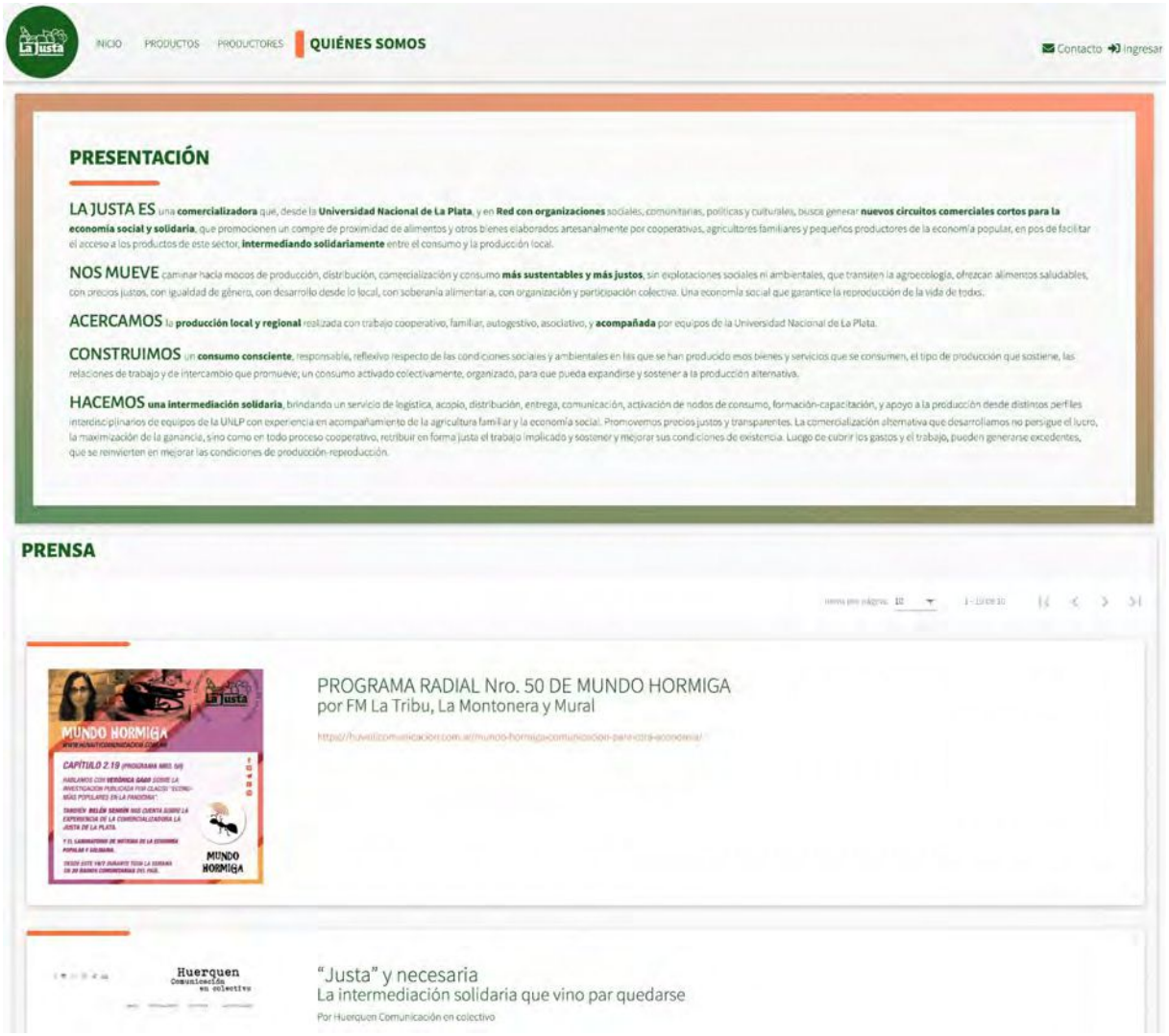


Anexo figura 20: Página productores

De cada productor se puede ver la información básica, sobre lo que producen, imagen/es sobre ellos y un video desde Youtube.

SOBRE NOSOTROS Y PRENSA

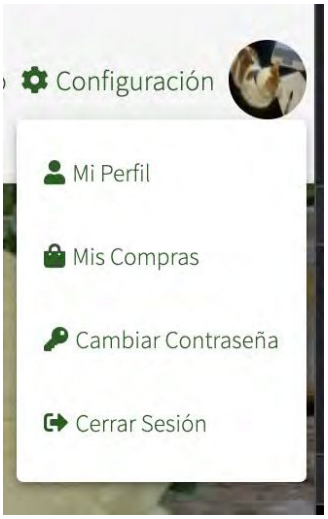
El portal también cuenta con un apartado donde se muestra una presentación sobre “La Justa” y una sección de prensa (Anexo figura 21), donde se informa sobre notas o entrevistas que se le realizan a la comercializadora.



Anexo figura 21: Página sobre nosotros y prensa

MENÚ USUARIO LOGUEADO

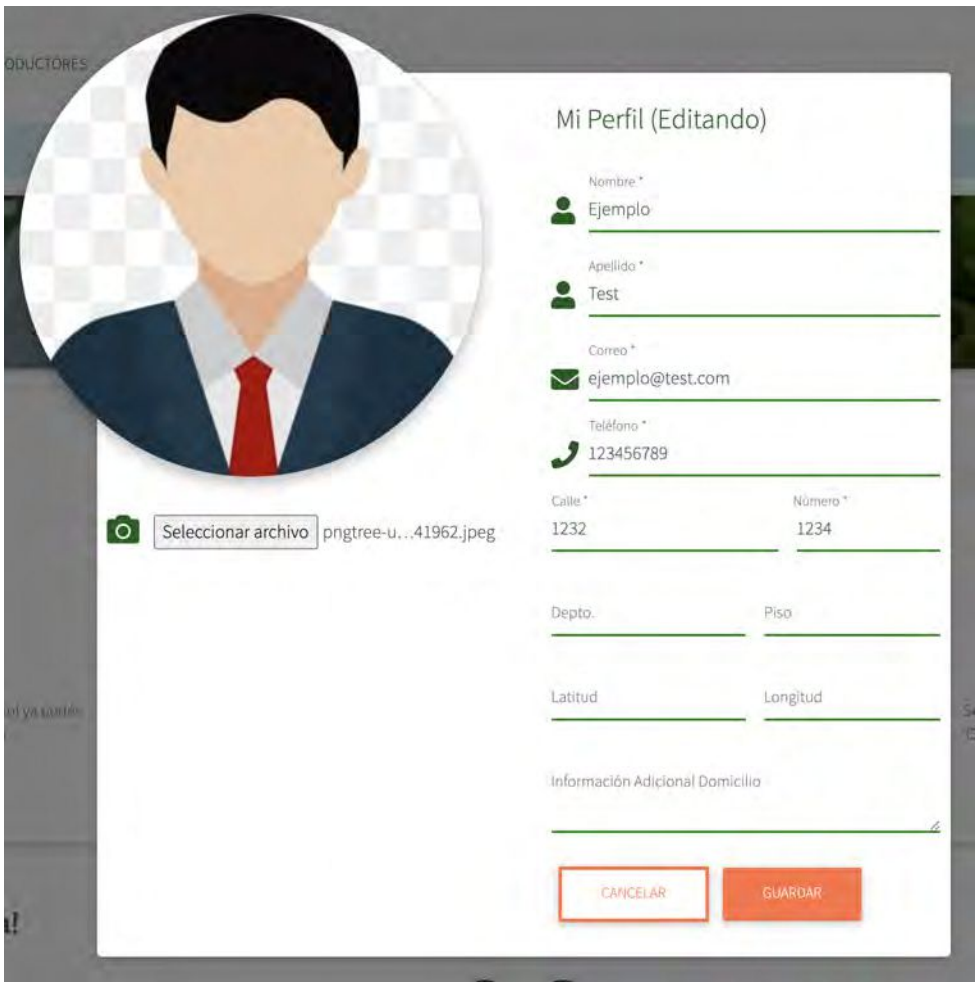
Los usuarios logueados al clickear en la imagen de su perfil se les despliega un menú (Anexo figura 22) con las siguientes opciones, editar su perfil, ver el historial de sus compras, cambiar la contraseña y cerrar la sesión.



Anexo figura 22: Menú usuarios logueados

Mi Perfil

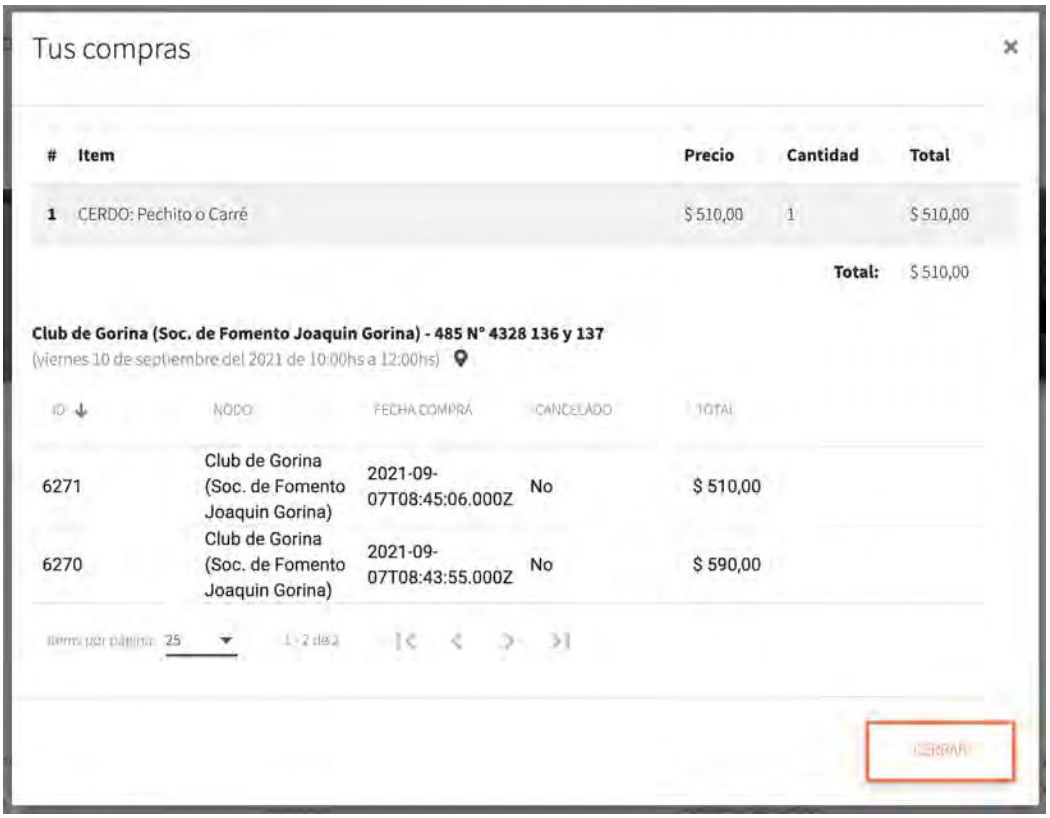
Al acceder a su perfil van a ver un modal desde donde pueden modificar su información (Anexo figura 23)



Anexo figura 23: Modal del perfil

Mis Compras

Al acceder a mis compras se pueden ver un modal (Anexo figura 24) que cuenta en la parte de arriba con el detalle de los productos de un pedido (en principio se visualiza la última compra realizada) y debajo se visualiza la lista de todas las compras realizadas siendo la primera de dicha lista la última que se realizó. Al hacer click en cada pedido se visualiza el detalle en la parte superior. Si la ronda se encuentra activa, y el pedido seleccionado corresponde a la misma, esta se puede modificar y también se puede modificar el nodo de entrega.



Anexo figura 24: Historial de compras

Cambiar Contraseña

Al cambiar contraseña, se le muestra un modal (Anexo figura 25)

Anexo figura 25: Modal cambio de contraseña

donde se le solicita que ingrese el correo, la contraseña actual y la nueva contraseña dos veces.

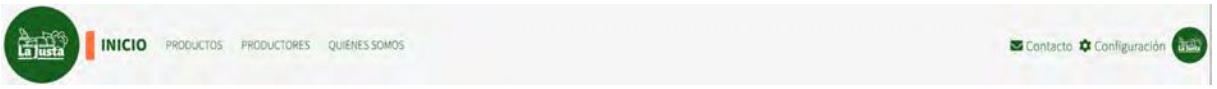
Cerrar Sesión

Al apretar en cerrar sesion se muestra un cartel de confirmación (Anexo figura 26)

Anexo figura 26: Modal cerrar sesión

CONFIGURACIÓN

Los usuarios administradores además de tener acceso a lo mismo que los consumidores, puede acceder a la configuración desde el menú (Anexo figura 27)



Anexo figura 27: Menú principal de usuario con rol administrador

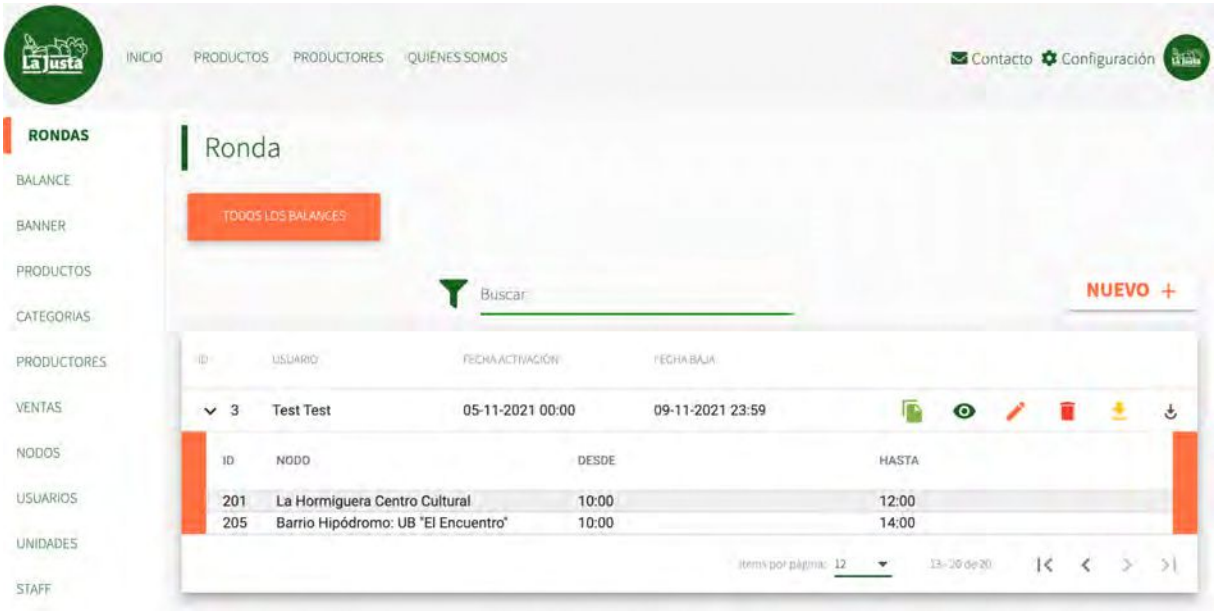
Dentro de la página de configuración se puede ver un sidebar, menú lateral, a la izquierda desde el que se tiene acceso a las distintas secciones configurables (Anexo figura 28).

Anexo figura 28: Sidebar página configuración

Todas las opciones cuentan con un listado, un buscador, un paginado configurable y la opción de ordenar por uno o más campos de las tablas, a excepción de Balances que solo cuenta con varias tablas a modo informativo y un buscador para cada una.

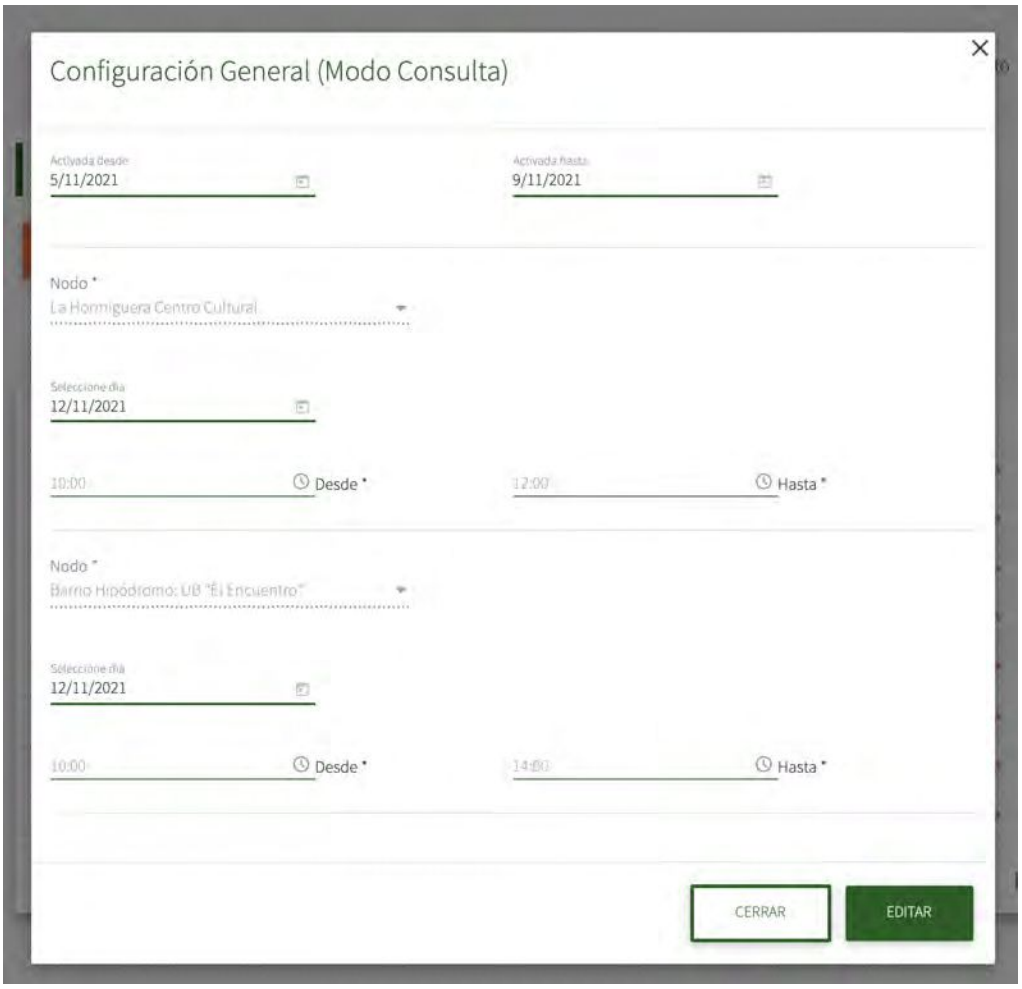
RONDAS

Es la página donde se configura la fecha de activación del sitio para que los usuarios puedan comprar y la fecha de baja junto con los nodos activos para realizar las entregas de los pedidos realizados(Anexo figura 29).



Anexo figura 29: Página configuración de rondas

La configuración de la Ronda (en un principio llamada General) permite crear una nueva, duplicar una existente, abrir en modo consulta, modo edición y eliminar. También se pueden descargar las ventas en un archivo excel que cuenta en la primera hoja con todas las ventas, y en las restantes, las ventas por cada nodo activo, con la información de los clientes que hicieron dichas compras y por último el balance de dicha ronda.
Al consultar (ejemplo Anexo figura 30), crear, editar o copiar se puede visualizar un modal con los datos a completar, con la fecha de activación y fecha límite del sitio para las ventas y la lista de nodos con días fechas y horarios posteriores a la fecha límite en los que se van a realizar las entregas. El modal es el mismo para todos los casos, solo cambia el estado y que en caso de ser uno nuevo, va a estar con todas las propiedades en blanco.



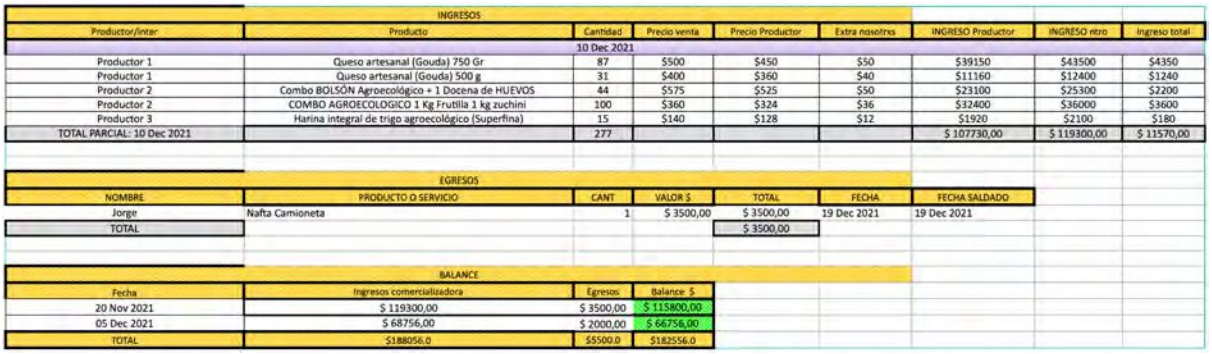
Anexo figura 30: Modal de Ronda en modo consulta

Si se aprieta en el botón de arriba que se visualiza en la página de configuración de la ronda (Anexo figura 29) que dice “TODOS LOS BALANCES”, se puede descargar un archivo excel que muestra información con los balances de todas las rondas (Anexo figura 31).

BALANCE			
Fecha	Ingresos comercializadora	Egresos	Balance $
20 Nov 2021	$ 119300,00	$ 3500,00	$ 115800,00
05 Dec 2021	$ 68756,00	$ 2000,00	$ 66756,00
TOTAL	$ 188056,00	$ 5500,00	$ 182556,00

Anexo figura 31: Ejemplo de todos los balances

También cada ronda cuenta con dos archivos para descargar, uno con todos los pedidos, una vista de resumen por productor, pago de productores y las ventas que van a entregar en cada nodo con los datos del usuario que realizo la compra (boton de color amarillo de descarga en la pagina de configuracion de ronda) https://docs.google.com/spreadsheets/d/1kkFYLepZJoWCX8mDm-I-DzbwrkrwoISd/edit?usp=sharing&ouid=109438985780127108592&rtpof=true&sd=true y otro con el balance de una ronda en particular en la que debajo se pueden ver todos los balances (Anexo figura 32).



INGRESOS

Productor/inter	Producto	Cantidad	Precio venta	Precio Productor	Extra nosotrxs	INGRESO Productor	INGRESO nitro	Ingreso total
10 Dec 2021								
Productor 1	Queso artesanal (Gouda) 750 Gr	87	$500	$450	$50	$39150	$43500	$4350
Productor 1	Queso artesanal (Gouda) 500 g	31	$400	$360	$40	$11160	$12400	$1240
Productor 2	Combo BOLSÓN Agroecológico + 1 Docena de HUEVOS	44	$575	$525	$50	$23100	$25300	$2200
Productor 2	COMBO AGROECOLOGICO 1 Kg Frutilla 1 kg zuchini	100	$360	$324	$36	$32400	$36000	$3600
Productor 3	Harina integral de trigo agroecológico (Superfina)	15	$140	$128	$12	$1920	$2100	$180
TOTAL PARCIAL: 10 Dec 2021		277				$ 107730,00	$ 119300,00	$ 11570,00

EGRESOS

NOMBRE	PRODUCTO O SERVICIO	CANT	VALOR $	TOTAL	FECHA	FECHA SALDADO
Jorge	Nafta Camioneta	1	$ 3500,00	$ 3500,00	19 Dec 2021	19 Dec 2021
TOTAL				$ 3500,00		

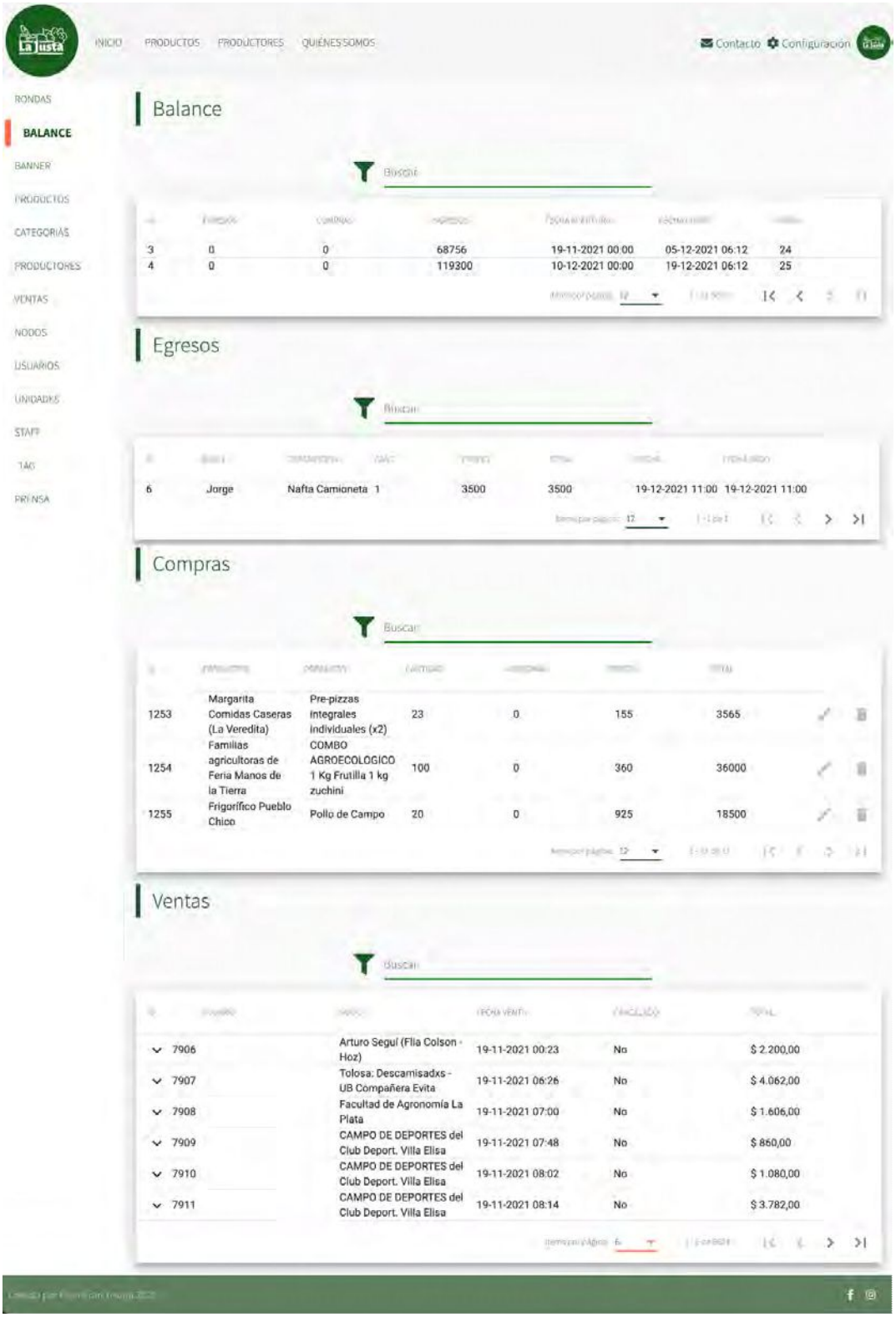
BALANCE

Fecha	Ingresos comercializadora	Egresos	Balance $
20 Nov 2021	$ 119300,00	$ 3500,00	$ 115800,00
05 Dec 2021	$ 68756,00	$ 2000,00	$ 66756,00
TOTAL	$188056.0	$5500.0	$182556.0

Anexo figura 32: Ejemplo de un balance de una ronda particular

BALANCE

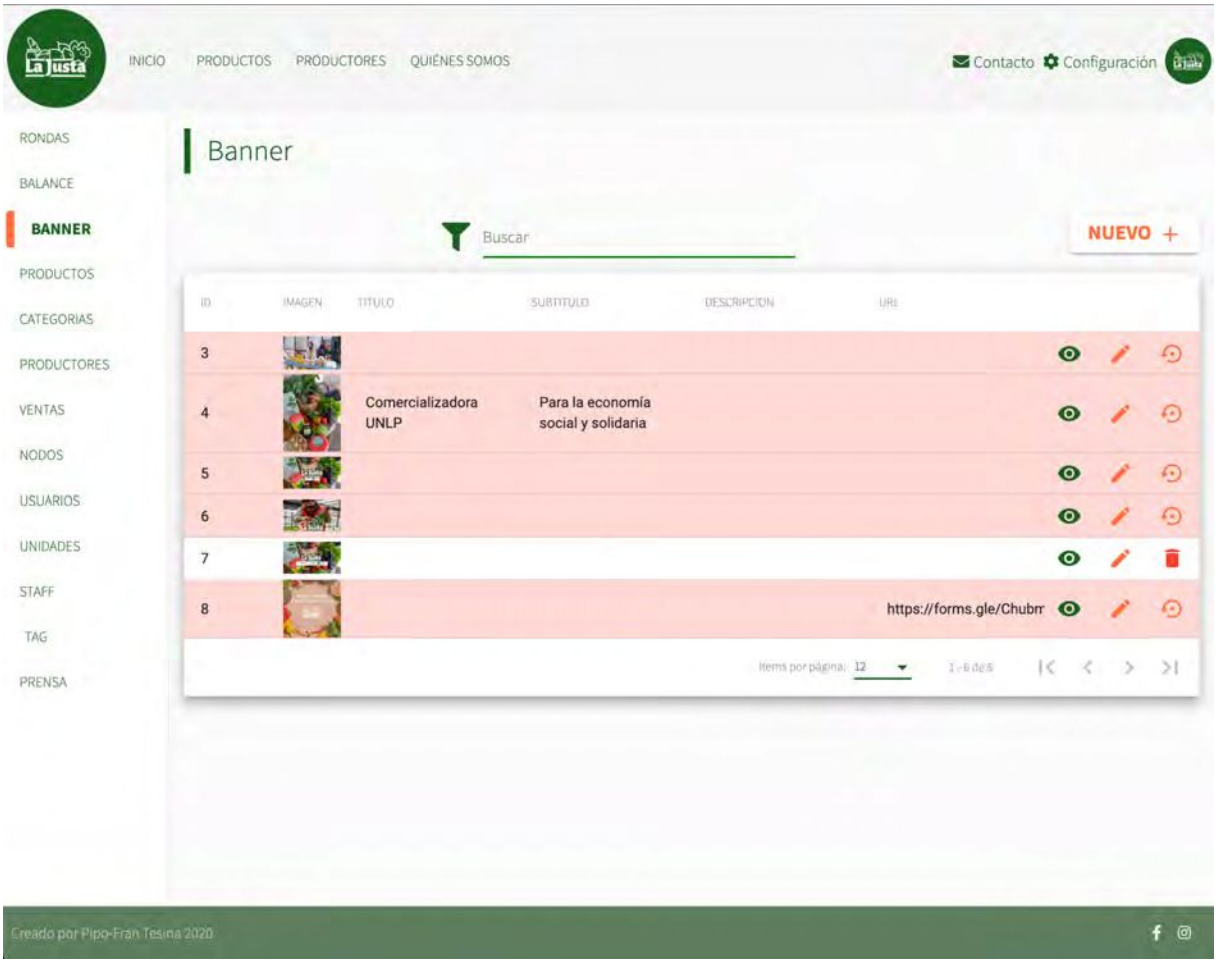
En esta sección se puede ver el balance en las tablas (Anexo figura 33), al seleccionar una fila de la tabla de balance correspondiente a una ronda, se actualizan las demás tablas con los egresos, compras y ventas.



Anexo figura 33: Página de los balance

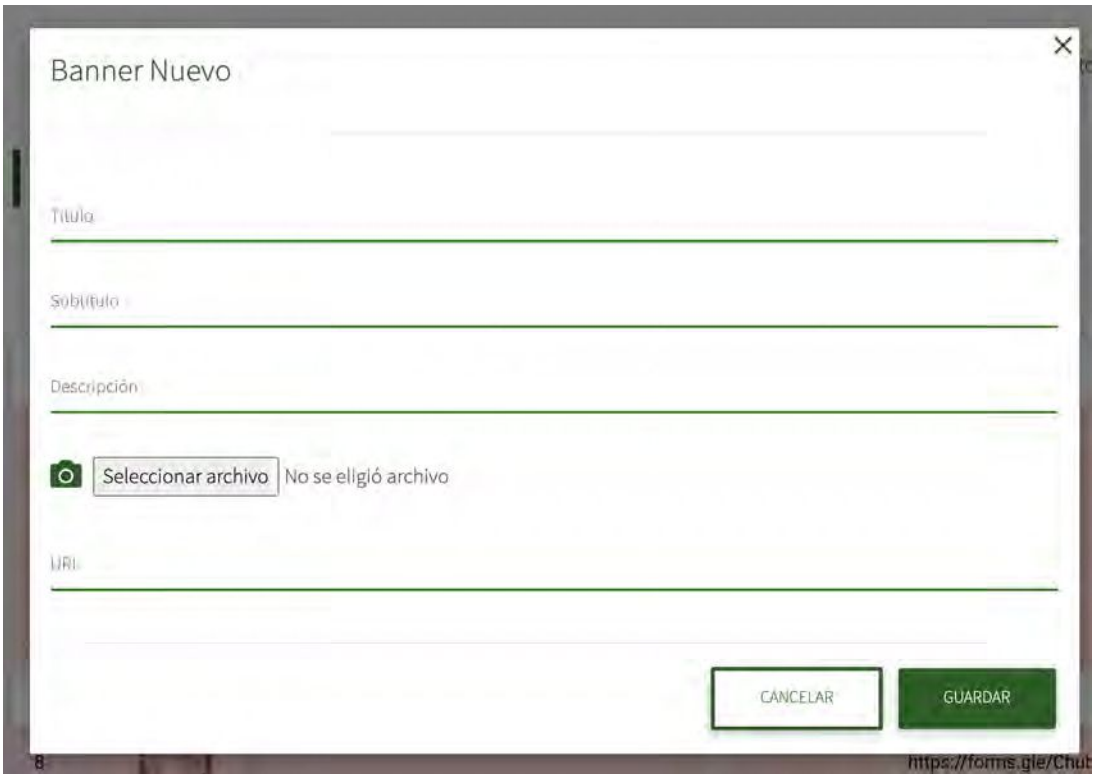
BANNER

En esta sección (Anexo figura 34) se cargan los banners que se encuentran en la home, a los mismos se les pueden agregar titulo, subtitulo, descripcion y hasta una url en caso de querer que al apretar en el banner se redirija a otra página.
Se puede crear un banner nuevo, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado.



Anexo figura 34: Página de configuración de banners

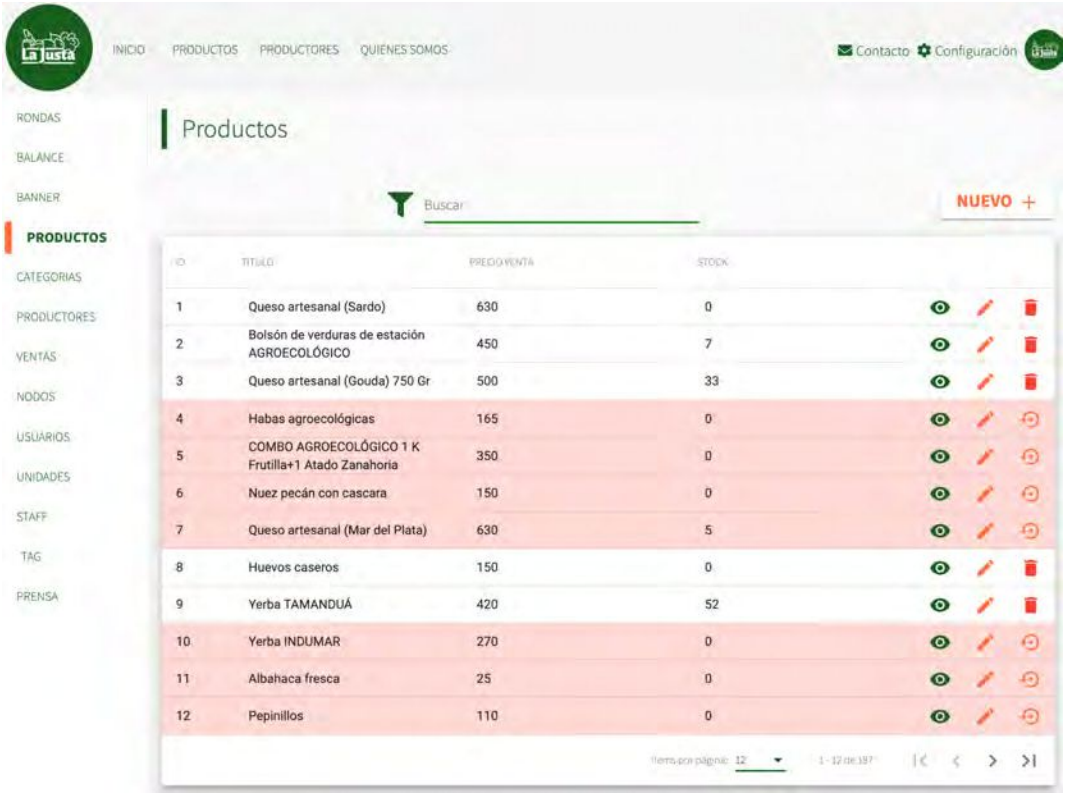
Al crear un banner nuevo, al editar o consultar, se visualiza el mismo modal, cambiando el estado de los campos y por supuesto, en caso de ser uno nuevo se visualiza con todos los campos en blanco (Anexo figura 35).



Anexo figura 35: Modal de banner (modo nuevo)

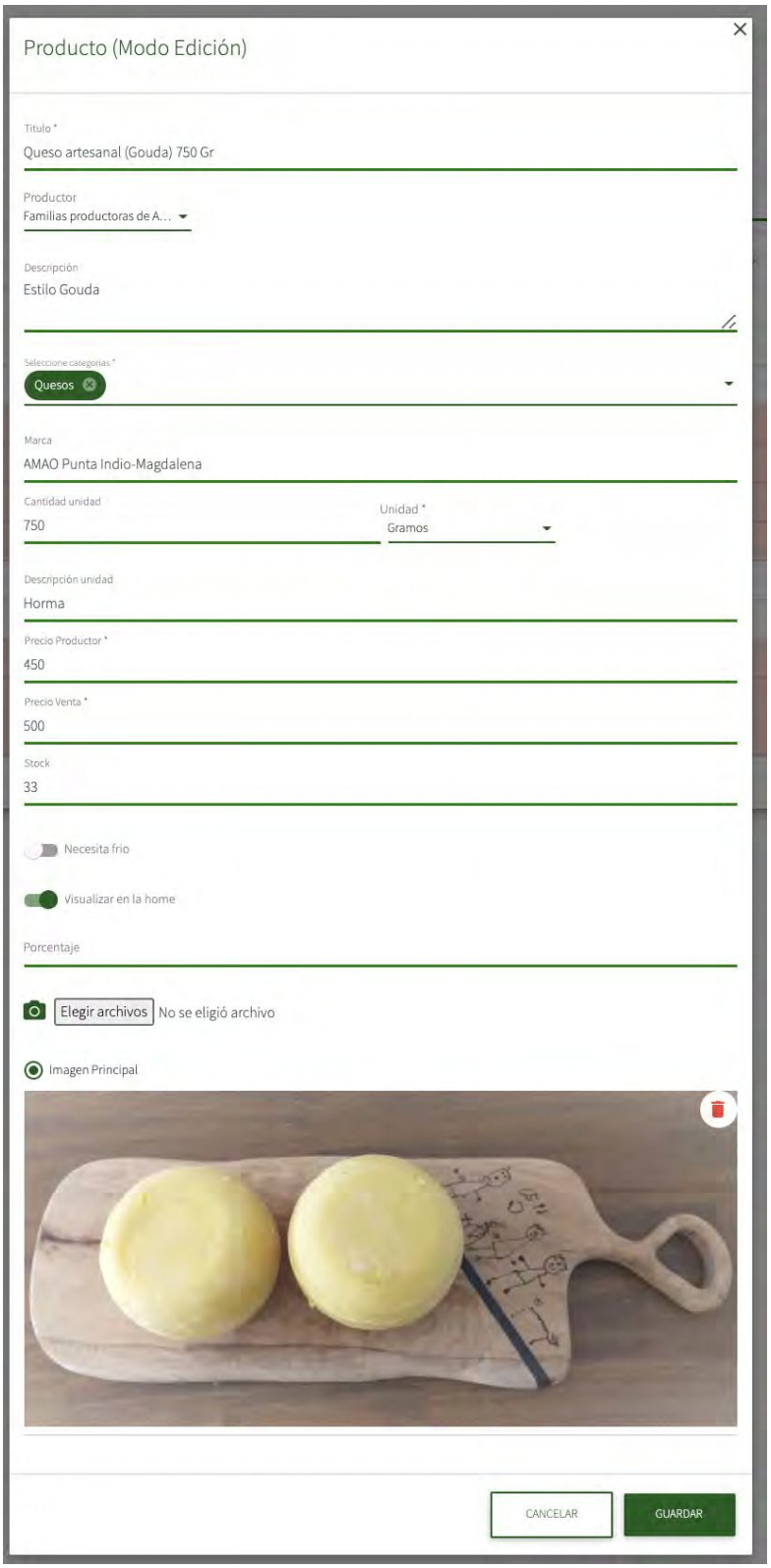
PRODUCTO

En esta sección se cargan los productos que se encuentran a la venta (Anexo figura 36).
Se puede crear un producto nuevo, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado.



Anexo figura 36: Página de configuración de banners

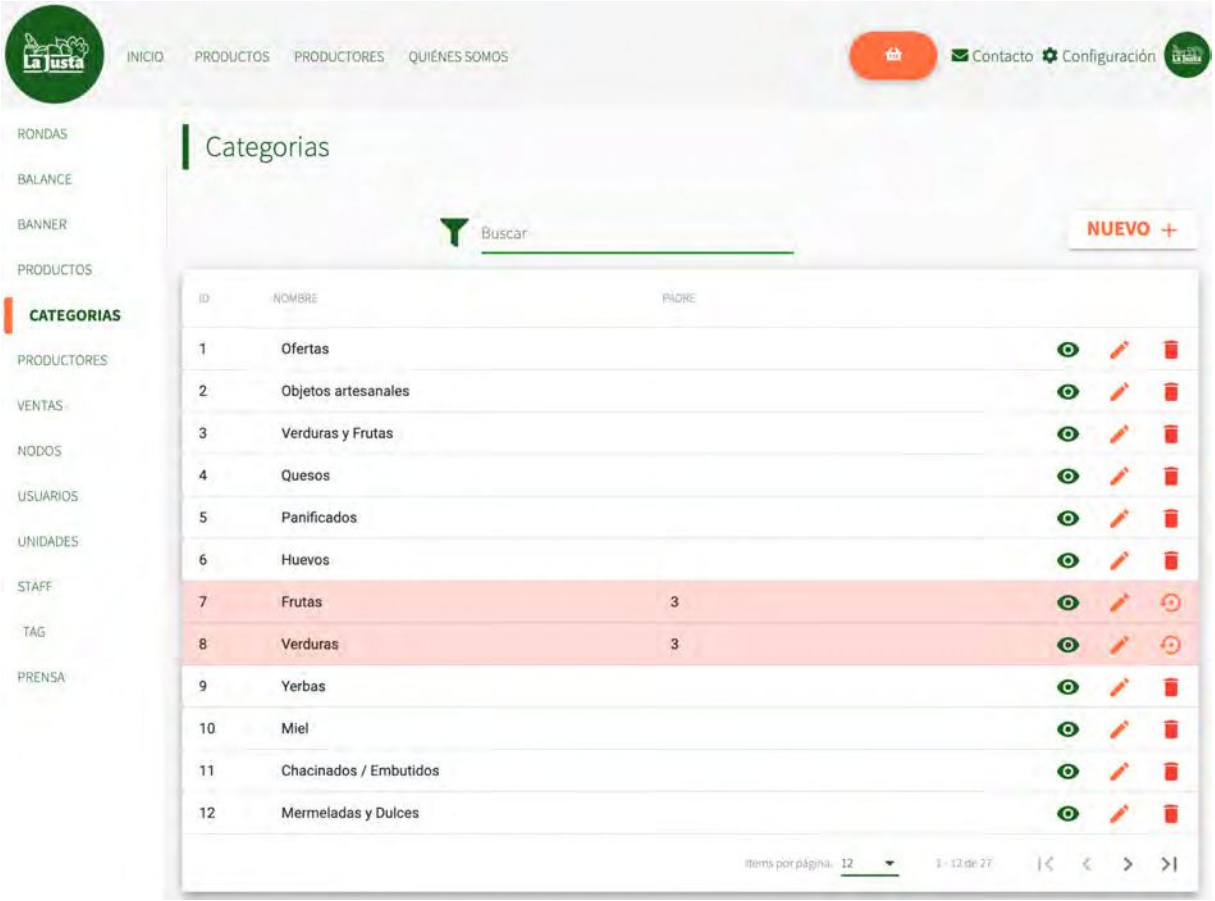
El modal del producto que se utiliza para la carga, edicion (Anexo figura 37) y/o visualización, cuenta con un título, productor, descripcion, categoria/s, marca, cantidad de unidad y unidad de medida, descripción de la unidad, precio de venta del productor, precio de venta al público, stock, si necesita frío, si se va a mostrar en la home, en el carrusel principal, porcentaje de descuento en caso de querer que se muestre un descuento sobre el producto, e imagen/es de las cuales una va a ser seleccionada como principal.



Anexo figura 37: Modal de producto (modo edición)

CATEGORÍAS

En apartadó (Anexo figura 38) se cargan las categorías que se utilizan para seccionar los productos y cuenta con las opciones de crear una categoría nueva, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, la misma puede ser restaurada.



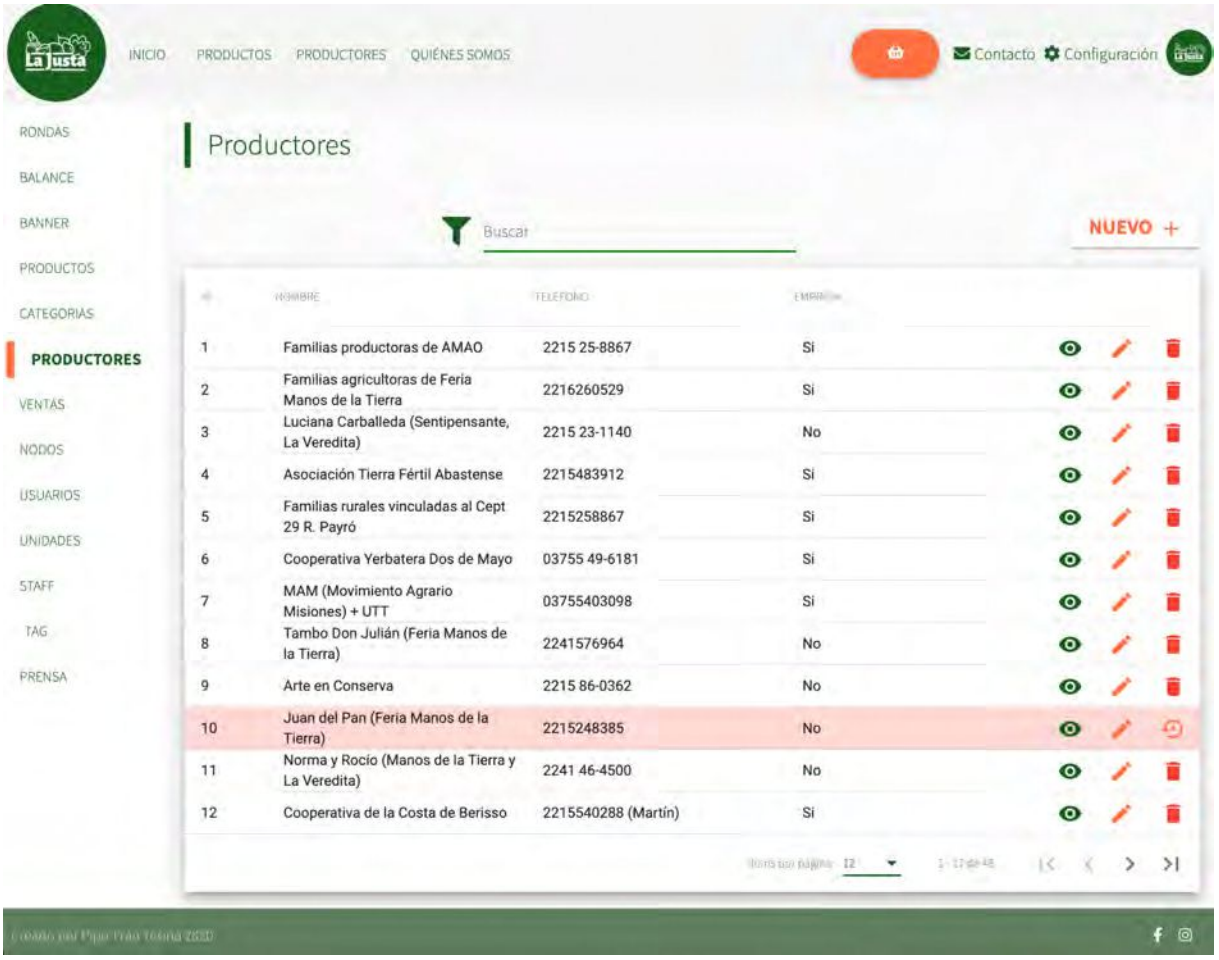
Anexo figura 38: Página de configuración de categorías

El modal utilizado para la visualización (Anexo figura 39), alta o modificación de una categoría, cuenta con el nombre que es obligatorio, una categoría padre (opcional) y una imagen, que va a ser usada para la parte visual de la tarjeta del producto.

Anexo figura 39: Modal de categoría (modo consulta)

PRODUCTORES

En la vista de configuración de productores (Anexo figura 40), se puede crear un producto nuevo, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado.



Anexo figura 40: Página de configuración de Productores

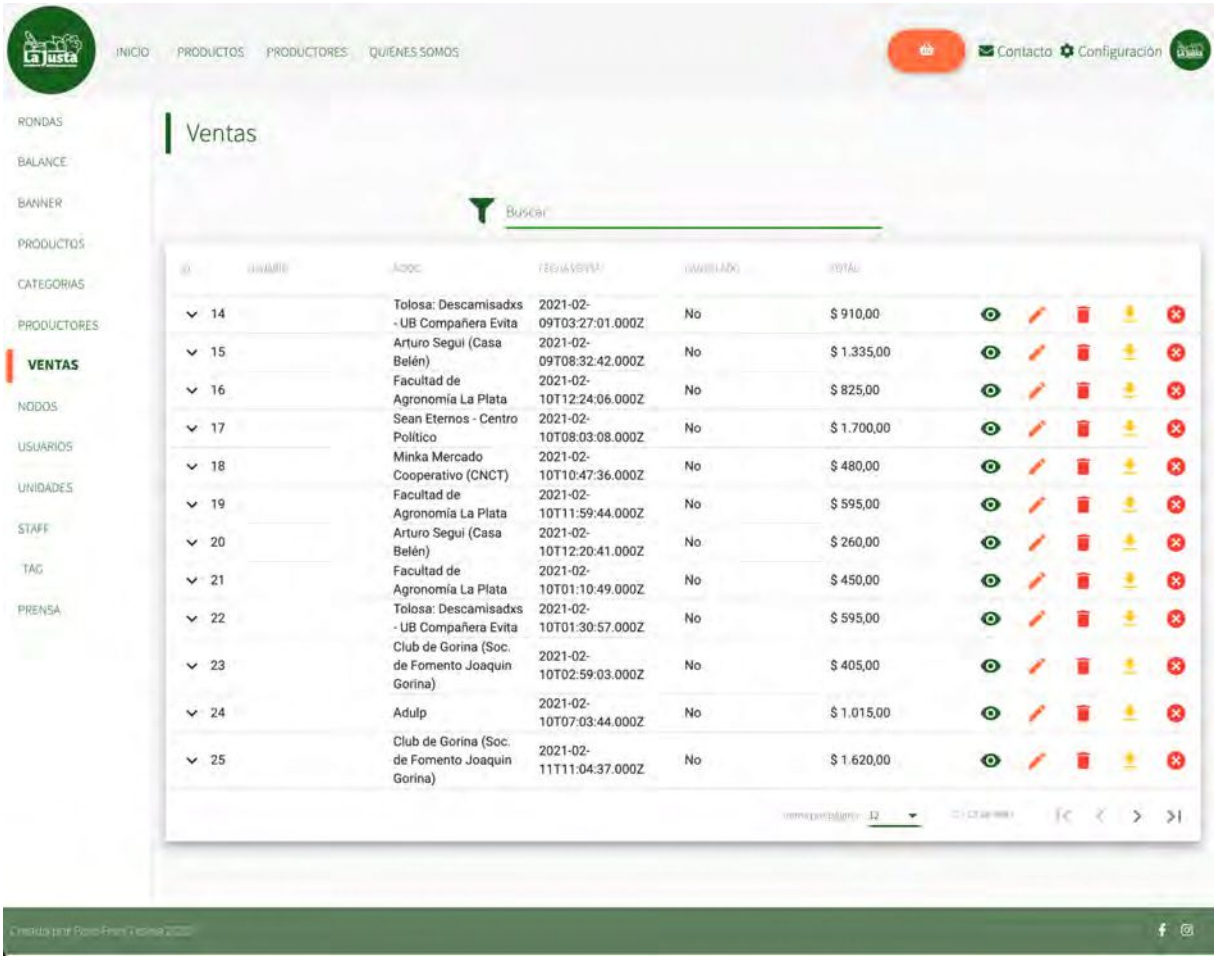
Cada productor cuenta con los siguientes campos en el modal que utilizamos para consultar (Anexo figura 41), editar o dar de alta, nombre (obligatorio), procedencia, calle, número, dpto, piso, longitud, latitud, información adicional al domicilio, teléfono (obligatorio), selección de tags (obligatorio, utilizado para palabras claves que se visualizan en la página de productores), correo, descripción (información adicional referente al productor), imagen/es, video (si es que cuentan con algún video en youtube) y un switch que dice si el productor es una organización

Anexo figura 41: Modal de productor (modo consulta)

VENTAS

En esta sección se pueden visualizar todas las ventas realizadas (Anexo figura 42), vale aclarar que desde acá no se puede realizar una nueva venta, y se puede consultar, editar (solo modificar el punto de entrega y las cantidades de los productos que un usuario haya

podido siempre y cuando la ronda de ventas se encuentre activa), eliminar la compra, descargar el archivo en pdf que se les envía al usuario al realizar una compra y por último cancelar el pedido (muestra un cartelito para confirmar si esta seguro de cancelar la entrega).
Al cancelar, se muestra en rojo dentro del listado y en el reporte de compra no aparece.
Al eliminar se filtran al generar el balance TOTAL todos los pedidos que no fueron eliminados.
Un pedido cancelado puede aparecer en un reporte de compra, mientras que uno eliminado no, ahora, el cancelado nunca aparece en el reporte de balance positivo, dado que no se vendió.



Anexo figura 42: Página de configuración de ventas

Al apretar en cancelar pedido se visualiza un cartel para confirmar si realmente se desea cancelar (Anexo figura 43)

Anexo figura 43: Ventana emergente de confirmación al cancelar un pedido

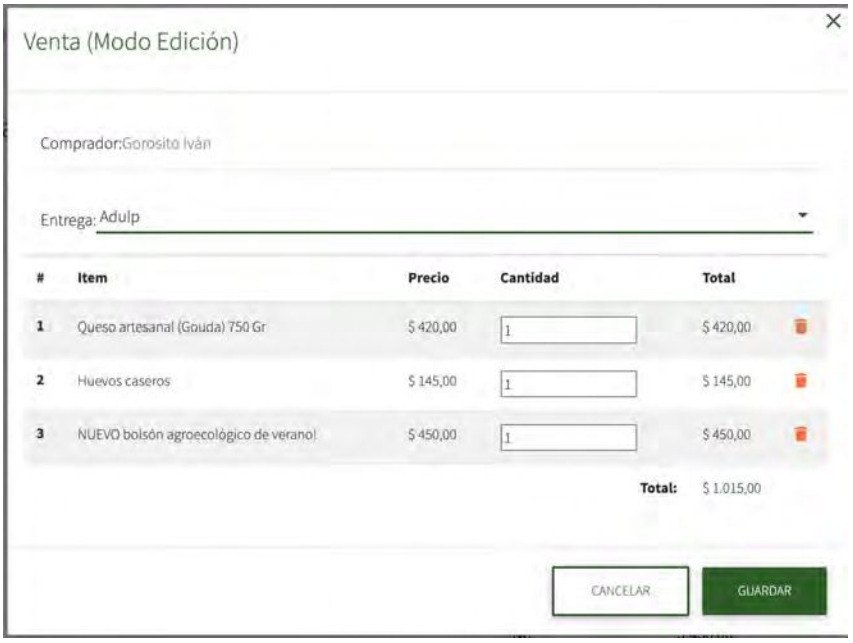
A continuación se puede ver un ejemplo del PDF (Anexo figura 44) que se les envía por correo a los usuarios una vez realizada la compra a través del sitio con los productos, su precio y total a abonar.

Detalle de su compra realizada el dia 10/02/2021

Producto	Cantidad	Precio
Queso artesanal (Gouda) 750 Gr	1	420
NUEVO bolsón agroecológico de verano!	1	450
Huevos caseros	1	145
TOTAL	3	1015.0

Anexo figura 44: PDF de ejemplo enviado a los usuarios luego de realizar una compra

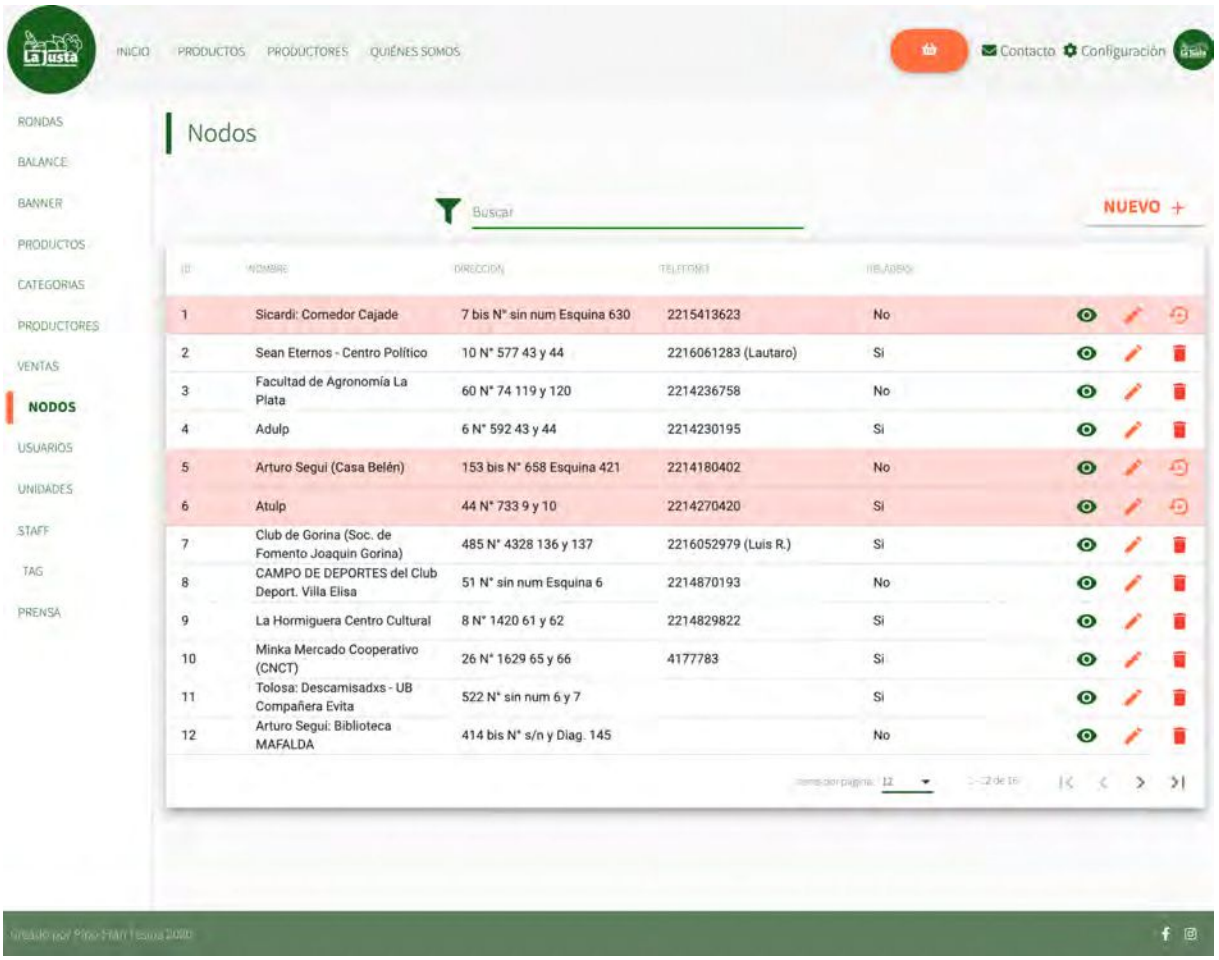
El modal que se muestra al apretar en la edición (Anexo figura 45) o consulta muestra el nodo de entrega con un listado de los productos comprados, su cantidad, monto y la opción de eliminar alguno.



Anexo figura 45: Modal de venta (modo edición)

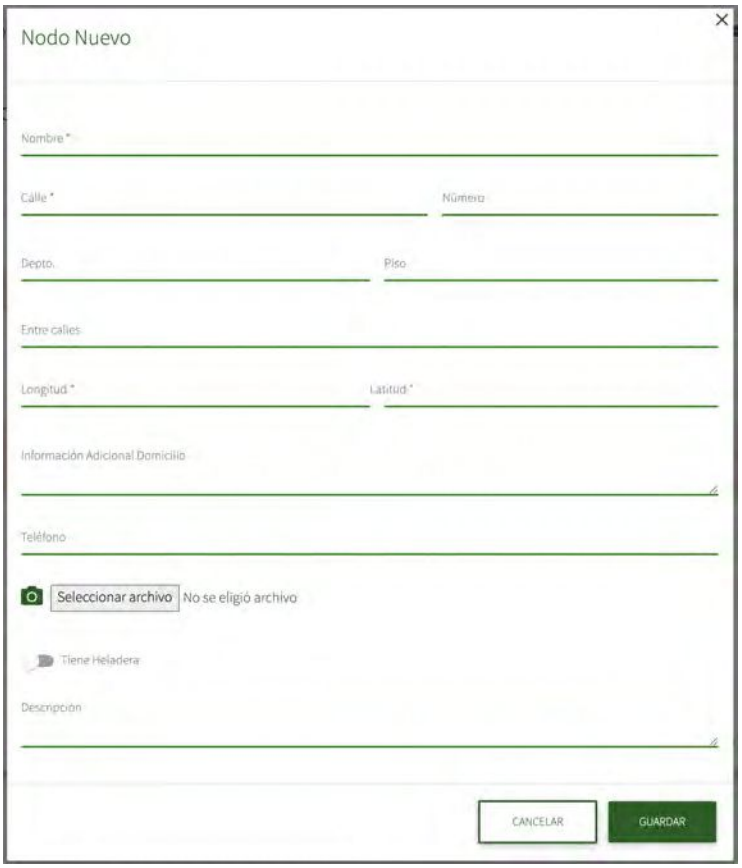
NODOS

En este apartado se pueden crear los puntos de entrega, llamados nodos (Anexo figura 46), donde se entregan las órdenes, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado



Anexo figura 46: Página de configuración de nodos

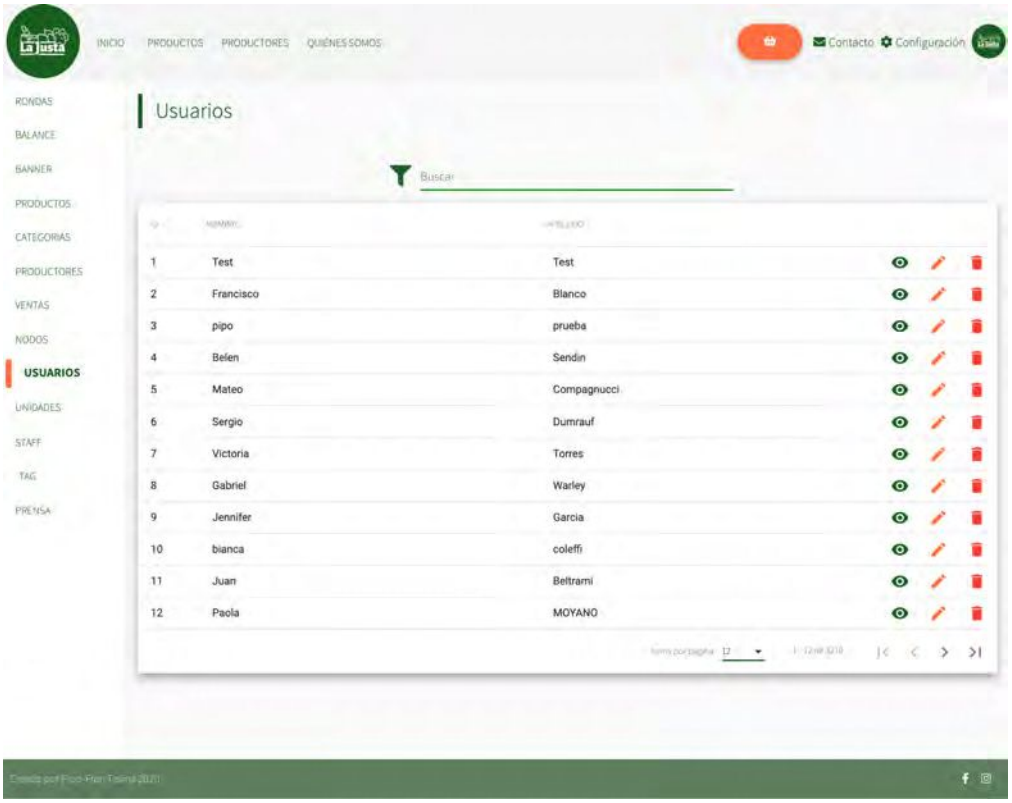
Cada Nodo cuenta con los siguientes campos que deben ser completados al crear uno nuevo (Anexo figura 47), editar o abrir en modo consulta, siendo nombre (obligatorio), calle(obligatorio), número, dpto, piso, entre calles, longitud, latitud, información adicional al domicilio, teléfono, imagen, un switch que dice si el nodo cuenta con heladera y una descripción adicional sobre el nodo.



Anexo figura 47: Modal de nodo nuevo

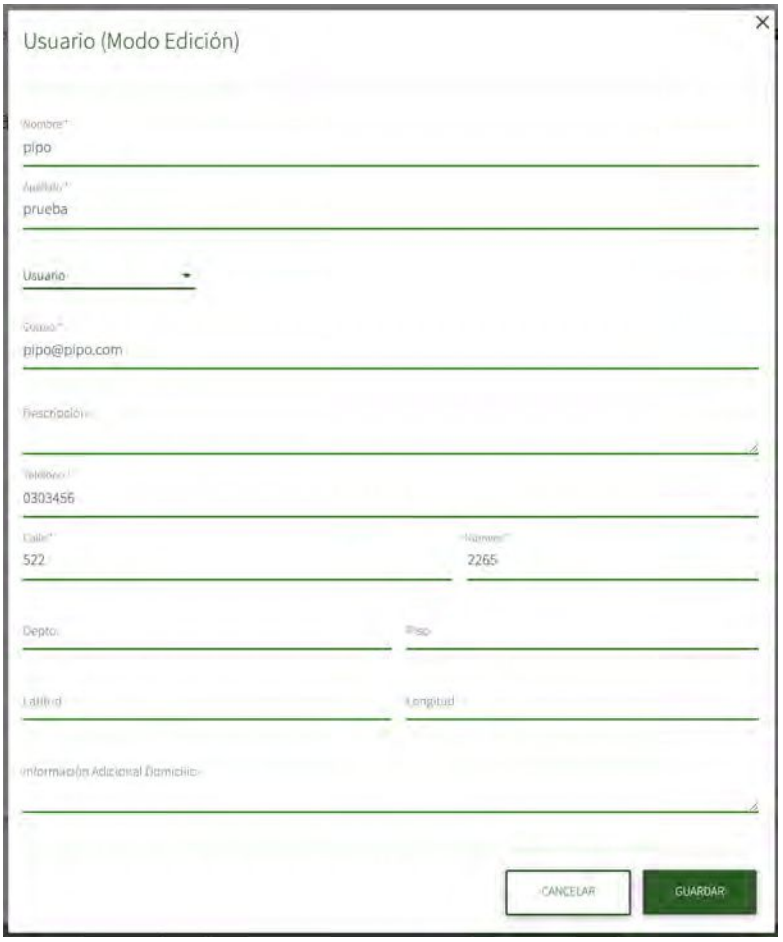
USUARIOS

En este sector se ven los usuarios registrados (Anexo figura 48), se pueden abrir en modo consulta, editar o eliminar de manera lógica. En caso de estar eliminado, el mismo puede ser restaurado.



Anexo figura 48: Página de configuración de usuarios

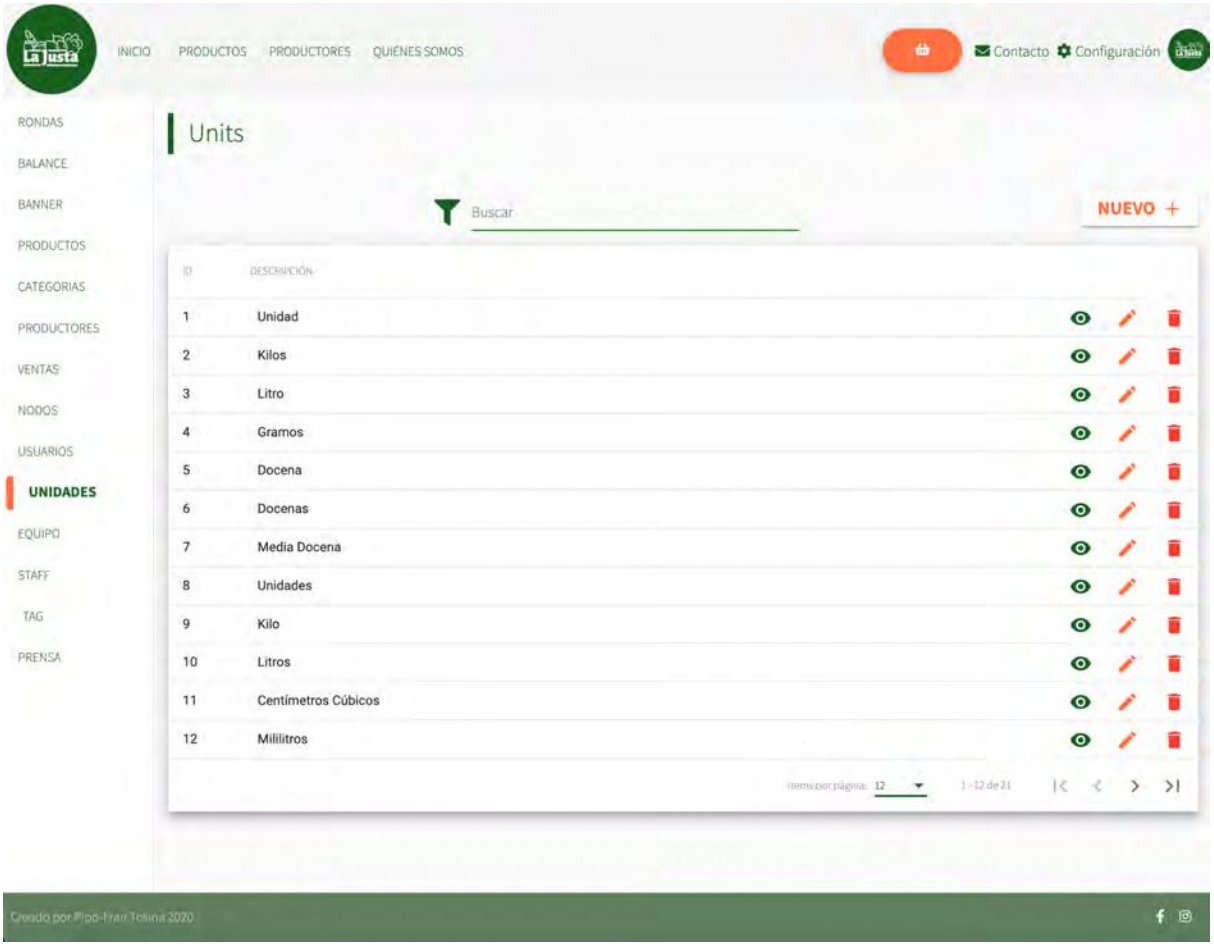
En modal utilizado para la edición (Anexo figura 49), se le pueden realizar cambios a cualquier campo del usuario, nombre, apellido, rol (administrador o usuario comun), correo, descripción, teléfono, calle, número, dpto, piso, latitud, longitud e información adicional a la dirección, también se puede visualizar el mismo modal en modo consulta.



Anexo figura 49: Modal de usuario (modo edición)

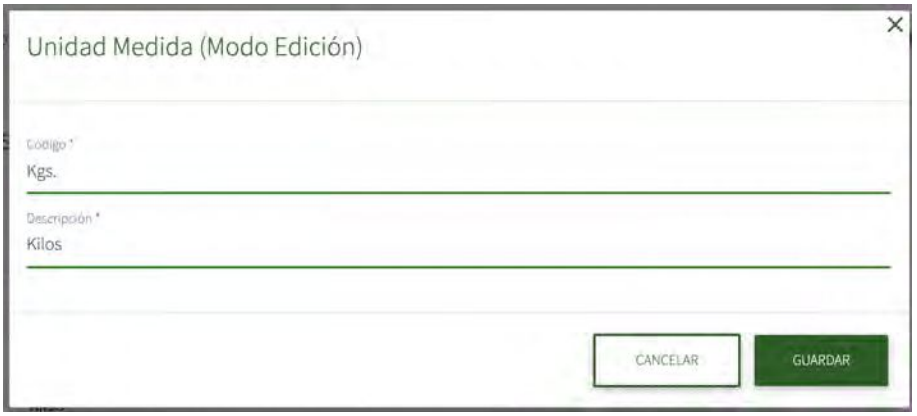
UNIDADES

En esta página se cargan las unidades que se utilizan en la carga de los productos para identificar el tipo de unidad de peso (Anexo figura 50). Se puede crear una unidad nueva, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado



Anexo figura 50: Página de configuración de unidades

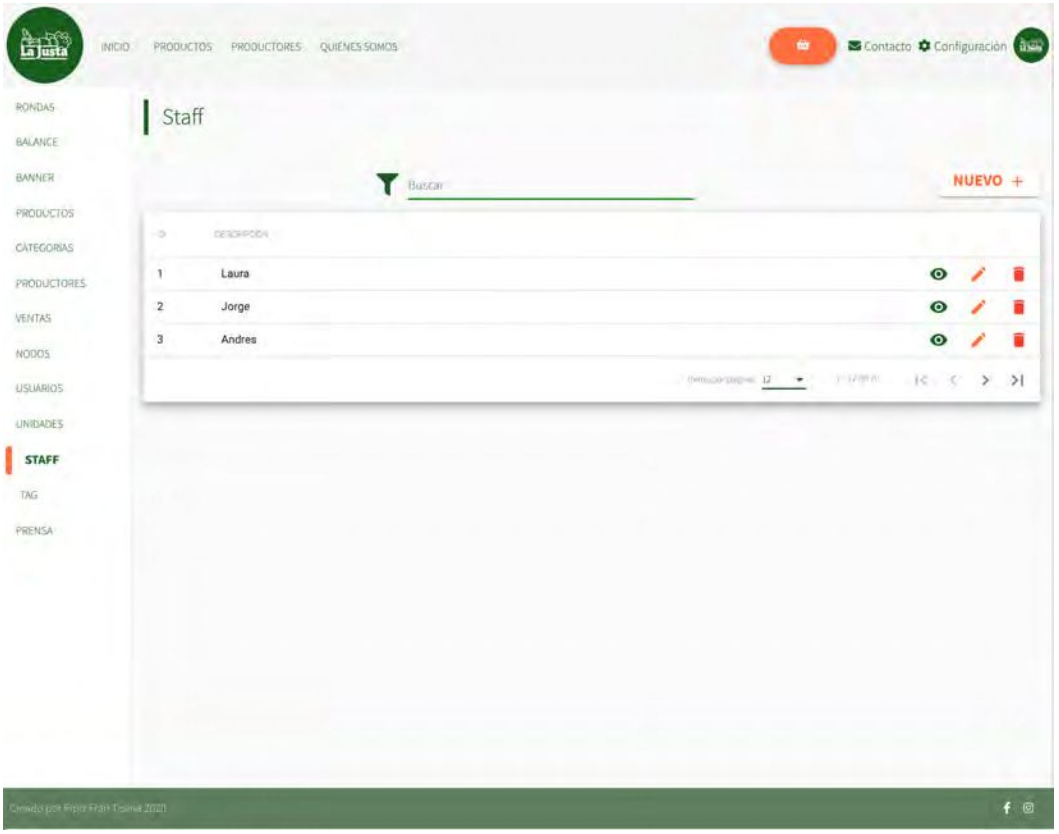
Cada unidad de medida está compuesta por un código y una descripción y cuenta con un modal que puede adoptar la forma en modo de consulta, alta o edición (Anexo figura 51).



Anexo figura 51: Modal de unidad medida (modo edición)

STAFF

Este panel de configuración es para las personas que son parte del staff (Anexo figura 52) y realizan algún gasto, como por ejemplo la carga de combustible para los vehículos que reparte la mercadería entre los nodos o la compra de bolsas, para poder ser identificado dicho gasto generado y poder hacerle devolución del monto, es que se utiliza esta tabla. Se puede crear un elemento nuevo, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado



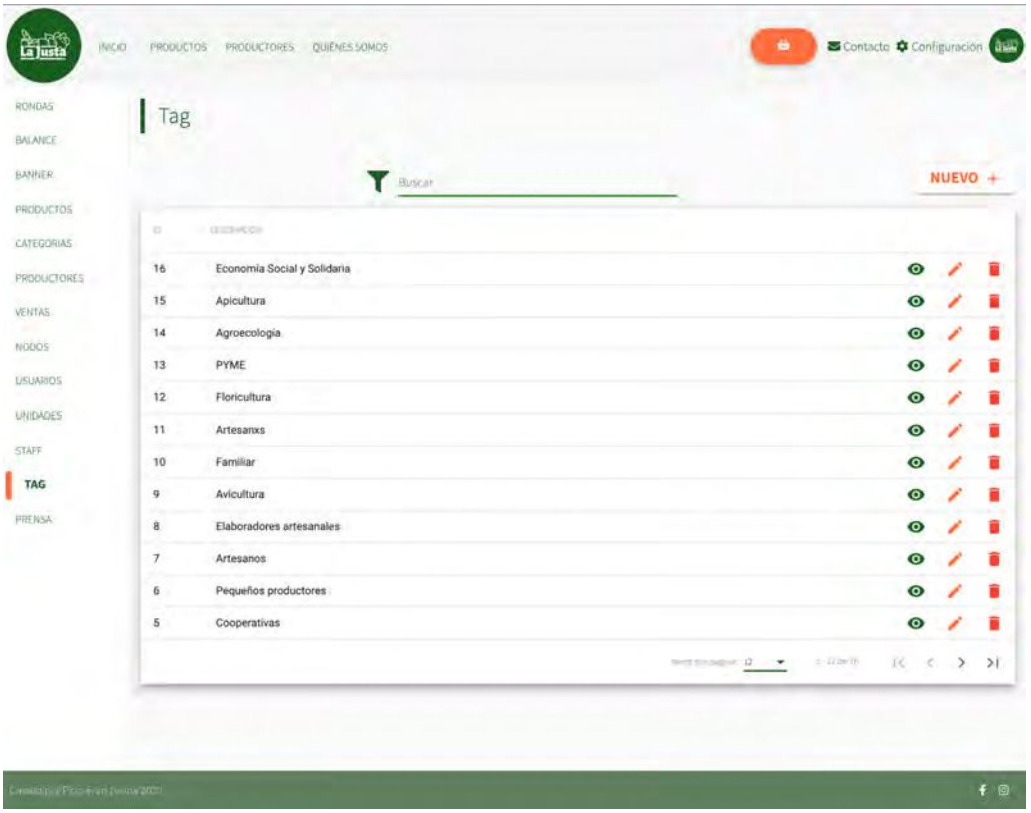
Anexo figura 52: Página de configuración de staff

El modal que se utiliza para dar de alta (Anexo figura 53), editar o modificar solo consta de un campo nombre que es obligatorio.

Anexo figura 53: Modal de staff (modo nuevo)

TAG

Esta página de configuración cuenta con una tabla en la que se pueden ver los tags habilitados para poder ser usados en los productores (Anexo figura 54), que hacen referencia al tipo de producto y/o organización que provee de la mercadería. Se puede crear un tag nuevo, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado



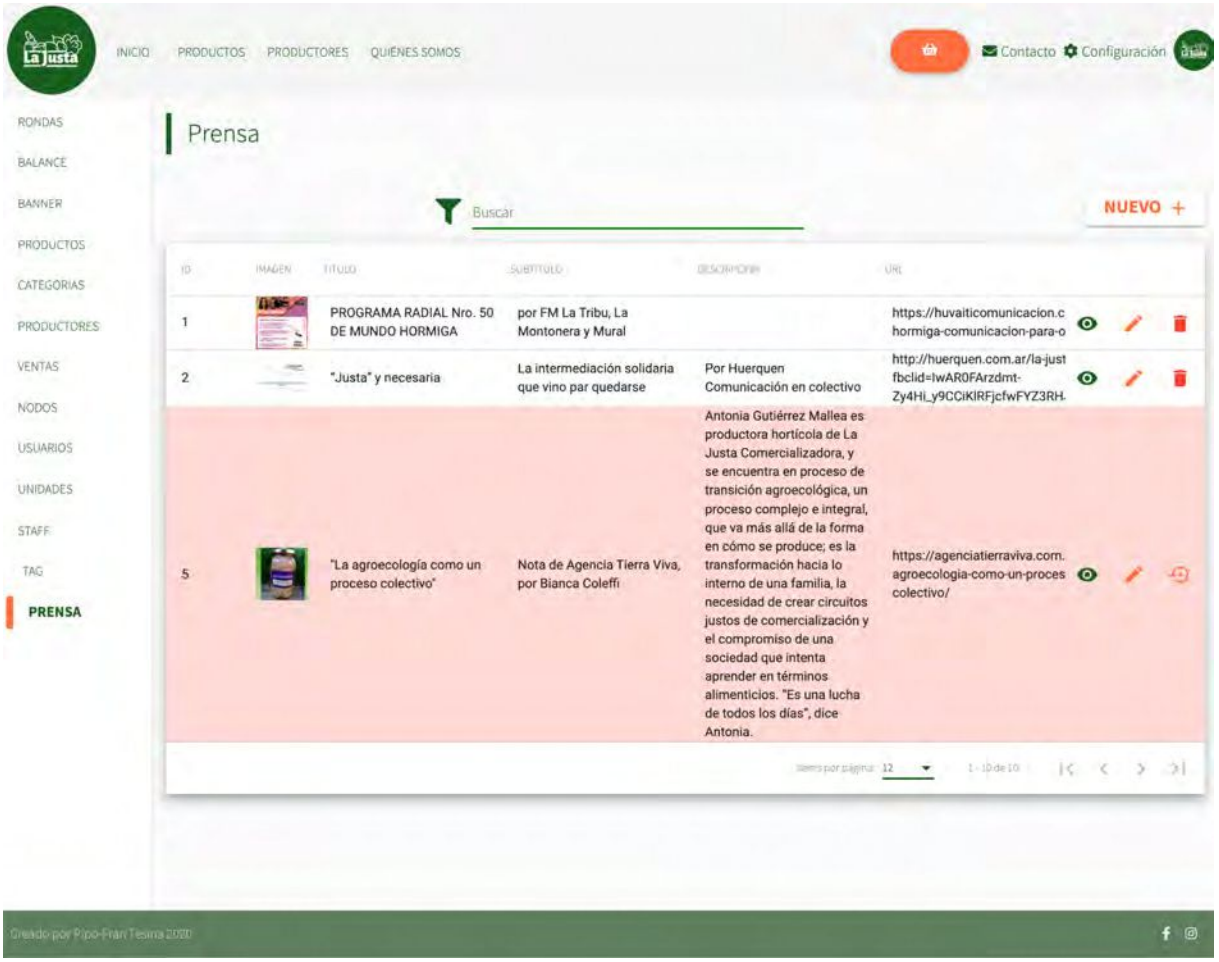
Anexo figura 54: Página de configuración de tags

El modal utilizado para generar un nuevo tag, editar (Anexo figura 55) o consultar, solo consta de un campo nombre que es obligatorio.

Anexo figura 55: Modal de tag (modo edición)

PRENSA

En este apartado se carga todo lo relacionado con las entrevistas, notas e información sobre La Justa (Anexo figura 56), la misma se ve reflejada en la página que hace referencia a quienes somos. Se puede crear una prensa nueva, abrir en modo consulta, editar o eliminar. En caso de estar eliminado, el mismo puede ser restaurado



Anexo figura 56: Página de configuración de prensa

El modal que se utiliza para consultar (Anexo figura 57), cargar y/o modificar cuenta con los siguientes campos: título, subtítulo, descripción, imagen y URL

Anexo figura 57: Modal de prensa (modo consulta)

PAQUETES ADICIONALES PARA EL DESARROLLO DEL FRONTEND

Los paquetes que Angular nos provee por defecto para su funcionamiento que son los siguientes:

"@angular/animations": "~9.0.5",
"@angular/common": "~9.0.5",
"@angular/compiler": "~9.0.5",
"@angular/core": "~9.0.5",
"@angular/forms": "~9.0.5",
"@angular/platform-browser": "~9.0.5",
"@angular/platform-browser-dynamic": "~9.0.5",
"@angular/router": "~9.0.5",
"rxjs": "~6.5.4",
"tslib": "^1.10.0",
"zone.js": "~0.10.2"

Tuvimos la necesidad de instalar los siguiente paquetes:

@angular/flex-layout para manejar la parte responsive y los distintos tamaños en los que se va a visualizar la aplicación

"@angular/cdk": "^9.1.1", y "@angular/material": "^9.2.4", para la parte visual, para la utilización de angular material, era necesario para el angular boostrap md.

"@angular/youtube-player": "^9.1.1", para la reproducción de videos de youtube en el sitio.

"@fortawesome/fontawesome-free": "^5.6.3", para la utilización de los iconos

"angular-bootstrap-md": "9.3.0", Es un angular boostrap con material design para la parte visual de la página.

"chart.js": "^2.5.0", Se utilizó para gráficos con canvas.

"classlist.js": "^1.1.20150312", Para el acceso a la lista de elementos a través de javascript

"hammerjs": "^2.0.8", Para la deteccion de acciones en el sitio

"jwt-decode": "^3.0.0", Para el manejo de jwt (JSON Web Token) es un estándar para el manejo de sesiones y usuarios

"mapbox-gl": "^1.10.1", Para la utilización de una librería con los mapas, para mostrar los nodos de entrega

"ng-sidebar": "^9.2.1", Para el menú lateral de la aplicación

"ngx-material-timepicker": "^5.5.3", Para la selección de fecha y hora

"ngx-number-spinner": "^2.2.1", Para incrementar y decrementar la cantidad de elementos a comprar

"ngx-toastr": "^12.0.1", Para mostrar los carteles informativos al usuario

"web-animations-js": "^2.3.2", Animaciones adicionales