UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Herramienta basada en el navegador Web para el reporte de errores en el contexto de sistemas Web

AUTORES: Otero Llambay Rodrigo y Seoane Tomás Ernesto

DIRECTOR: Sergio Firmenich

CODIRECTOR: Alejandra Garrido

ASESOR PROFESIONAL:

CARRERA: Licenciatura en Sistemas

Resumen

Esta tesina se enfoca en mejorar el proceso de reporte de errores en sistemas web, el cual cumple un rol central en el proceso de desarrollo de software. Los usuarios encargados de realizar estos reportes suelen ser ajenos al ámbito de la informática y desconocen qué información es útil para la identificación de los errores, lo que deriva en un retraso en la resolución del mismo. Es importante que la información brindada por el usuario ayude con la rápida y correcta resolución de los errores. Para ello desarrollamos una extensión de navegador que recopila información del contexto de la navegación del usuario que será procesada y anexada al reporte que el usuario realiza normalmente. Esta herramienta fue probada por desarrolladores a los cuales se los hizo identificar errores reportados con y sin la herramienta desarrollada para probar la eficacia en cuanto al tiempo requerido para la identificación de los errores.

Palabras Clave

Reporte de errores, Desarrollo de software, Bug, Chrome, Extension de Chrome, HTML, Issue, JavaScript, Gestor de Proyecto, Trello, Basecamp, Jira, Testing, Captura de pantalla, Páginas web, Automatización, Plugin, QA.

Conclusiones

Contar con una herramienta de este estilo quita responsabilidad sobre el reportador al recopilar la información automáticamente y procesarla para generar un mejor reporte de errores, ya que no alcanza con especificarle al reportador qué información es de utilidad o siempre debe ser incluida en los reportes porque no toda esta información es de fácil acceso. Mejora los tiempos de resolución de bugs y errores en sistemas web ya que el desarrollador identifica más rápido los errores y reduce significativamente la comunicación entre desarrollador y reportador.

Trabajos Realizados

-Investigación sobre la problemática en el proceso de reporte e identificación de errores en sistemas web.

-Desarrollo de una extensión de navegador para la captura automática de información contextual de la navegación y su posterior reporte en gestor de proyecto.

-Evaluación de desempeño de la herramienta desarrollada.

Trabajos Futuros

-Permitir editar y realizar anotaciones sobre las capturas de pantalla.

-Posibilitar la creación de mockups desde la extensión.

-Grabar navegación para posterior ejecución mediante herramientas de automatización.

-Mejorar la visualización de la información.

-Integración con más gestores de proyectos.

Fecha de la presentación: 11/2020

Agradecimientos

Queremos agradecer en principio a nuestras familias por el incentivo y el apoyo tanto moral como económico para que podamos enfocarnos en nuestros estudios. A nuestros compañeros de la facultad por todo el conocimiento compartido. A Sergio y Alejandra, por la ayuda, paciencia y motivación durante todo el proceso de desarrollo de este trabajo. Y por ultimo a la UNLP, por su condición de publica y gratuita y por darnos las bases para poder formarnos como profesionales.

Índice general

Capítulo 1

Introducción

1.1. Motivación

El reporte de bugs [1] y errores en el sistema es algo muy común en el proceso de desarrollo de software y cumple un rol central, ya que permite a los usuarios y desarrolladores reportar, discutir y resolver errores en el sistema [1], pero que siempre es un problema debido diversos factores. Normalmente en todos los ámbitos de trabajos se utiliza algún gestor de proyecto para la mantención de los mismos. Estos gestores permiten a los desarrolladores, QA [2] y usuarios finales agregar reportes de bugs y nuevos features [3].

Al ocurrir un bug o un error, el usuario ingresa al sistema de gestión de proyecto, selecciona una categoría, prioridad, ingresa un título y una descripción y puede o no asignarlo a un desarrollador. El título y la descripción contendrá toda la información que el desarrollador utilizara a la hora de resolver el error, de no ser suficiente para identificar la razón del mismo comienza un intercambio de mensajes entre el desarrollador y el usuario/reportador en el que el primero intentara obtener datos del contexto en el que el error ocurrió y bajo qué condiciones para poder replicarlo.

Por experiencia propia de todos los proyectos en los que hemos trabajado y al hablar con colegas de como es el proceso de desarrollo en sus trabajos, en la mayoría de los casos por no decir todos, suele haber problemas con la

[1] Un bug es un error o un defecto en el software que hace que un programa funcione incorrectamente.

[2] En el ámbito del desarrollo de software, la sigla QA significa Quality Assurance, o aseguramiento de la calidad. Se trata de un conjunto de actividades de evaluación de las distintas etapas del proceso de desarrollo para garantizar que el producto final sea de calidad.

[3] Pequeña función orientada al cliente.

información brindada por los usuarios que reportan los errores cuando no pertenecen al ámbito del desarrollo de software. Esto puede ocurrir porque el usuario al desconocer la razón del error, no sepa qué información del contexto es relevante para el desarrollador.

Un reporte típicamente contiene una descripción detallada del error y ocasionalmente insinúa la ubicación. Sin embargo, los informes de errores varían en su calidad de contenido, generalmente proveen información inadecuada o incorrecta. Por lo tanto, los desarrolladores a veces tienen que enfrentar errores con descripciones como: "Sem Web" (APACHE bug COCOON-1254), "wqqwqw" (ECLIPSE bug #145133), o simplemente "GUI" con un comentario "The page is too clumsy" (MOZILLA bug 109242). No es de extrañar que los desarrolladores se retrasen por una mala redacción en los reportes de errores porque identificar el problema a partir de dichos informes lleva más tiempo. [2]

Es importante que la información que el usuario/reportador del sistema provea sea relevante y completa para ayudar con la rápida y correcta resolución de los bugs [3]. Sin embargo, suele suceder que esa información llega a los desarrolladores luego de varios intercambios de mensajes.

Como se explica en otros trabajos, los desarrolladores de software invierten una significativa cantidad de recursos en atender los informes/reportes de bugs y errores enviados por los usuarios [4]. Suele ocurrir que debido a la gran cantidad de reportes, no todos lleguen a ser atendidos debido a la demanda de tiempo que pueda llevar entenderlos y resolverlos.

En trabajos de investigación de la literatura del área se ha demostrado que entre la información que resulta de gran utilidad para los desarrolladores por parte de los que reportan los bugs se encuentran: versión del software, stack traces [4], pasos a llevar a cabo para reproducir el error, casos de prueba, comportamiento esperado y capturas de pantalla entre otros [5][6][7]. Los desarrolladores se suelen ver forzados a solicitar la información explícitamente y dependiendo de la calidad y el tiempo de la respuesta brindada puede llegar a impactar drásticamente en el tiempo de desarrollo de la solución del error/bug reportado.

Las herramientas de gestión de proyecto existentes como Trello, Basecamp, Freshdesk, Jira y otras brindan un sistema muy completo de gestión de proyectos. Permiten armar y organizar equipos de desarrollo y asignar tareas, deadlines [5] y además proveen un sistema de mensajería y alertas de

[4]Informe de los elementos activos en la pila de ejecución en un momento determinado durante la ejecución de un programa.

[5]Fecha limite de entrega o resolución, ya sea de una información o un producto.

eventos con prioridad.

Dentro de la creación de tareas, si bien se puede asignar, dar prioridad, especificar tiempos límites y permite escribir una descripción del mismo, estas herramientas no proveen ninguna facilidad a los usuarios de reportar bugs, son ellos los encargados de volcar a texto el problema que ha ocurrido. Por esto surge la necesidad de extender estos sistemas y brindarle una herramienta para los usuarios de desarrollos de aplicaciones web[6] que les permita reportar errores desde el sistema en el que trabajan en el momento en el que ocurre el error, recopilando automáticamente toda información que pueda ser útil para el desarrollador a la hora de resolver el problema.

1.2. Contribuciones

En esta tesina se analizaron los inconvenientes del proceso y las herramientas existentes para el reporte de errores, y se propone una herramienta que soluciona estos problemas. La herramienta permite a los usuarios encargados de realizar los reportes de bugs en aplicaciones web que lo realicen sin tener que cambiar de entorno e ingresar al gestor de proyecto para hacerlo, para lograr esto, la herramienta fue desarrollada en forma de extensión de navegador y será la misma la encargada de crear una tarea en el gestor de proyecto.

La característica fundamental de la extensión desarrollada es que enriquecerá el reporte utilizando información del entorno que recopila automáticamente, para proveer al desarrollador encargado de resolverlo de un contexto lleno de información útil para resolver problemas frecuentes y otros no tan frecuentes.

Para corroborar y medir la utilidad de la extensión realizamos un experimento en el cual dos grupos de desarrolladores identificarán errores en un proyecto web utilizando reportes realizados con y sin la extensión desarrollada con el fin de comparar los resultados.

1.3. Organización

En cuanto a la estructura del documento, está dividido en 6 grandes capítulos. En el primero encontramos la introducción. En el segundo capítulo se encuentra el marco teórico en el que se describe la importancia y los

[6]Herramienta que los usuarios pueden utilizar accediendo a un servidor web a través de internet mediante un navegador.

problemas que suelen haber en el proceso de reporte de errores. Luego contamos con el capítulo de desarrollo propuesto en donde se habla de la herramienta desarrollada, explicando su arquitectura y las tecnologías usadas. A continuación un capítulo en donde se explica cómo se realiza la instalación, configuración y uso de la extensión, seguido por el capítulo de evaluación en el cual se describe cómo será el proceso del experimento que se realizó, desde el proyecto utilizado como base, encuestas, conformación de grupos, métricas y resultados. Por último los capítulos de conclusión y trabajo a futuro.

Capítulo 2

Marco teórico y trabajos relacionados

2.1. Introducción

Una parte importante del desarrollo de software se dedica a buscar y resolver los defectos. Boehm y Basili afirman que el mantenimiento consume más del 70 % del coste total del ciclo de vida de un producto de software [8].

Si un reporte de error es efectivo, entonces las probabilidades de que sea arreglado son mayores. Por lo tanto el arreglo del error depende de cuan efectivo sea el reporte. Reportar un error no es más que una habilidad que se adquiere con la experiencia, en esta sección se mencionan errores comunes y buenas prácticas para generar un reporte de manera correcta.

> "El objetivo de escribir un reporte de error (bug report) es que estos errores sean arreglados" - By Cem Kaner

Si el usuario no reporta el error correctamente, probablemente el programador termine marcándolo como irreproducible.

2.2. Trabajos de investigación relacionados

En su trabajo "How long did it take to fix bugs?", Kim y Whitehead miden el tiempo para corregir un error en dos proyectos de software y afirman que el tiempo de corrección de errores es una medida útil de la calidad del software [9].

Con el fin de identificar que características consideran los usuarios y los desarrolladores que un reporte debe tener para ser considerado un buen reporte, en el trabajo "What Makes a Good Bug Report?"[2] se realizo una encuesta a mas de 400 personas, entre ellos desarrolladores y reportadores. El resultado demuestra que no coinciden las respuestas entre la información que los desarrolladores necesitan y la información que los reportadores proveen. La mayoría de los desarrolladores considera que son muy importantes los siguientes items:

- Pasos para reproducir el error.
- Stack traces.
- Casos de prueba.
- Comportamiento actual y esperado.

En cambio para los reportadores únicamente suelen proveer el comportamiento actual, el esperado y los pasos para reproducir el error. Solo unos pocos agregan stack traces, casos de prueba y otro tipo de información relevante.

Otro estudio enfocado en la información de los reportes es el de "The significance of bug report elements"[10], en donde se realizo otra encuesta a desarrolladores para conocer su preferencia a la hora de leer reportes. La mayor parte de estos remarca que es de gran importancia una óptima descripción, pasos para reproducir el error o casos de prueba, stack traces, versión del software y sugerencias de arreglo. En este trabajo, para evaluar que tan bien están reportados los errores en repositorios públicos, se analizó más de 250 repositorios de proyectos populares de GitHub. El análisis estadístico de los reportes analizados demostraron que toda esa información impacta significativamente en los tiempos de resolución, sin embargo el 70 % de los reportes no poseen esta información.

Un enfoque interesante es el descripto en el trabajo "Situational Awareness: Personalizing Issue Tracking Systems"[11] en donde se plantea la importancia y la necesidad de la personalización de los sistemas de reporte de errores en base a la necesidad de los desarrolladores dependiendo del

proyecto, que es algo que implementamos en nuestra herramienta permitiendo a los usuarios y desarrolladores de agregar preguntas personalizadas que el usuario debe responder antes de realizar el reporte y también mediante selectores DOM que permiten a los desarrolladores el obtener automáticamente información especifica del HTML sin necesidad de que el usuario la ingrese.

2.3. Proceso de reporte de errores

Cuando hablamos de reporte de errores necesitamos diferenciar dos conceptos básicos:

- **El proceso de trackeo de errores:** es la manera de recolectar, organizar, seguir y procesar los errores.
- **El proceso de reporte de errores:** es la manera de reportar, documentar y explicar los errores lo más rápido posible.

Existen dos grandes distinciones entre sistema de reporte de errores:

- Proyectos de software libre en los cuales son los mismos usuarios los encargados de reportar los errores encontrados, para esto se les provee una interfaz propia del proyecto como Mozilla Firefox [1] y Django [2] entre otros. El potencial de estos repositorios de bugs es que permiten identificar y resolver mas bugs logrando un producto de mejor calidad [12].
- Proyectos de desarrollo de software privados, se utilizan herramientas de gestión de proyecto para el reporte de errores y es el personal de la misma los que utilizan la herramienta.

En los primeros, las organizaciones suelen describir un estándar que debe ser seguido estrictamente a la hora de reportar un error y generalmente no suelen ser tomados en cuenta los reportes que no sean correctamente realizados. Por ejemplo Firefox exige que se informe una serie de datos como pasos para reproducirlo, sistema operativo, versión del navegador, etc. En los proyectos privados en cambio, son las mismas empresas/organizaciones las que definen pautas para el reporte de errores, pero en general solo se destina una sección para hacerlo y se pide un título, descripción y opcionalmente la urgencia del mismo.

Muchos de los sistemas de seguimiento de errores de software se integran frecuentemente con otras herramientas, como pueden ser correo electrónico,

[1] Navegador web libre y de código abierto desarrollado para Linux, Android, iOS, macOS y Microsoft Windows.

[2] Framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como MVC.

sistemas de control de versiones como GitHub y GitLab [3], y otras herramientas de gestión administrativa.

El proceso de reporte de errores utilizando herramientas de gestión de proyectos que suele seguir un usuario consta de:

1. Identificar el error.
2. Crear una nueva tarjeta/entrada que contenga un título y una descripción del error. Opcionalmente puede incluir capturas de pantalla o documentos.
3. Asignar a un desarrollador o moverlo a una lista destinada a los bugs y errores.
4. Contestar las inquietudes del desarrollador si es que aparecen, proveyendo más información o siendo más descriptivo.
5. Esperar a que el error sea resuelto y marcado para testear la solución.

[3] GitHub y GitLab son servicios web de control de versiones y desarrollo de software colaborativo basado en Git.

2.4. Herramientas de gestión de proyecto

2.4.1. Qué son y para que se usan

Las herramientas de gestión de proyectos son softwares usados para la planificación de proyectos, programación, asignación de recursos y gestión de cambios.

Permite que los gerentes de proyecto (PM), stakeholders [4] y los usuarios controlen los costos y administren los presupuestos, la gestión de la calidad y la documentación, y también pueden usarse como un sistema de administración. Las herramientas de gestión de proyecto también son utilizadas para la colaboración y comunicación entre los stakeholders del proyecto.

Si bien las herramientas de gestión de proyectos son utilizadas de diversas maneras y con distintos fines, gran parte del tiempo se utiliza tanto para realizar reportes de errores así como también su posterior control y seguimiento.

2.4.2. Funciones principales

Algunas de las funciones principales de las herramientas de gestión de proyecto son:

- **Planificación de proyecto:** para definir un cronograma del proyecto, un PM puede usar el software para mapear las tareas del proyecto y describir visualmente las interacciones.
- **Gestión de tareas:** permite la creación y asignación de tareas, plazos e informes de estado.
- **Compartir documentos y archivos:** trabajo colaborativo sobre archivos compartidos.
- **Calendario y contactos:** líneas de tiempo que incluyen reuniones programadas, fechas de actividades y contactos.
- **Administración de Bugs/Errores:** Una herramienta de gestión de proyecto facilita el reporte de bugs/errores, visualización, notificación y actualización para las partes interesadas.

[4] Involucrados, parte interesada o interesados. Hace referencia a una persona, organización o empresa que tiene interés en una empresa u organización dada.

- **Rastreo de tiempo:** El software debe tener la habilidad de rastrear y almacenar el tiempo de todas las tareas para que puedan ser consultadas por terceros.

2.4.3. Tipos

- **Lightweight (Trello):** Diseñado para proyectos cortos o temporarios, o para equipos con procesos livianos.
- **Midweight (Basecamp, Asana, Wrike):** Herramientas más elaboradas, se utiliza cuando se necesita balancear procesos más complicados y facilidad de uso.
- **Heavyweight (Jira, MS Project):** Herramientas utilizadas cuando se tienen procesos realmente elaborados y se necesita personalizar muchos componentes para que se ajusten.

A continuación se muestra una tabla comparativa de las características y funcionalidades claves que uno debe tener en cuenta cuando elige una herramienta de gestión de proyectos. El color verde indica si la característica/funcionalidad viene incorporada en la versión estándar del software. Amarillo significa que dicha característica esta disponible a través de librerías de terceros o extensiones. Rojo quiere decir que la característica no está disponible de ningún modo.

	Kanban Board	Scrum	Gantt Chart	Time Tacking	Calendar View	Mobile Apps	Desktop Apps	API	Free Trial
Trello	🟢	🔴	🟡	🟡	🟢	🟢	🟢	🟢	🟢
Basecamp	🟡	🟡	🟡	🟡	🟢	🟢	🟢	🟢	🟢
Asana	🟢	🟡	🟢	🟡	🟢	🟢	🔴	🟢	🟢
Wrike	🟢	🔴	🟢	🟢	🟢	🟢	🟢	🟢	🟢
JIRA	🟢	🟢	🟡	🟡	🟡	🟢	🟡	🟢	🟢
MS Project	🟢	🟢	🟢	🟡	🟢	🟡	🟢	🟢	🟢

Figura 2.1: Tabla comparativa de caracteristicas en principales softwares de gestion de proyectos.

2.4.4. Trello

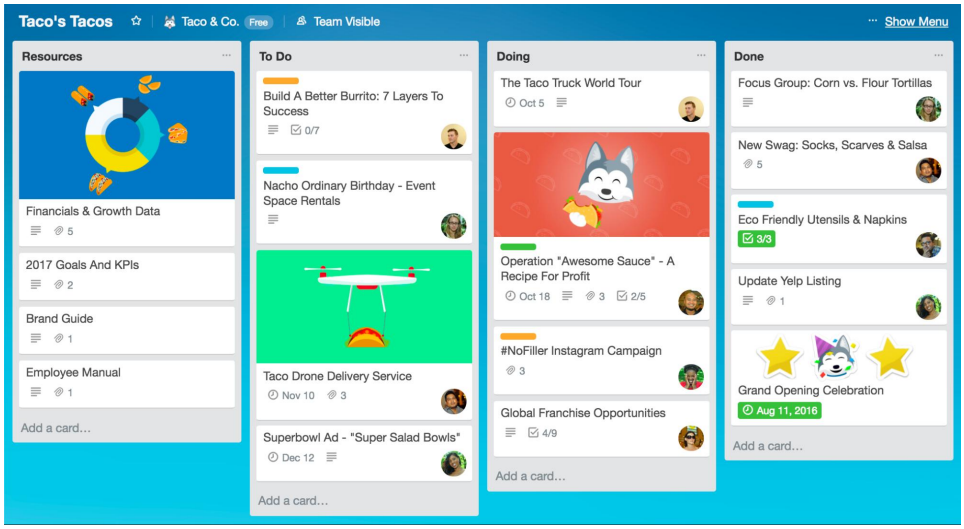

Figura 2.2: Ejemplo de pizarra de Trello con listas más comunes y tareas.

Trello es una de las herramientas de gestión de proyectos livianas más populares del mercado. Es mayormente utilizado por startups, equipos pequeños, proyectos temporales y en varios otros escenarios en los que el líder de proyecto necesita una estructura mínima de organización.

También es una buena herramienta para utilizar cuando se introduce un equipo a un ambiente de metodologías ágiles del estilo Kanban. A grandes rasgos la interfaz contiene un board (pizarra), columnas y cards (tarjetas). Sin embargo, las limitaciones de Trello aparecen si se desea tener un proceso de equipo más elaborado o si desea que toda la organización trabaje con la misma herramienta. Trello permite realizar reportes de errores sencillos con título, descripción, archivos adjuntos, usuarios asignados y etiquetas de colores para utilizar como referencias de urgencia o simplemente cosméticas.

Ventajas

- **Muy liviano:** se puede configurar en pocos minutos y es fácil de mantener.
- **Interfaz intuitiva:** fácil de comprender e interactuar.
- **Precio:** la versión gratuita es suficientemente buena para la mayoría de los proyectos simples y procesos livianos.

Desventajas:

- **Muy liviano:** la otra cara de ser liviano es que no permite una variedad amplia de customizaciones ni un flujo elaborado.
- **La información expira rápidamente:** al ser la actividad de log [5] y su capacidad de búsqueda muy limitada hace que sea difícil rastrear información en periodos largos de tiempo.

2.4.5. Basecamp

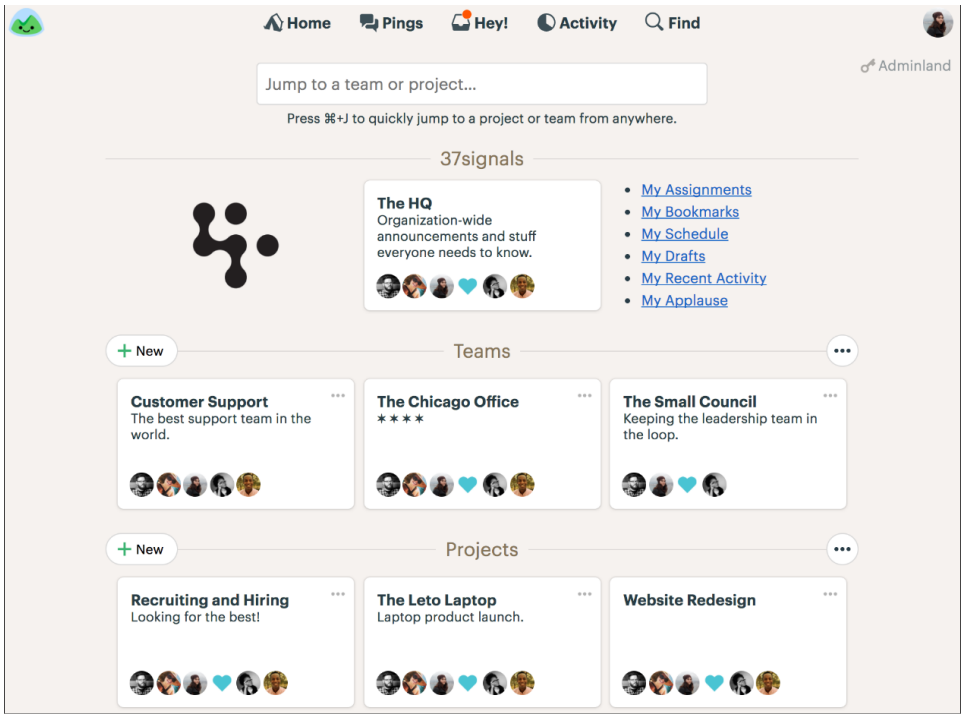

Figura 2.3: Ejemplo de listado de proyectos, equipos y actividades recientes en Basecamp.

Esta herramienta de gestión de proyectos ha experimentado una serie de crecimientos a lo largo de los últimos 15 años y parece ser que ha tenido la mayor aceptación con las agencias de desarrollos digitales/web. Ha

[5]En informática, se usa el término registro, log o historial de log para referirse a la grabación secuencial en un archivo o en una base de datos de todos los acontecimientos que afectan a un proceso particular.

logrado esto gracias a la combinación de tener una interfaz liviana y funcionalidades poderosas como un chat incorporado, poder compartir archivos, y la habilidad de poder comunicarse con clientes dentro de Basecamp. Sin embargo, Basecamp escasea de boards (pizarras), estimaciones, y otros atributos necesarios para el desarrollo ágil. Si bien hay aplicaciones de terceros, que pueden ayudarlo a lograr una configuración ágil, otras herramientas de gestión de proyectos nombradas anteriormente tienen estas características listas para usar. Basecamp permite crear TO-DO's (tareas) con título, descripción, usuario asignado, fecha límite, archivos adjuntos y la posibilidad de crear tantas listas como sean necesarias para mover las tareas dentro de estas para indicar su estado/prioridad.

Ventajas:

- **Precio:** un precio fijo por mes que no aumenta si se agregan nuevos proyectos.
- **Liviano:** Basecamp es de fácil usabilidad pero con poderosas características.

Desventajas:

- **Scrum/kanban boards:** Boards (pizarras) están disponibles a través de extensiones, lo que hace que Basecamp sea menos competitiva con otras herramientas midweight.
- **Sprints:** El seguimiento para Sprint [6] es soportado solo a través extensiones (librerías de terceros).
- **Chat:** Aunque se promociona a sí mismo como un reemplazo de Slack [7], la interfaz de usuario en cuanto a la comunicación no está a la par con las herramientas de chat en tiempo real dedicadas.

[6] Sprint es el nombre que va a recibir cada uno de los ciclos o iteraciones que vamos a tener dentro de un proyecto Scrum.

[7] Slack es una herramienta de comunicación en equipo.

2.4.6. Jira

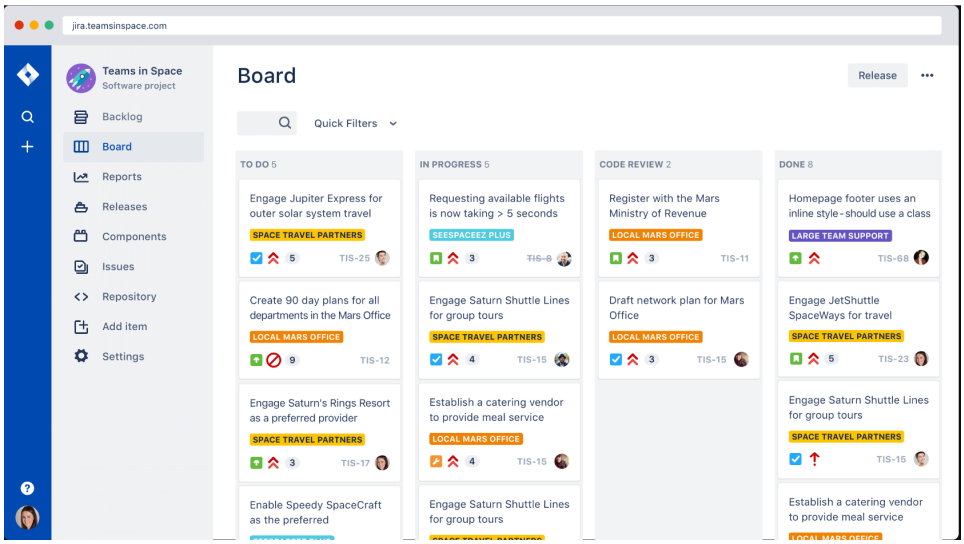

Figura 2.4: Ejemplo de pizarra en Jira con listas y tareas asignadas a usuarios.

Jira es una de las herramientas de gestión de proyecto más populares del mercado y es una opción segura para las organizaciones. Su potencial se puede notar más en empresas de gran tamaño que necesiten coordinar entre múltiples procesos complejos y tienen la necesidad de personalizar el flujo de tareas. Jira, como el resto de los gestores nombrados, brinda la posibilidad de crear reportes dentro de proyectos o subproyectos con: titulo, descripción, archivos adjuntos, usuarios asignados y fecha límite, pero además brinda la posibilidad de especificar la prioridad o urgencia y peso de la tarea (dificultad u horas estimadas que se requieren para resolver la tarea) entre otras de sus principales características.

Ventajas:

- **Precio:** la versión gratuita cubre todas las necesidades básicas para la gestión de proyectos chicos.
- **Altamente personalizable:** Las herramientas de flujo de trabajo le permiten adaptar Jira a la forma en que opera su empresa.
- **Integraciones múltiples:** provee múltiples integraciones con otras aplicaciones populares tales como GitHub, Salesforce, Outlook, etc.

- **Scrum y sprints:** El flujo de trabajo de Scrum con el seguimiento Sprint está incluido en la herramienta sin necesidad de agregar extensiones.

Desventajas:

- **Tiempo de configuración largo:** El tener tantas funciones para personalizar quiere decir que le lleva más tiempo a los usuarios configurar su entorno y enseñar a los nuevos a como usar la herramienta.
- **Interfaz compleja:** El manejo de roles, boards (pizarras), y issues [8] no es tan intuitivo como podría serlo.

[8]El término issue se atribuye a la unidad de trabajo para realizar una mejora en un Sistema informático. Un issue puede ser el arreglo de un fallo, una característica pedida,

2.5. Problemas frecuentes

2.5.1. Introducción

Cualquiera que haya escrito software de uso público probablemente haya recibido al menos un mal reporte de errores. Reportes que dicen algo tan vago como: "Esto no funciona!", reportes que no tienen sentido, que no dan suficiente información para identificar el problema o incluso reportes que brindan información equivocada, reportes de problemas que suelen terminar siendo errores de los usuarios y no del software en sí o que suelen terminar siendo fallas de conexión.

2.5.2. Falta de información básica

Usualmente los usuarios/testers del sistema encargados de reportar los errores que ocurren en el mismo no son desarrolladores y suelen omitir información básica que suele resultar de utilidad para la rápida identificación del mismo. Ejemplos:

- **URL donde ocurre el mismo:** tener la URL [9] donde ocurre el error ayuda a despejar rápidamente cualquier duda de en qué sección del software ocurre el error reportado. Por más que la descripción del mismo sea ambigua, poseer la URL ayuda a la rápida identificación del mismo tanto en el sitio web como en el código del mismo, pudiendo acceder a los endpoints [10] involucrados casi de manera instantánea. También es de gran utilidad para obtener los datos enviados en una petición de tipo GET, esto nos permite por ejemplo si se trata de un request [11] para la eliminación de un usuario, saber a través de la URL el id del mismo y poder así poder reproducirlo bajo las mismas condiciones.

- **Pasos para reproducir el error:** aportar los pasos para reproducir un error ayuda al desarrollador a entrar en contexto y aporta información previa al error que puede ser crítica para replicar el mismo y que quizás sea la fuente de este.

[9] URL significa Uniform Resource Locator (Localizador de Recursos Uniforme). Una URL no es más que una dirección que es dada a un recurso único en la Web.

[10] Es un tipo de nodo de red de comunicación, una interfaz expuesta por un comunicante o un canal de comunicación.

[11] Permite el acceso a toda la información que pasa desde el navegador del cliente al servidor.

- **Información ingresada en formularios:** generalmente cuando el error aparece al realizar la carga de un formulario es de gran ayuda poseer toda la información ingresada en sus campos, esto permite encontrar rápidamente problemas comunes tales como errores de tipo o campos obligatorios.

- **Capturas de pantalla:** ayudan a identificar rápidamente el lugar donde ocurre el error y a la hora de explicar errores de estilos y/o formatos como posición de botones, colores, fuentes, etc.

- **Ambiente en el cual aparece el error:** un proyecto usualmente cuenta con al menos 3 ambientes para los usuarios: testing, producción y staging. Es importante que se destaque en cuál de los ambientes ocurre el error ya que la urgencia de este no es la misma si ocurre en ambiente de testing que en producción. Por otro lado los errores que ocurren en el ambiente de testing puede que sean producto del desarrollo de nuevas features.

- **Explicar el comportamiento esperado:** Es un error asumir que el desarrollador sabe cual es el correcto funcionamiento, explicar el comportamiento esperado de una funcionalidad al producirse un error siempre es de ayuda para comprender el porqué del mismo. También es de gran utilidad para introducir en tema a los nuevos desarrolladores en un proyecto.

2.5.3. Falta de información específica para el desarrollador

- **Información almacenada en la caché del navegador:** la información almacenada en las bases de datos de los navegadores tales como LocalStorage, SessionStorage, IndexedDB pueden ser de ayuda para resolver problemas en particular y no son de común conocimiento para los usuarios.

- **Assets cargados:** cuando se trata de aplicaciones web en ocasiones pueden producirse errores por incompatibilidades de versiones de assets [12] o permisos para cargar los mismos.

- **Información específica de cada software en particular:** dependiendo del software puede que haya información que sea útil disponer

[12] Los assets web son las hojas de estilo CSS, los archivos JavaScript y las imágenes que se utilizan en el frontend de las aplicaciones para que tengan un buen aspecto.

de ella solo en ese desarrollo y no en otros, como por ejemplo si el sistema tiene diferentes comportamientos dependiendo de la antigüedad del usuario, puede ser de utilidad el poseer el usuario en cuestión.

2.5.4. Mala descripción del error

Una mala descripción del error siempre entorpece la identificación y resolución del mismo, incrementando el esfuerzo y tiempo del desarrollador [13]. Si la descripción es muy larga puede marear al desarrollador y conducir a realizar cambios en una funcionalidad que no sean los esperados, si es muy corta puede no ser suficiente para la identificación o reproducción del mismo. Lo ideal sería que en breves palabras, el usuario explique los pasos para reproducir el error y cual es el comportamiento esperado.

2.5.5. Otras problemáticas comunes

- **No asignar prioridad o urgencia:** cuando no se asignan prioridades, el esfuerzo de los desarrolladores puede estar concentrado en bugs de poca importancia en lugar de enfocado a los errores críticos.
- **Priorizar todos los errores como críticos:** creer que asignar todos los errores como críticos se van a resolver más rápido, al igual que en el punto anterior solo se conseguirá que el esfuerzo de los desarrolladores no esté bien administrado.
- **No asignar desarrollador de ser necesario:** en proyectos grandes y complejos es de gran ayuda que si el usuario reportador sabe que desarrollador trabajo en una determinada funcionalidad, se asigne el bug a este que tendrá conocimientos del comportamiento de la misma y le sera mas fácil identificar los errores que a otros desarrolladores sin los conocimientos de los requerimientos o particularidades de la funcionalidad en cuestión.
- **Reabrir continuamente el mismo reporte:** los usuarios tienden a reabrir reportes viejos cuando encuentran bugs similares o a extender reportes a medida que descubren el mal funcionamiento de una aplicación lo que conlleva a que el reporte no sea específico, lleve mucho tiempo su resolución y dificulta el seguimiento del mismo.
- **Combinar múltiples bugs en el mismo reporte:** esto puede resultar de utilidad para el usuario pero entorpece la asignación de recursos ya que por más que parezcan relacionados, puede uno de ellos ser un

error crítico/urgente y el otro no, lo que puede retrasar el arreglo del mismo en producción.

2.6. Buenas prácticas en el proceso de reporte de errores

2.6.1. Introducción

El reporte de errores es un aspecto importante en la vida de un software. Un buen reporte ayuda a la comunicación del equipo de desarrollo con los usuarios del sistema y evita confusiones y malentendidos, si el reporte es efectivo hay mas probabilidades de que sea resuelto [14]. Cualquiera puede escribir un reporte de error, pero no cualquiera puede escribir un buen reporte de error. Uno debería poder distinguir entre un reporte común y uno bueno. Un buen reporte posee ciertas características y se realiza siguiendo las siguientes técnicas.

2.6.2. Características y técnicas

1. **Ser específico:** no sobredimensionar y complejizar un problema, tratar de ser específico y acotar el problema al contexto en el que ocurre. No mezclar distintos errores por más que parezcan similares, de ser necesario es mejor siempre escribir un reporte para cada error.

2. **Reproducible:** Un error que no es reproducible muy difícilmente puede llegar a ser resuelto. Se deben mencionar todos los pasos involucrados para reproducir el error, evitando asumir como implícito u obviar alguno. Un error descripto paso a paso es fácil de reproducir y arreglar.

3. **Tener un número de error específico:** Siempre asignar un número único para identificar cada error y poder realizar un seguimiento. Si se utiliza una herramienta de gestión de proyectos, estas generan automáticamente un número de identificación único para cada reporte en su creación.

2.6.3. Cualidades de un buen reporte de errores

Un buen reporte debe ser claro y conciso sin obviar los puntos clave. La falta de claridad al reportar un error puede producir malentendidos y retrasar el proceso de desarrollo. El reporte de errores es una de las áreas más descuidadas durante el ciclo de vida de un software .

- No asumir que el error es por culpa del desarrollador y no usar palabras fuertes o que suenen agresivas.

- Corroborar que el error no haya sido reportado anteriormente, revisar la lista de todos los bugs antes de comenzar a escribir uno nuevo y si la lista llegara a ser muy extensa, agrupar los reportes por funcionalidad o características del proyecto. Publicar un reporte de error duplicado no solo es una pérdida de tiempo para el usuario, sino que también puede ocurrir que dos desarrolladores en algún momento dado, estén enfocados en resolver en el mismo error sin ser consciente de ello. Estudios anteriores han demostrado que hasta el 36 % de los informes de errores eran duplicados o no eran válidos [15][16][17].

- La descripción de un reporte debe contener las respuestas a dos preguntas básicas, "¿Cómo ocurrió?" y "¿Dónde ocurrió?". El reporte debería responder con claridad cómo se produjo el error, y dónde fue producido exactamente. Brindar toda la información relevante que el desarrollador necesita para encontrar y reproducir el error con facilidad.

- Un reporte puede ser pospuesto y tomarse para resolver en un tiempo lejano y si no se es claro y no se brinda toda la información necesaria para identificarlo, puede ser que el usuario ya no se acuerde donde o como ocurrió el error y no pueda ser resuelto.

- Generar un reporte por cada error y evitar agrupar errores relacionados en un mismo reporte. El mayor problema de poseer múltiples errores en un mismo reporte, es que uno no puede ser cerrado hasta que se resuelvan todos los demás.

2.6.4. Contenido de un buen reporte de errores

Número único de identificación

Ayuda al desarrollador a mantener un correcto seguimiento del error para saber si fue resuelto o no.

Título

El título es la parte del reporte que más se lee. Debería describir de manera concisa de qué trata el error.

Prioridad

Basados en la urgencia de que un bug o error sea resuelto podemos aplicarles prioridades. Un error puede ser:

- **Crítico:** un error que afecta a las reglas o flujo de negocio, que produzca un fallo en el sistema o pérdida de información.
- **Mayor:** un error importante que debe ser arreglado lo antes posible si no existen críticos.
- **Menor:** un error importante pero que puede esperar a ser resuelto como errores de falta de ortografía en textos.
- **Trivial:** errores que pueden esperar a que todos los críticos, mayores y menores sean resueltos, como mejoras en la interfaz de usuarios.

Los errores deben ser resueltos siguiendo el orden de prioridades, los críticos deben ser atendidos inmediatamente y luego continuar con los menores. Esto permite que el esfuerzo de los desarrolladores esté bien administrado.

Estado

Estado en el que se encuentra actualmente el error. Por defecto suele ser "nuevo" o "creado". En general, el ciclo de resolución de un error pasa por los siguientes estados:

- Nuevo
- Asignado
- En proceso
- Listo para probar
- Aprobado
- Cerrado

Descripción

La descripción es la parte más importante del reporte ya que brinda toda la información necesaria para que el desarrollador comprenda tanto el motivo del error como los pasos para reproducirlo. Una descripción pobre puede crear confusión y pérdida de tiempo. Para que un reporte sea completo debe contener al menos la siguiente información:

- **Reportador:** el nombre y contacto del usuario que genera el reporte. Las herramientas de gestión de proyecto ya se encargan de este punto y no es necesario que el usuario complete esta información a la hora de crear cada reporte.
- **Pasos para reproducirlo:** deben mencionarse de manera clara todos los pasos necesarios y en correcto orden para reproducir el error.
- **Comportamiento actual y esperado:** el comportamiento actual es necesario para identificar donde ocurre el error y por qué, y el comportamiento esperado ayuda al desarrollador a entender que hacer para arreglarlo.
- **Producto/funcionalidad:** en proyecto grandes que involucran distintas y complejas funcionalidades y un gran número de desarrolladores, es bueno que los reportes sean agrupados en distintas secciones para facilitar la asignación de recursos.
- **Versión del software:** versión del software si posee.
- **Plataforma/Entorno:** Las aplicaciones pueden comportarse de manera diferente dependiendo del entorno en el que corren. Esto conlleva que el desarrollador pueda no ser capaz de replicar ciertos errores si no utiliza el mismo entorno que el usuario que reporta el error. Por ello es importante siempre nombrar cual es el sistema operativo y/o navegador que utilizo.
- **URL:** la URL de la página donde ocurre el error.
- **Información de los formularios:** datos ingresados en los campos de los formularios.

Usuario asignado

Por motivos de control y asignación de recursos siempre es bueno saber que desarrollador está trabajando en cual error. También si se conoce quién es el encargado del módulo donde ocurre el error se puede asignar a este, sino se deja en blanco y el encargado del proyecto se hará cargo de la asignación.

Captura de pantalla y archivos adjuntos

Una imagen vale más que mil palabras. Realizar capturas de pantalla en el momento que ocurre el error puede ayudar al desarrollador a identificar la sección donde ocurrió por más que no conozca el nombre de la misma.

2.7. Herramientas relacionadas

2.7.1. Introducción

Actualmente existen muchas herramientas desarrolladas para realizar reportes y seguimiento de errores, tanto para pequeños y grandes proyectos. Estas herramientas fueron desarrolladas por comunidades de código abierto algunas, y otras por empresas privadas. Al haber tanta variedad se hace difícil elegir una herramienta en particular que nos provea una eficaz, recomendable, útil y rentable herramienta para el monitoreo y reporte de un error [18].

2.7.2. Trello Extension

Trello provee una extensión del navegador para facilitar y mejorar la manera de realizar reportes de bugs y errores de tipo visual, errores de tipeo, y otros en sistemas web.

Permite realizar capturas de pantalla y realizar ediciones sobre dichas capturas, permitiendo dibujar flechas, recuadros y escribir texto sobre ellas entre otras funciones.

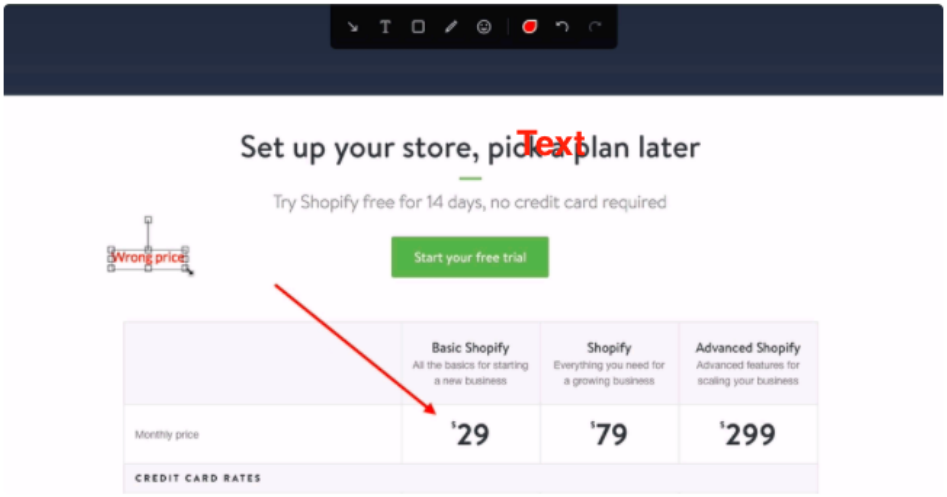

Figura 2.5: Ejemplo de de captura de pantalla con anotaciones utilizando Trello Extension.

La extensión además anexa en la descripción del reporte/tarea la infor-

mación del navegador donde fue utilizada para reportar como así también la URL:

- Nombre y versión del navegador
- Resolución del navegador
- Resolución del dispositivo
- Nivel de zoom
- Pixel ratio
- Nombre y versión del sistema operativo
- Agente de usuario

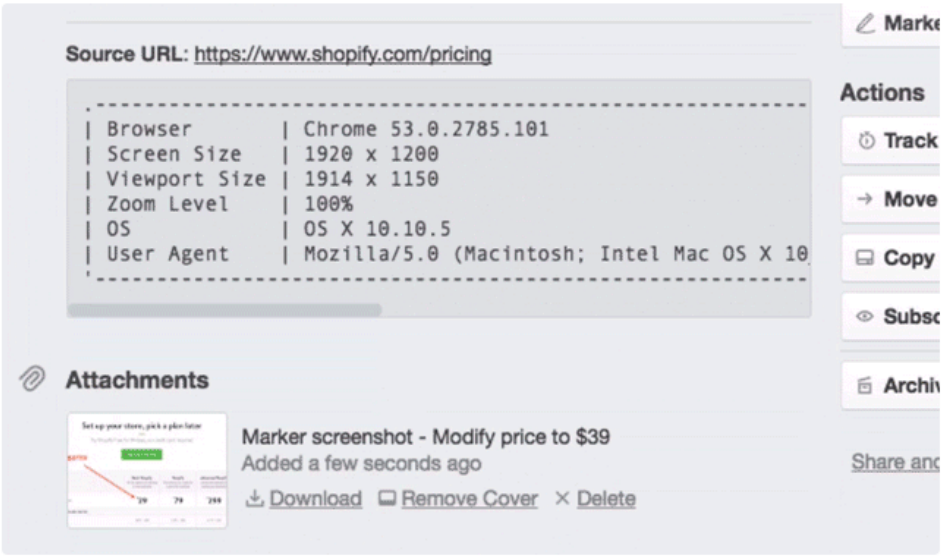

Figura 2.6: Ejemplo de tarea creada con Trello Extension con información recopilada automáticamente.

Además permite desde la extensión la posibilidad de seleccionar el proyecto en el cual se quiere crear la card, como así también la lista de tareas, ingresar un título y una descripción para la tarea/reporte, prioridad/importancia (trivial, menor, mayor o crítico), integrantes o usuarios asignados y fecha de inicio y fecha límite para resolver el error.

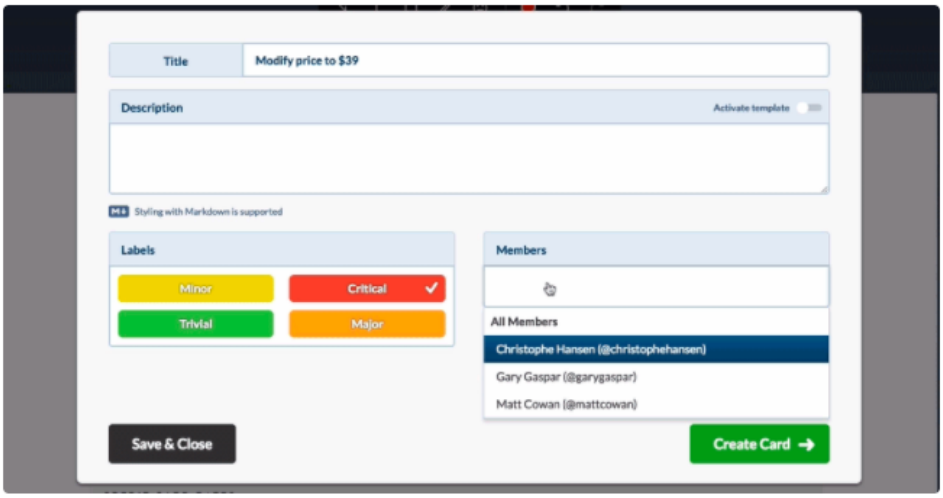

Figura 2.7: Formulario de creación de tarea/reporte con Trello Extension con asignación de usuario y prioridad.

2.7.3. BugDigger

BugDigger es una herramienta ágil que ayuda a crear reportes de error útiles y consistentes. Esta extensión captura información contextual como:

- Versión del navegador
- Sistema Operativo
- URL donde se reportó
- IP del usuario
- Agente de usuario
- Requests (POST, GET data) con fecha y hora
- Historial reciente de navegación

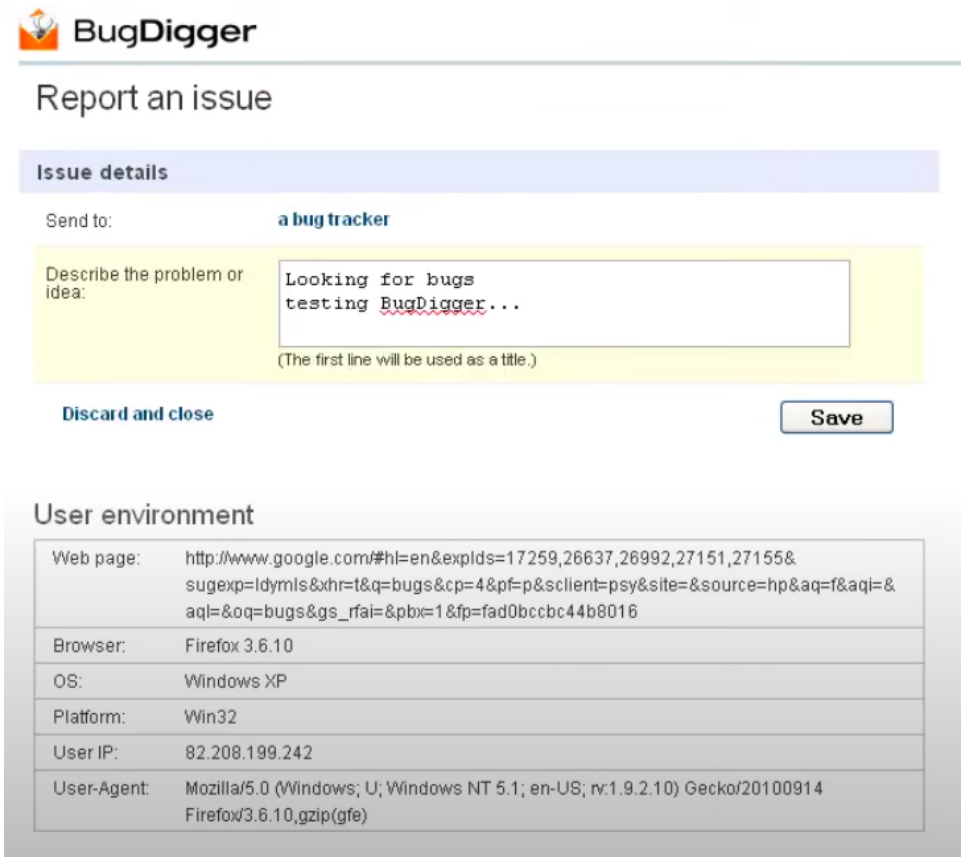

Figura 2.8: Ejemplo de formulario de reporte de errores utilizando la extensión de navegador BuggDigger.

También permite realizar capturas de pantalla y realizar anotaciones sobre las mismas.

La información capturada se guarda en el servicio de "My BugDiggerz puede ser enviado directamente hacia un gestor de proyecto externo como Jira, Pivotal Tracker, Trello, Basecamp, etc. Cuando se utiliza dicha funcionalidad esta extensión provee un formulario personalizado dependiendo del gestor de proyecto seleccionado para ingresar información más detallada, como por ejemplo asignar a un usuario determinado, prioridad, etc.

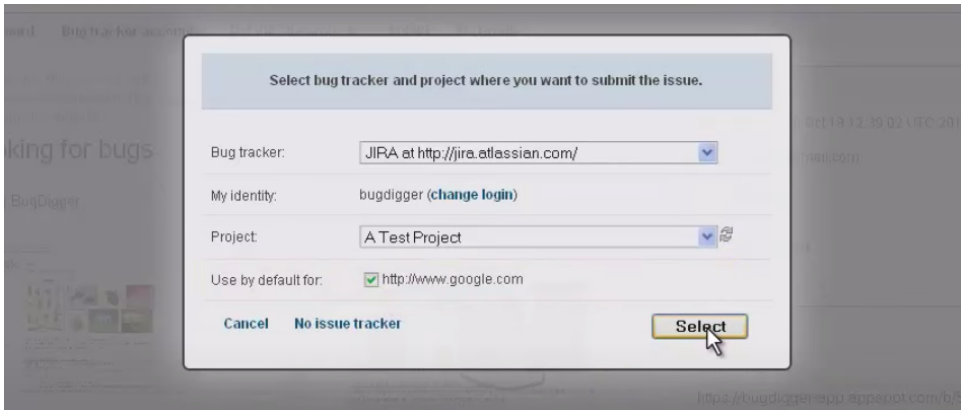

Figura 2.9: Ejemplo de selección de herramienta de gestió de proyectos con BugDigger.

2.7.4. BugMeBack

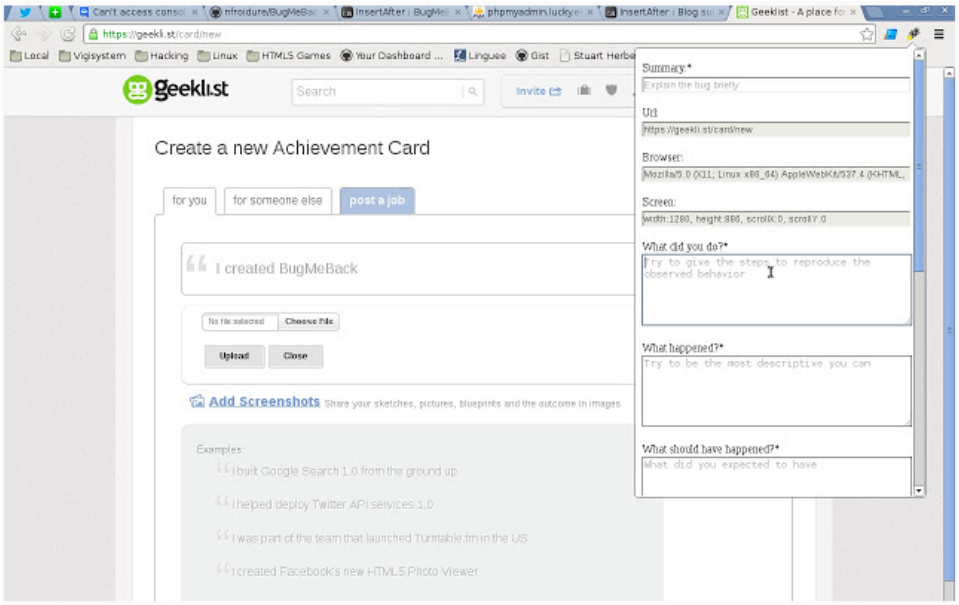

Figura 2.10: Ejemplo de creación de reporte de errores con la extensión de navegador BugMeBack.

BugMeBack es una extensión de Google Chrome [13] bajo la licencia GNU/GPL que permite a los usuarios realizar reportes para los cuales estos tienen que ingresar obligatoriamente cierta información, así como también automáticamente se adjunta información del contexto de la navegación:

- Pasos para reproducir y entender el error.
- Tomar capturas de pantallas
- Agente de usuario
- Tamaño de pantalla
- URL del error
- Logs de la consola del navegador (console.log, errores de JavaScript, etc)

Para poder enviar estos reportes, el usuario puede copiar una URL generada por la extensión para compartir o copiar el contenido del reporte y enviarlo por email o creando una tarea en el gestor de proyecto.

2.7.5. Usersnap

Usersnap es una herramienta que permite realizar reportes de errores en cualquier aplicación web. Cuenta con su propio gestor de proyecto pero también brinda la posibilidad de integrarse con más de 50 gestores de proyecto como Jira, Slack, Zapier, etc. Facilita realizar test de aceptación de usuario cuando hay equipos remotos, y su comunicación en los diseños y feedback de los desarrollos.

Al igual que la mayoría permite realizar capturas de pantalla y realizar anotaciones sobre las mismas y provee de un modesto formulario para la creación del reporte el cual cuenta con una descripción, un email y usuario al cual se le asigna el reporte.

[13] Es un navegador web de código cerrado desarrollado por Google.

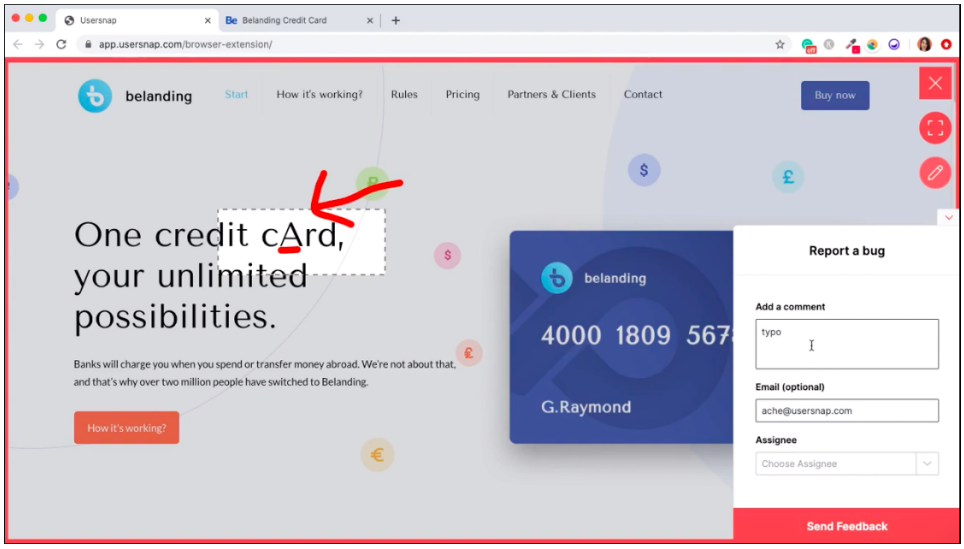

Figura 2.11: Ejemplo de captura de pantalla con anotaciones y creación de reporte de errores con UserSnap.

Entre sus principales funcionalidades se encuentra:

- Capturas de pantallas
- Asignación de usuarios
- Asignación de etiquetas y archivos adjuntos a los reportes
- Errores de consola del navegador
- Información meta de la navegación: URL, agente de usuario, tamaño de pantalla, ubicación entre otras.

2.7.6. Comparación

Al realizar una comparación entre las herramientas mencionadas anteriormente y la nuestra, podemos notar algunas similitudes en la información que anexan al reporte, como por ejemplo: URL donde se genero el error, posibilidad de tomar capturas de pantalla, historial de navegación, etc. De las herramientas vistas podemos destacar cierta información que brindan a diferencia de la nuestra:

- Posibilidad de realizar anotaciones sobre las capturas de pantalla.

- Asignación de usuario, prioridad y fecha limite.
- Versión, nombre y resolución del navegador.
- Logs de la consola del navegador.

Si bien las herramientas relacionadas cumplen con la información mínima requerida para considerarse un óptimo reporte[19], entre las características que nuestra herramienta brinda y el resto no destacamos:

- Permite conectarse con múltiples herramientas de gestión de proyecto para realizar los reportes (solo BugDigger lo hace también).
- Información guardada en LocalStorage y SessionStorage.
- Posibilidad de agregar selectores DOM para la obtención de información.
- Posibilidad de agregar preguntas personalizadas que el reportador debe contestar antes de realizar el reporte.
- Peticiones al servidor con su data ingresada en los formularios (solo BugDigger lo hace también).
- GIF animado con las acciones realizadas durante la navegación hasta el reporte.

Capítulo 3

Desarrollo Propuesto

3.1. Introducción

Comprendida la problemática existente de comunicación entre los desarrolladores y reportadores, nuestra solución consiste en el desarrollo de una extensión de navegador que recopila información de contexto de la navegación del usuario reportador.

La extensión permite al reportador iniciar sesión a su gestor de proyecto y poder crear los reportes en donde será anexada la información recopilada.

Información recopilada:

- **Current URL:** URL desde la que se realizó el reporte.
- **Custom DOM Info:** contenido de los elementos DOM [1] configurados en las opciones de la extensión.
- **History:** listado de las últimas 10 URLs por las que pasó.
- **Requests:** Peticiones al servidor (sincrónicas y sincrónicas) con su información.
- **Local Storage:** contenido del Local Storage.
- **Session Storage:** contenido del Session Storage.

[1] Document Object Model o DOM es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Como así también, se le brindara la posibilidad de realizar capturas de pantalla y se construirá un GIF [2] animado a partir de los clicks realizados por el usuario hasta el momento de realizar el reporte.

[2]Graphics Interchange Format, también conocido como GIF, es un formato gráfico digital utilizado ampliamente en la Web, tanto para imágenes como para animaciones.

3.2. Arquitectura

Para poder comprender la arquitectura de nuestro proyecto es necesario conocer primero cómo es la arquitectura de una extensión de navegador.

Si bien la arquitectura puede variar de extensión a extensión dependiendo de su robustez, en general la mayoría suele incluir los siguientes componentes:

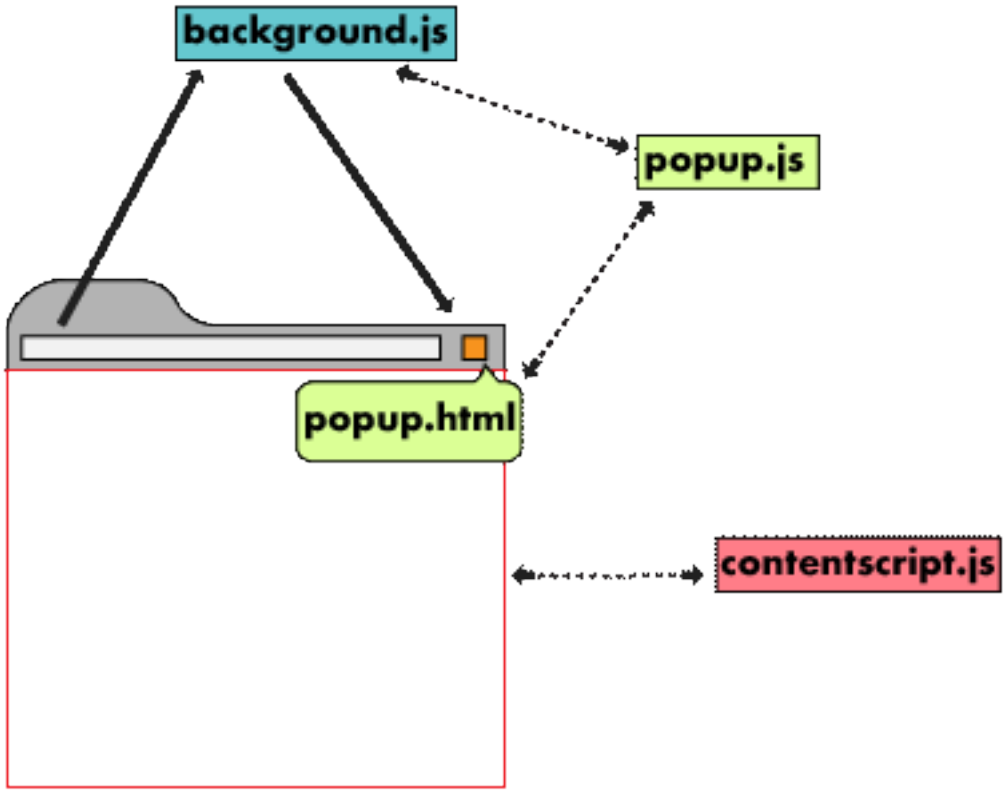

Figura 3.1: Arquitectura de una extension de navegador web.

- **Manifest:** Es un archivo en formato JSON que contiene información de la extensión como nombre y versión, así como también aspectos funcionales como scripts [3] de contenido, permisos de acciones de navegador y scripts en segundo plano entre otros.

[3]En informática, un script, secuencia de comandos es un término informal que se usa para designar a un programa relativamente simple.

- **Background Script:** Es el manejador de eventos de la extensión, está siempre alerta a los eventos "disparados" por el navegador. Se mantiene a la espera de los eventos para poder realizar la lógica necesaria.
- **UI Elements:** La mayoría de las extensiones suelen limitarse a un popup para brindar la interfaz de usuario, pero puede ser insertado HTML para extender dicha interacción.
- **Content Script:** Contiene el código JavaScript que se inserta en el contexto de la página donde el plugin está activo. Content scripts pueden leer y modificar el DOM de las páginas que el navegador visita.
- **Options Page:** permite la customización de la extensión al habilitar o deshabilitar funcionalidades.

Ya que el Content Script corre en el contexto de la página y no de la extensión, para poder comunicarse con el resto de la extensión (página de opciones, background script, etc.) puede hacerlo de dos maneras, mediante la API [4] de mensajes y/o la API de almacenamiento.

[4] API significa interfaz de programación de aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

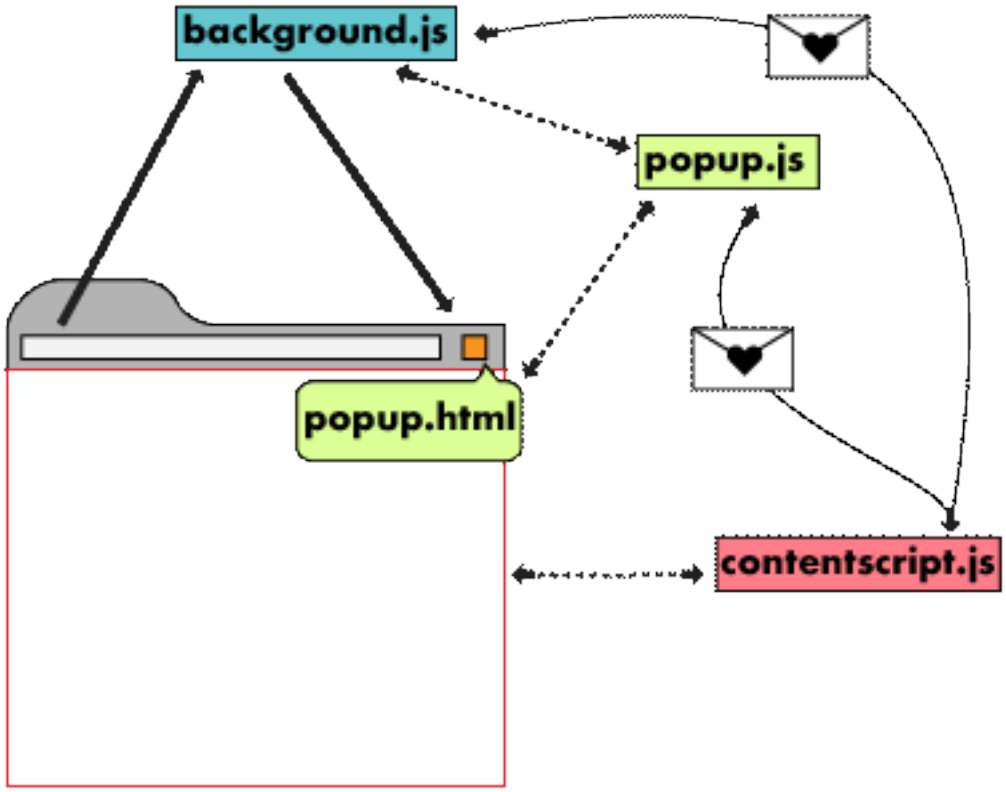

Figura 3.2: Demostración de comunicación entre los componentes de la extensión.

La arquitectura de nuestro proyecto puede ser graficada de la siguiente manera:

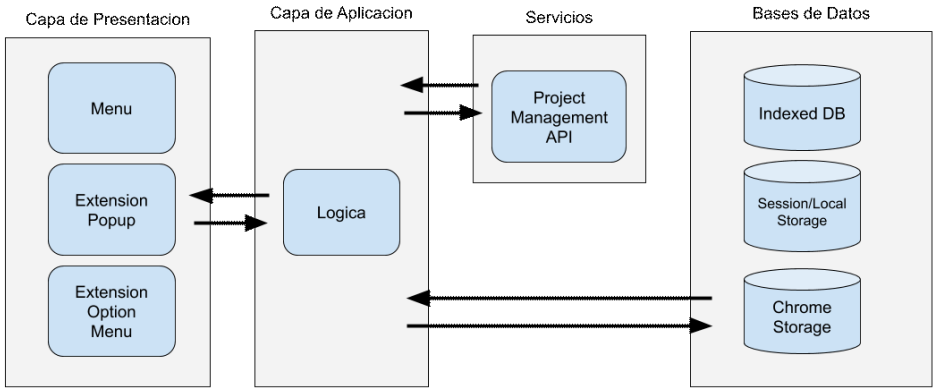

Figura 3.3: Arquitectura de la aplicacion desarrollada.

3.2.1. Capa de presentación

El usuario puede interactuar con la extensión a través de tres secciones: el menú de configuración de la extensión, el popup de la extensión y el menú principal que se encuentra en la barra lateral.

El popup de la extensión puede ser abierto a través del icono situado a la derecha de la barra de navegación. Una vez desplegado permite al usuario iniciar sesión en el gestor de proyecto introduciendo las credenciales correspondientes y una vez logueado muestra el mensaje de advertencia y permite al usuario cerrar sesión.

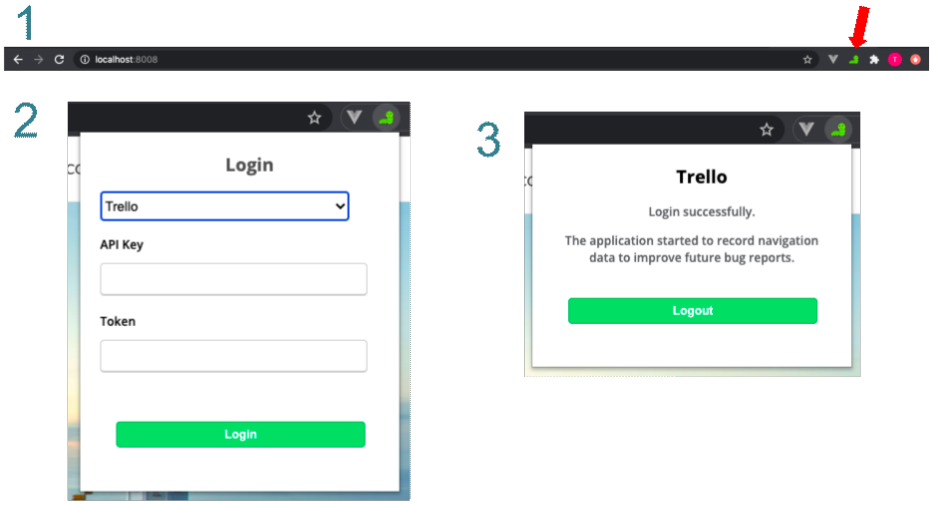

Figura 3.4: Icono para abrir el popup de la extensión, formulario de login y mensaje de exito.

Una vez que el usuario ha iniciado sesión aparece el menú colapsado para que este interactúe con el gestor de proyecto, primero seleccionando con qué proyecto está trabajando y luego completando el formulario para la creación de un nuevo reporte de error.

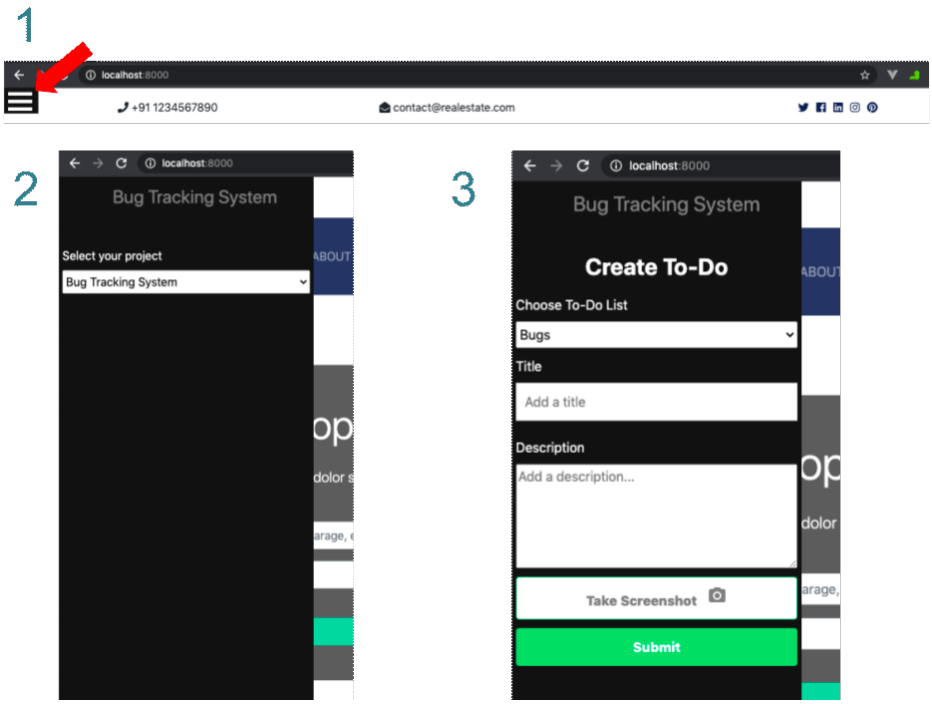

Figura 3.5: Menu colapsado de la extensión, selección de proyecto y formulario de creación de reporte.

Por último el usuario puede interactuar directamente con la extensión a través del menú de configuración de extensiones del navegador con el fin de agregar preguntas customizadas y selectores de DOM que más adelante veremos en detalle para que sirven.

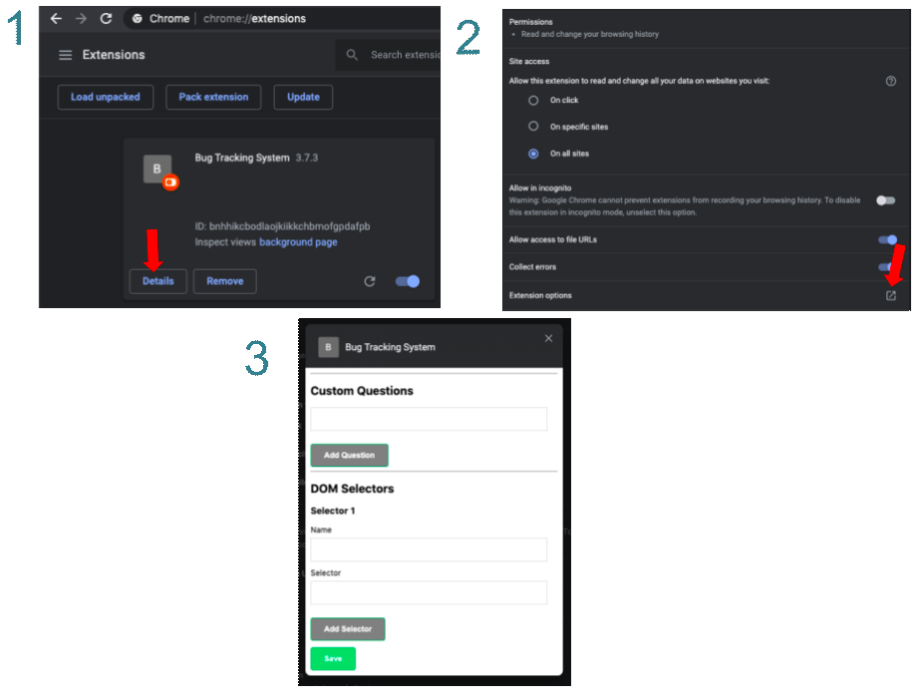

Figura 3.6: Opciones de configuración de la extensión desarrollada.

3.2.2. Capa de Aplicación

La lógica del sistema está compuesta por cuatro archivos principales respetando la arquitectura de una extensión de navegador como mencionamos anteriormente que son:

- background.js
- content.js
- options.js
- popup.js

background.js: encargado de recopilar el historial de navegación con la información de sus peticiones al servidor. Para esto mediante la interfaz provistas por el navegador nos quedamos a la escucha de dos eventos distintos:

- webRequest.onBeforeRequest: para guardar las últimas peticiones al servidor con sus parámetros incluidos.

```
48  browser.webRequest.onBeforeRequest.addListener(
49    logURL,
50    {urls: ["<all_urls>"]},
51    ['requestBody']
52  );
```

Figura 3.7: Código JavaScript para guardar peticiones al servidor.

- history.onVisited: para guardar las últimas 10 visitas del historial de navegación.

```
55  browser.history.onVisited.addListener(function(historyItem) {
56    if (historyItem.url.includes('localhost')) {
57      browser.storage.sync.get('navigationHistory', function(result) {
58        let navigation = result.navigationHistory || []
59        navigation.push(historyItem.url)
60        if (navigation.length >= 10) {
61          navigation = navigation.slice(1)
62        }
63        browser.storage.sync.set({ navigationHistory: navigation });
64      });
65    }
66  })
```

Figura 3.8: Código JavaScript para guardar historial de navegación.

content.js: al ser este script el encargado de ejecutarse en el contexto de la página, es el encargado de renderizar el menú principal, controlar sus acciones, guardar y recuperar el estado del mismo ante cada refresco del navegador. Además, se encarga de realizar capturas de pantalla ante cada click del usuario para luego generar un GIF animado y de la construcción de la descripción del reporte de error con toda la información recopilada por la extensión.

option.js: interfaz para guardar en la base de datos la configuración realizada por el usuario (preguntas custom y selectores DOM que se explicaran más adelante).

popup.js: encargado tanto del inicio y cierre de sesión como también de informar errores.

3.2.3. Servicios

La extensión debe comunicarse con la herramienta de gestión de proyecto que el usuario utiliza en su día a día (Trello, Basecamp, Jira, GitHub...) y para esto el proyecto cuenta con un directorio destinado a los adaptadores. Cada uno de estos respeta una interfaz para comunicarse con los otros componentes mediante pasaje de mensajes entre estos.

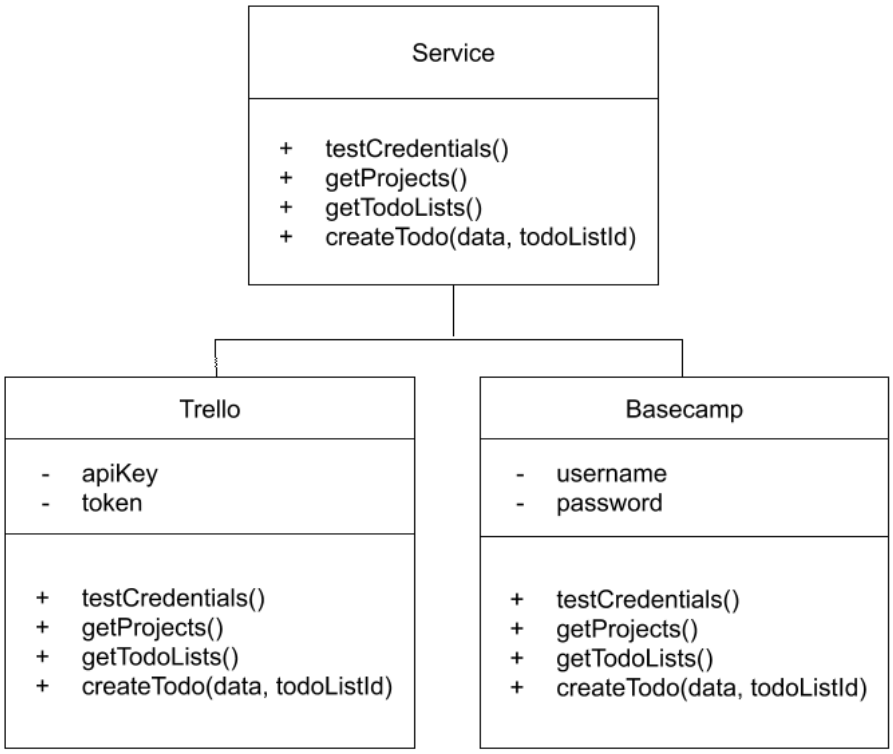

Figura 3.9: Diagrama UML de servicios.

testCredentials(): método utilizado desde el popup.js para corroborar que las credenciales ingresadas en el formulario de inicio de sesión sean correctas.

getProjects(): ya que un usuario puede estar asociado a muchos proyectos dentro de su gestor de proyecto, este método es utilizado para obtener

el listado de proyectos y así permitirle que seleccione en cuál quiere crear el reporte.

getTodoLists(): al igual que el método anterior este retorna todas las posibles listas de tareas/reportes para ser seleccionada.

createTodo(data, todoListId): es el encargado de crear una nueva tarea en el proyecto y lista seleccionados previamente, como así también de adjuntar las capturas de pantalla y el GIF animado.

- data: información requerida para crear el reporte (titulo, descripción, imágenes)
- todoListId: identificador del listado en donde se va a añadir la tarea.

3.2.4. Bases de Datos

Chrome Storage: persistencia de información que se utiliza internamente en la extensión, en ocasiones se utiliza como mecanismo de comunicación entre los componentes de la misma. Se guardan credenciales autenticadas, información del historial de navegación, etc.

LocalStorage: utilizado solo a modo de lectura para completar el reporte.

SessionStorage: igual que LocalStorage pero además se utiliza para mantener el estado de la extensión y lo ingresado en los formularios para evitar la pérdida de información ante los refrescos del navegador.

Indexed DB: utilizamos indexedDB para persistir imágenes ya que es la base de datos con más memoria disponible de un navegador. Persistimos tanto las imágenes tomadas manualmente mediante el botón "Take Screenshot" del panel, así como también las imágenes tomadas automáticamente mediante cada acción que se convertirán en GIF al enviar el reporte.

3.3. Tecnologías

En esta sección se mencionan las principales tecnologías utilizadas durante el desarrollo de la herramienta. Se describe brevemente cada una de ellas.

- **JQuery**
- **Html2canvas**
- **Jsgif**
- **IDB-Keyval**
- **Npm**
- **Git Version Control**
- **IndexedDB**
- **Chrome Extensions**

Chrome Extensions: Las extensiones son pequeños programas de software que customizan la experiencia de navegación. Permiten a los usuarios adaptar la funcionalidad y el comportamiento de Chrome a las necesidades o preferencias individuales. Están desarrollados con tecnologías web como HTML, JavaScript, y CSS [5]. Una extensión debe cumplir un único propósito que se define de forma limitada y fácil de entender. Una única extensión puede incluir múltiples componentes y un rango de funcionalidades, siempre y cuando todo contribuya a un mismo propósito. Optamos por utilizar esta tecnología debido a su amplia documentación y uso de la misma pero, más importante, por el hecho de que el proyecto está enfocado a la optimización de reporte de bugs y errores en sistemas web [20].

jQuery: es una rápida, pequeña y feature-rich librería de JavaScript. Hace que cosas como el desplazamiento y la manipulación de documentos HTML, el manejo de eventos, la animación y AJAX [6] sean mucho más simples con una API fácil de usar, que funciona en una multitud de navegadores. Con una combinación de versatilidad y extensibilidad, jQuery ha cambiado

[5] Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.

[6] Acrónimo de .ªsynchronous JavaScript And XML", es una técnica de desarrollo web para crear aplicaciones interactivas.

la forma en que millones de personas escriben JavaScript. Utilizamos jQuery a lo largo de toda la aplicación, principalmente para el manejo de eventos sobre el DOM así como también para la realización de peticiones XHR [7] a las APIs de los gestores de proyectos mediante AJAX.

npm (Node Package Manager): es el gestor de paquetes más grande del mundo. Desarrolladores open-source de todos los continentes utilizan npm para compartir y acceder a paquetes, así como también muchas organizaciones utilizan npm para el manejo de paquetes de desarrollos privados. npm consiste en tres componentes diferentes:

- **Website:** se utiliza para encontrar paquetes, configurar perfiles y manejar otros aspectos de tu experiencia en npm. Por ejemplo, se puede configurar Orgs (organizaciones) para manejar el acceso a paquetes privados o públicos.
- **Interfaz de comando (CLI):** La CLI corre sobre una terminal, es la forma de interactuar con npm.
- **Registro:** es una gran base de datos pública de software JavaScript y la meta-información que lo rodea.

Utilizamos npm para la descarga de librerías utilitarias de JavaScript (Html2canvas, Jsgif, IDB-Keyval)

IndexedDB(API de base de datos indexada): es una API de bajo nivel de JavaScript proporcionada por navegadores web para administrar una base de datos NoSQL de objetos JSON que ofrece almacenamiento en el cliente de cantidades significativas de datos estructurados, incluyendo archivos y BLOBs [8]. Para búsquedas de alto rendimiento en esos datos usa índices. Mientras DOM Storage es útil para el almacenamiento de pequeñas cantidades de datos, no es útil para almacenar grandes cantidades de datos estructurados. IndexedDB proporciona una solución.

Utilizamos IndexedDB para guardar un JSON que contiene una lista con las capturas de pantalla que son tomadas durante la navegación del sitio web. Debido a que necesitamos conservarlas en alta resolución, ni el LocalStorage ni el SessionStorage son de utilidad ya que el límite de almacenamiento de estos es muy limitado:

[7] XMLHttpRequest (XHR) es un objeto especial de JavaScript que permite realizar peticiones HTTP asíncronas (AJAX) de forma nativa desde JavaScript.

[8] Los BLOB son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica.

Mobile browsers:

Browser	Chrome	Android Browser	Firefox	iOS Safari
Version	40	4.3	34	6-8
Available	10MB	2MB	10MB	5MB

Desktop browsers:

Browser	Chrome	Opera	Firefox	Safari	IE
Version	40	27	34	6-8	9-11
Available	10MB	10MB	10MB	5MB	10MB

Figura 3.10: Comparación de almacenamiento por navegador.

IDB-Keyval: es un almacenamiento súper simple clave-valor basado en promesas implementado con IndexedDB.

Utilizamos IDB-Keyval para simplificar el acceso y manejo de la información mediante promesas y debido a lo complejo que resulta tratarla con IndexedDB puro.

Jsgif: es una librería escrita puramente en JavaScript para la conversión de GIF y la utilizamos para la creación del GIF a partir de las capturas de pantalla tomadas durante la navegación.

Html2canvas: es una librería que permite la toma de capturas de pantalla de páginas web o secciones de estas directamente desde el navegador. Las capturas de pantalla están basadas en el DOM y pueden no ser cien por ciento una representación real de cómo se vería una captura de pantalla común y corriente, pero construye la captura basada en la información disponible en la página web. Utilizamos esta librería para agregarle la funcionalidad de que desde el plugin el usuario pueda realizar capturas de pantalla sin tener que realizarlas con su computadora y luego tener que cargarlas, sino que al realizar la toma de las capturas con el botón provisto en el plugin, estas se almacenan y son mostradas inmediatamente en el panel.

GIT Version Control: es un software de control de versiones diseñado por Linus Torvalds [9], pensando en la eficiencia y la confiabilidad del mante-

[9]Linus Benedict Torvalds es un ingeniero de software finlandés-estadounidense, conocido por iniciar y mantener el desarrollo del kernel Linux, basándose en el sistema operativo libre Minix creado por Andrew S. Tanenbaum y en algunas herramientas, varias utilidades y los compiladores desarrollados por el proyecto GNU.

nimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos. Utilizamos GIT durante todo el desarrollo del proyecto para agilizar y sincronizar nuestro trabajo.

Capítulo 4

Herramienta de reporte: instalación, configuración y uso

4.1. Instalación

La instalación se puede realizar de dos maneras, desde el Web Chrome Store o utilizando un ZIP [1] con el proyecto.

Utilizando el ZIP del proyecto los pasos a seguir serían:

1. Descomprimir el ZIP.
2. Ingresar a la pestaña de extensiones del navegador (opciones → configuración → extensiones).
3. Presionar el botón "Load unpacked", seleccionar la carpeta del proyecto y presionar "Open".

[1] Es un formato de compresión sin pérdida, muy utilizado para la compresión de datos como documentos, imágenes o programas.

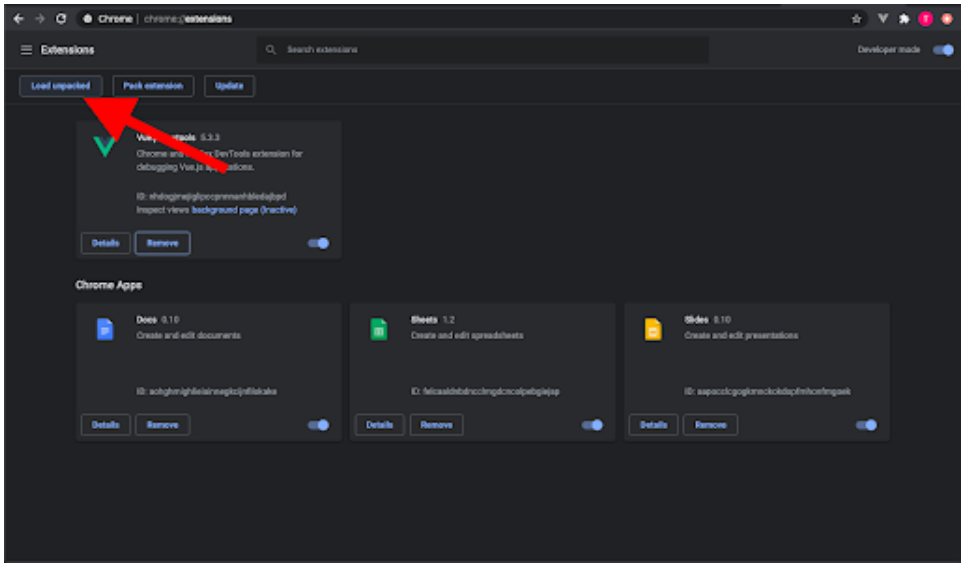

Figura 4.1: Pantalla de administración de extensiones del navegador Google Chrome. Botón de carga de nueva extensión.

Una vez instalada, la extensión aparecerá listada en este panel donde se podrá leer el detalle y las especificaciones de esta y podrá ser configurada o eliminada. También aparecerá el icono en color gris en la esquina superior derecha indicando que está instalada pero no activa en esta pestaña.

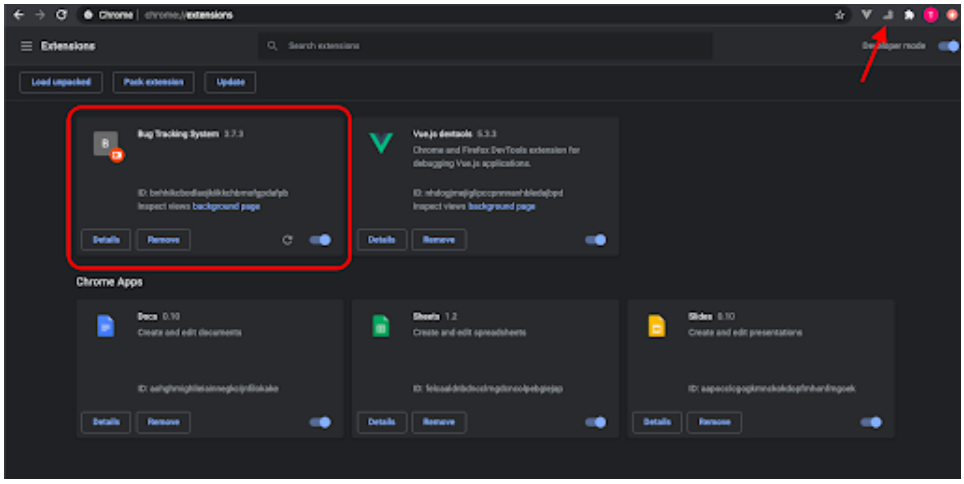

Figura 4.2: Pantalla de administración de extensiones del navegador Google Chrome. Indicaciones de extensión instalada.

4.2. Configuración

La extensión puede ser configurada posterior a su instalación para que agregue cierta información extra en los reportes. Para esto se debe ir al listado de extensiones instaladas de su navegador (opciones → configuración → extensiones), seleccionar la extensión deseada, en nuestro caso Bug Tracking System y dentro de esta se encuentra un icono de "ajustes" que al presionarlo va a abrir un popup con un formulario para configurar.

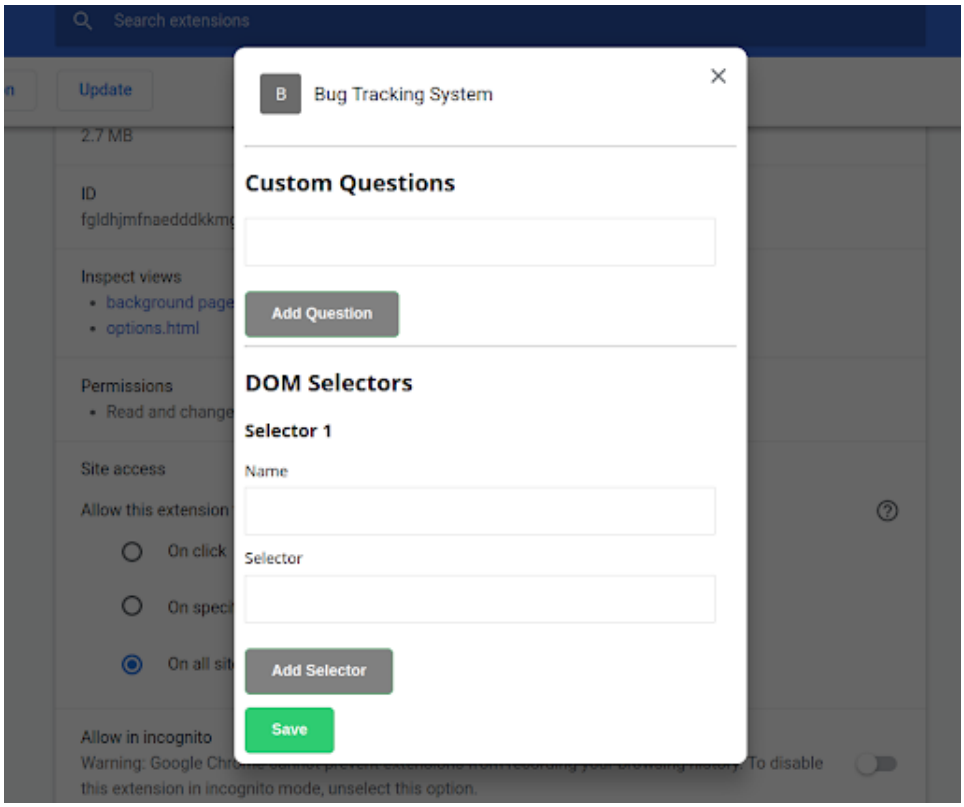

Figura 4.3: Formulario de configuración de la extensión, preguntas custom y selectores DOM.

Dentro de las configuraciones posibles podemos encontrar dos secciones:

- **Custom Questions:** son preguntas personalizadas para cada proyecto que el reportador va a tener que contestar obligatoriamente cada vez

que quiere realizar un reporte de error. Ejemplo: “Es el error repetible o sucedió por única vez?”

- **DOM Selectors:** es el contenido de los elementos del DOM que se van a agregar a los reportes, para poder configurar esta sección debemos utilizar la sintaxis de jQuery Selectors para seleccionar los elementos DOM del HTML. (ejemplo: algunID, .customclass, etc)

4.3. Uso

Una vez instalada y configurada la extensión, el usuario encargado de testear la aplicación debe iniciar sesión con su herramienta de gestión del proyecto. Para esto, cuenta con un formulario en el cual puede elegir entre los distintos gestores de proyectos soportados por la extensión, con su correspondiente formulario de ingreso de credenciales.

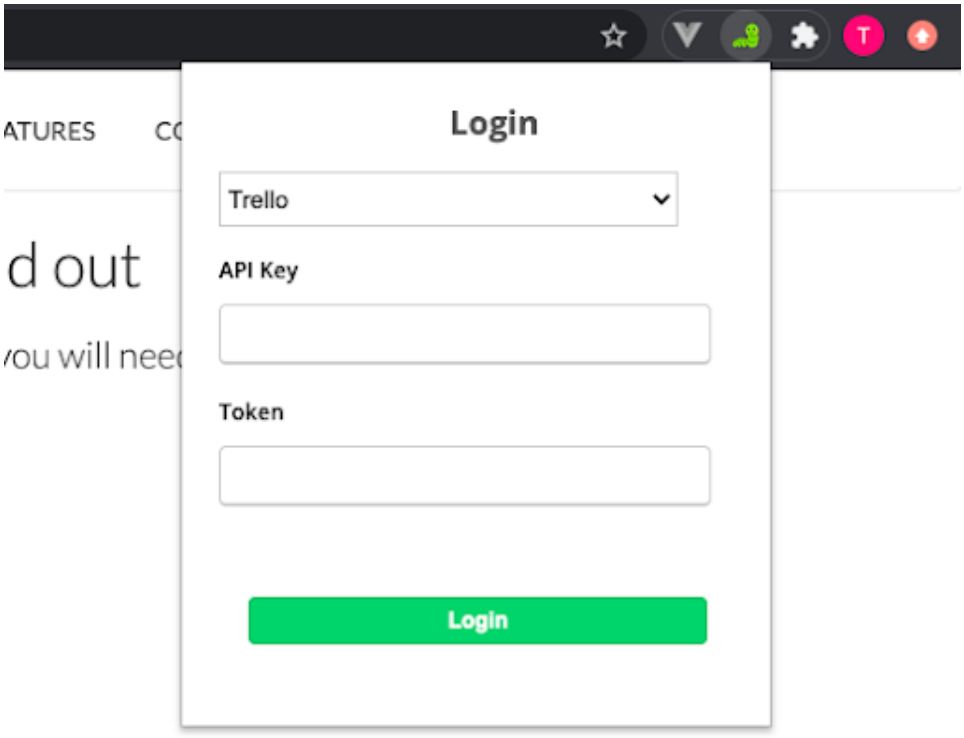

Figura 4.4: Formulario de inicio de sesión en la extensión utilizando Trello.

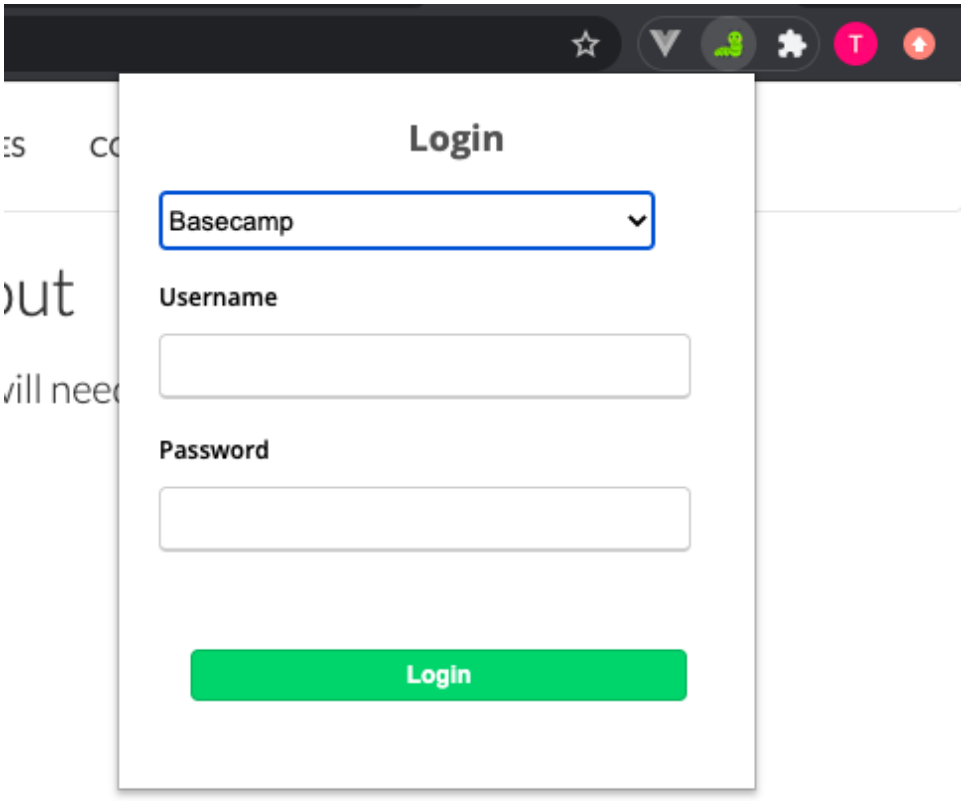

Figura 4.5: Formulario de inicio de sesión en la extensión utilizando Basecamp.

Al ser ingresadas las credenciales, de ser correctas, se podrá apreciar un mensaje que indica en cuál herramienta de gestión de proyectos hemos iniciado sesión y nos notifica que la extensión comenzó a capturar información del contexto para mejorar los futuros reportes de error.

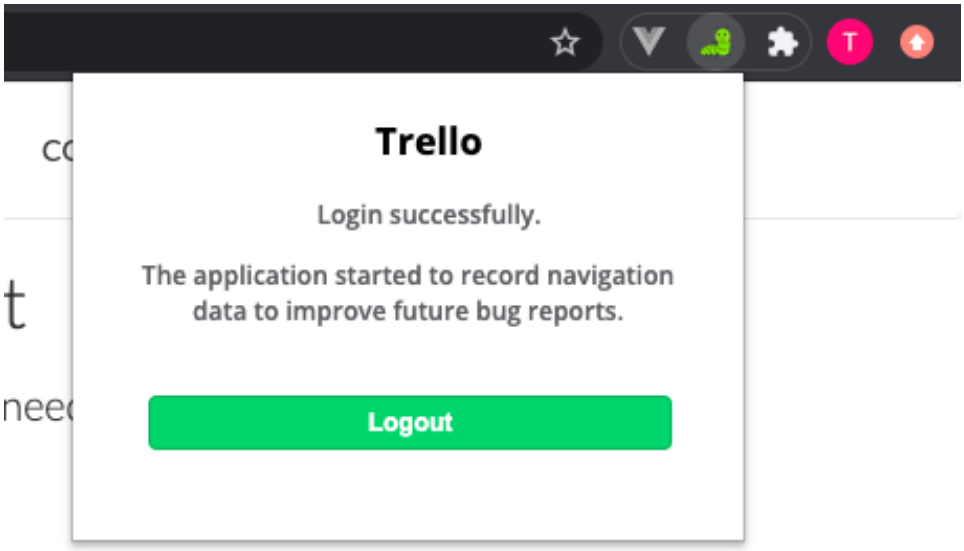

Figura 4.6: Mensaje de inicio de sesión exitoso en Trello.

En el caso de que se hayan ingresado credenciales erróneas o no sean reconocidas por el gestor de proyectos, la extensión nos notificara dicho error para poder reintentar.

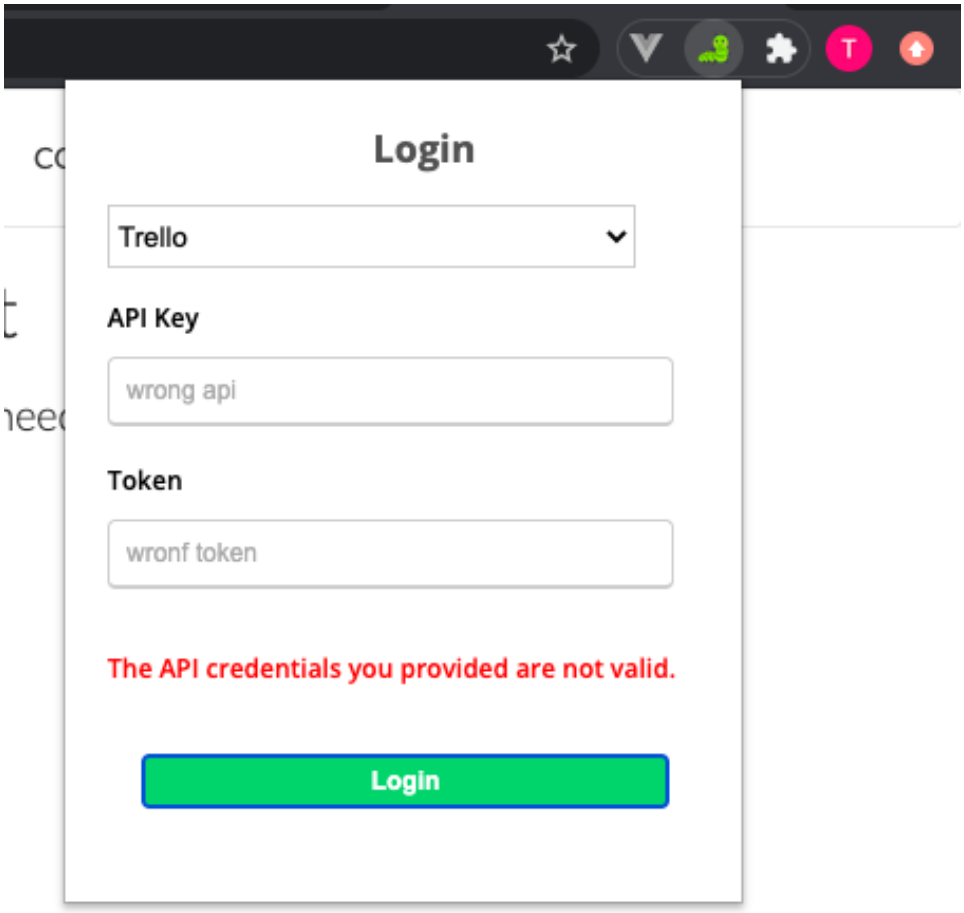

Figura 4.7: Mensaje de error al intentar iniciar sesión en Trello.

Después de iniciar sesión satisfactoriamente aparecerá en el margen superior izquierdo un icono de menú que se expande al posicionar el cursor sobre este, y se colapsa automáticamente al quitar el cursor para no molestar ni interferir con la normal navegación del usuario dentro de la aplicación web.

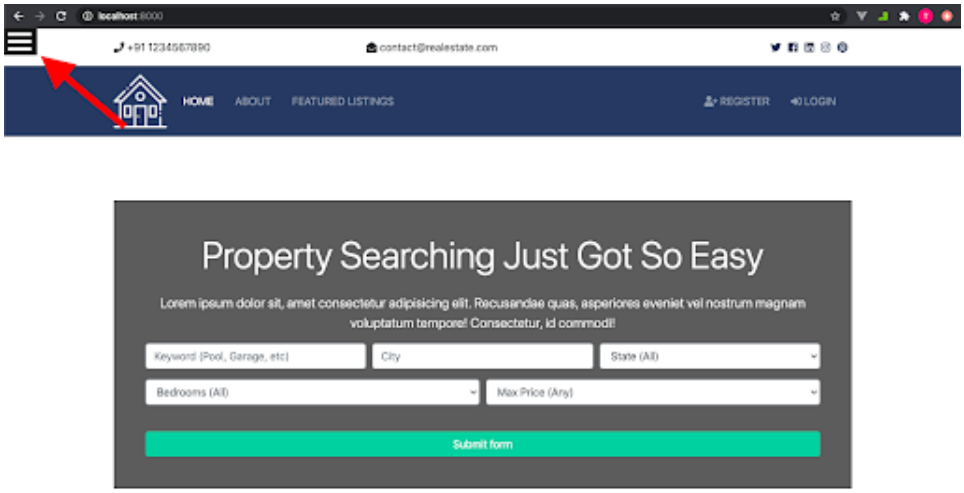

Figura 4.8: Menu de la extensión colapsado.

Al expandir el menú nos aparecerá un input de selección con el listado de proyectos en los cuales es miembro el usuarios de las credenciales provistas cuando se inicio sesión. Se deberá seleccionar el proyecto en el cual vamos a trabajar y subir los reportes.

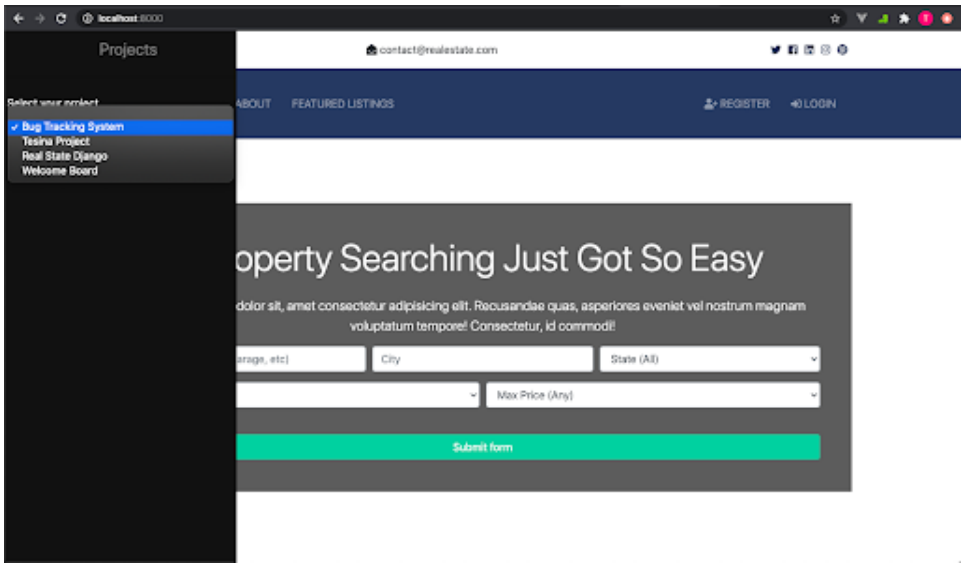

Figura 4.9: Selección de proyecto/pizarra de Trello desde la extensión.

Una vez seleccionado el proyecto, aparece el formulario de reporte de errores, el cual cuenta con los siguientes campos:

- **Nombre del proyecto:** indicador de cuál proyecto estamos trabajando
- **Input de selección de To-Do list:** listado al cual se creará el reporte.
- **Titulo:** titulo del reporte.
- **Descripción:** descripción del reporte.
- **Take Screenshot:** botón para tomar captura de la pantalla actual.
- **Preguntas personalizadas:** listado de campos agregados en la configuración de la extensión.
- **Submit:** botón de envío de formulario.

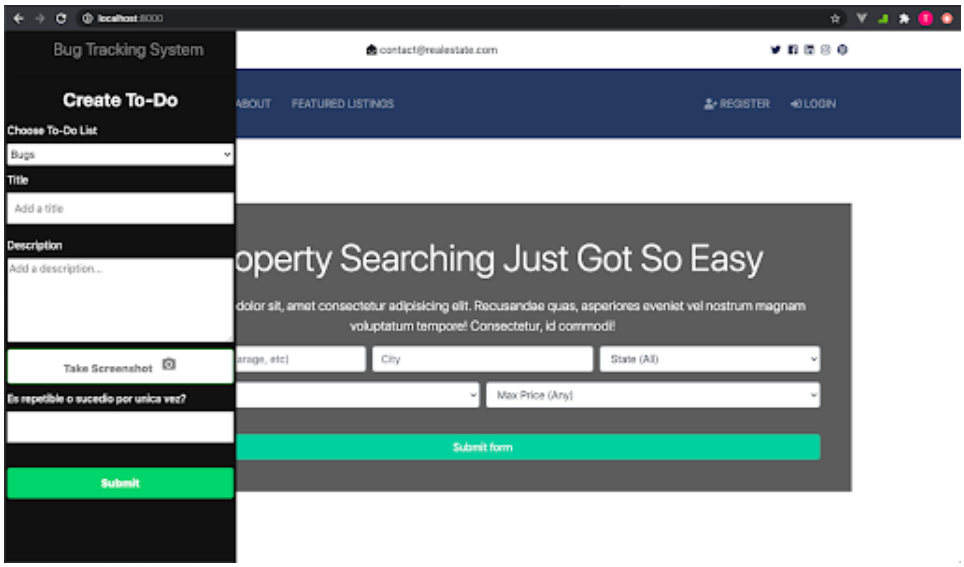

Figura 4.10: Formulario vacio de creación de reporte desde la extensión.

Al realizar capturas de pantalla, estas aparecerán en miniatura debajo del botón “Take Screenshot” con una cruz roja arriba a la izquierda de las mismas para poder removerlas del reporte de ser necesario.

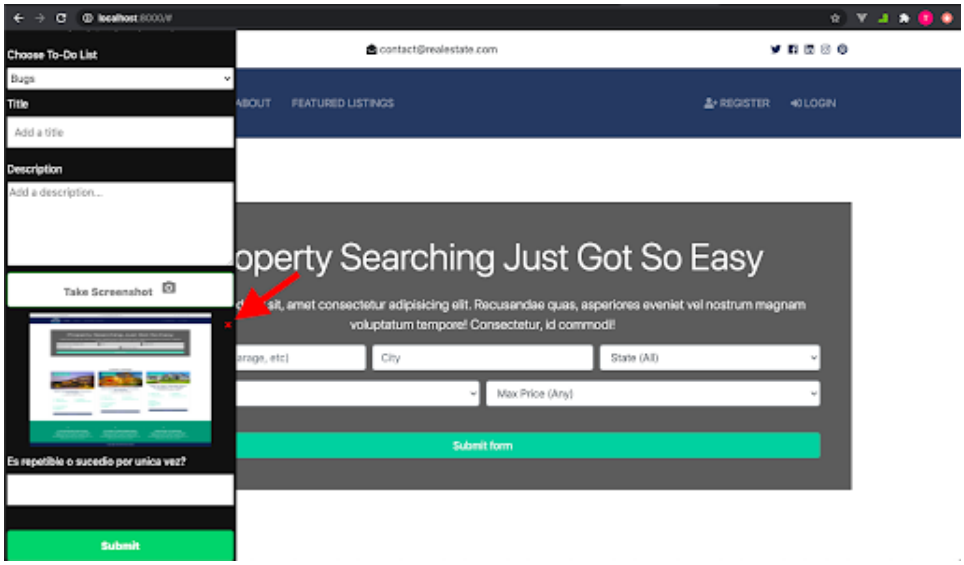

Figura 4.11: Señalización de cruz de eliminación de captura de pantalla.

Una vez enviado el reporte desde la extensión, este creará un nuevo To-Do en la lista de tareas seleccionada con la información ingresada manualmente (título, descripción, capturas de pantalla y preguntas predeterminadas). Además a este reporte se anexa la información que la extensión fue recopilando durante la navegación:

- **Current URL:** URL desde la que se realizó el reporte.
- **Custom DOM Info:** contenido de los elementos DOM configurados en las opciones de la extensión.
- **History:** listado de las últimas 10 URLs por las que pasó.
- **Requests:** peticiones al servidor (sincronicas y asincrónicas) con su información.
- **Local Storage:** contenido del LocalStorage.
- **Session Storage:** contenido del SessionStorage.
- **GIF:** GIF animado creado a partir de las capturas de pantalla tomadas al realizar cada click en la aplicación web.

4.4. Ejemplo de reporte con la extensión

A modo de ejemplo mostraremos como fue reportado uno de los cuatro errores utilizados en el siguiente capítulo llamado “Evaluación” del documento en el cual un usuario del sistema intenta registrarse al mismo sin completar todos los campos y recibe un error poco descriptivo.

Configuración de la extensión previo a su uso:

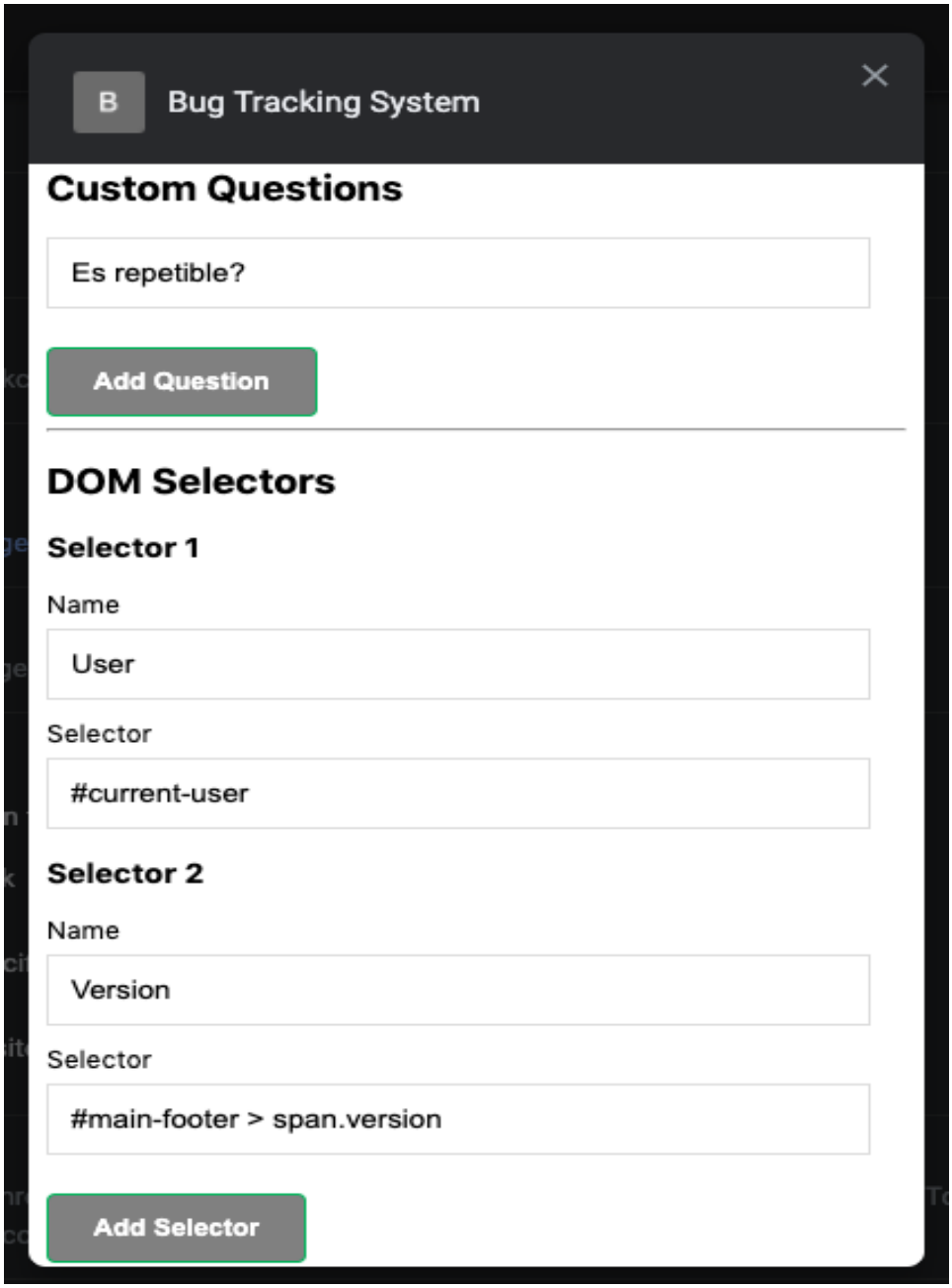

Figura 4.12: Formulario de opciones de configuración de la extension.

Ingreso de todos los campos excepto el apellido:

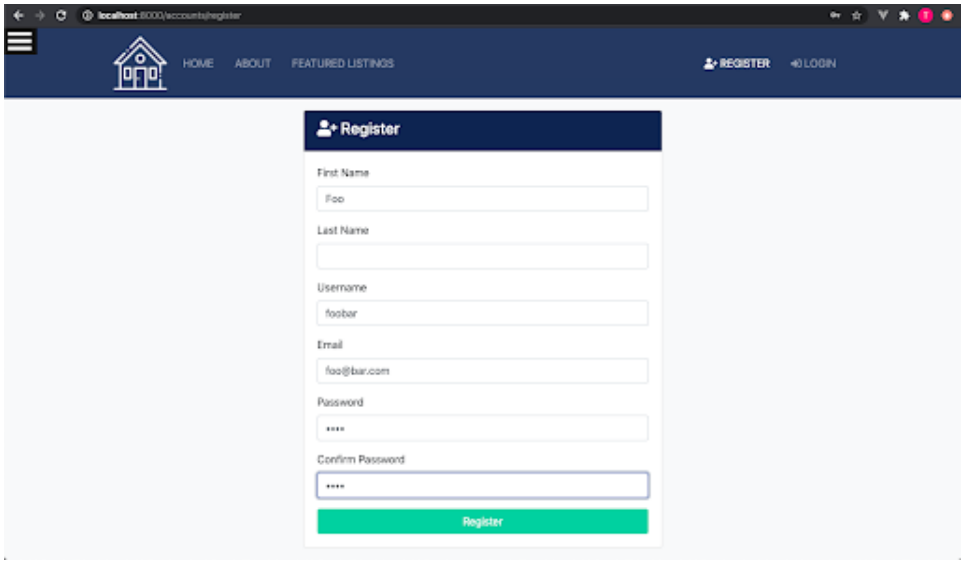

Figura 4.13: Formulario de registración en sitio web de propiedades.

Mensaje de error del sistema poco descriptivo:

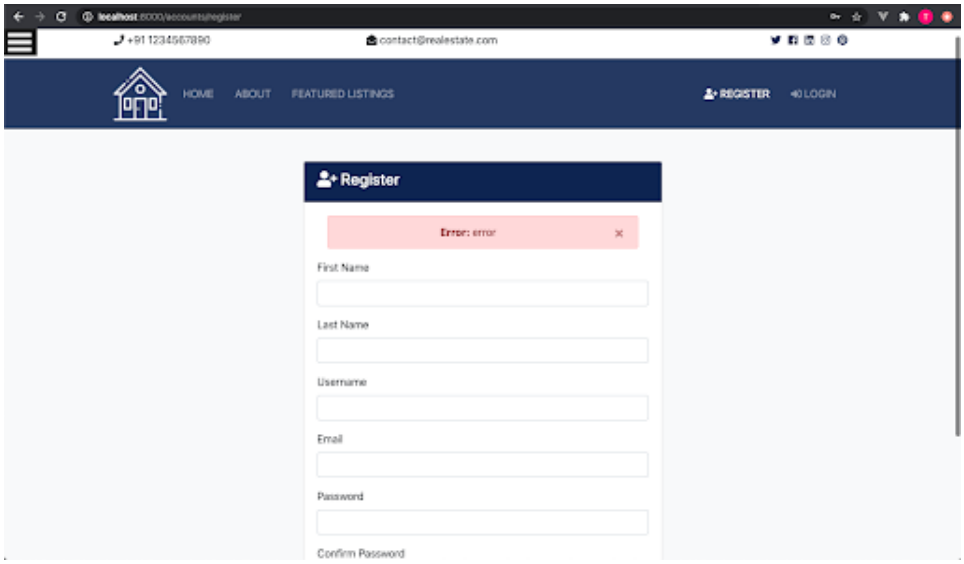

Figura 4.14: Error al submitear el formulario de registración en sitio web de propiedades.

Formulario de reporte con la información ingresada por el usuario:

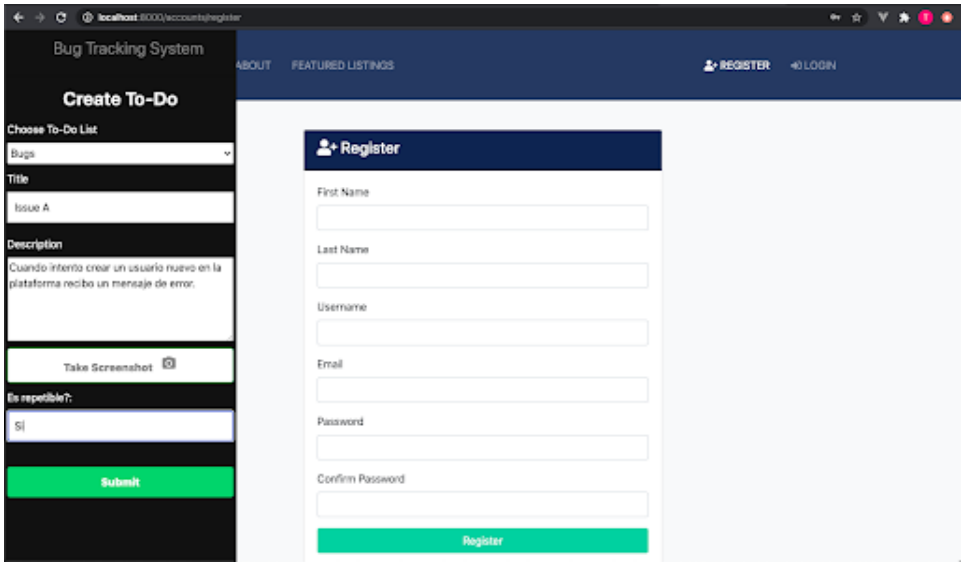

Figura 4.15: Ejemplo de reporte de error con la extensión del navegador.

Reporte creado en el gestor de proyecto, en nuestro caso Trello:

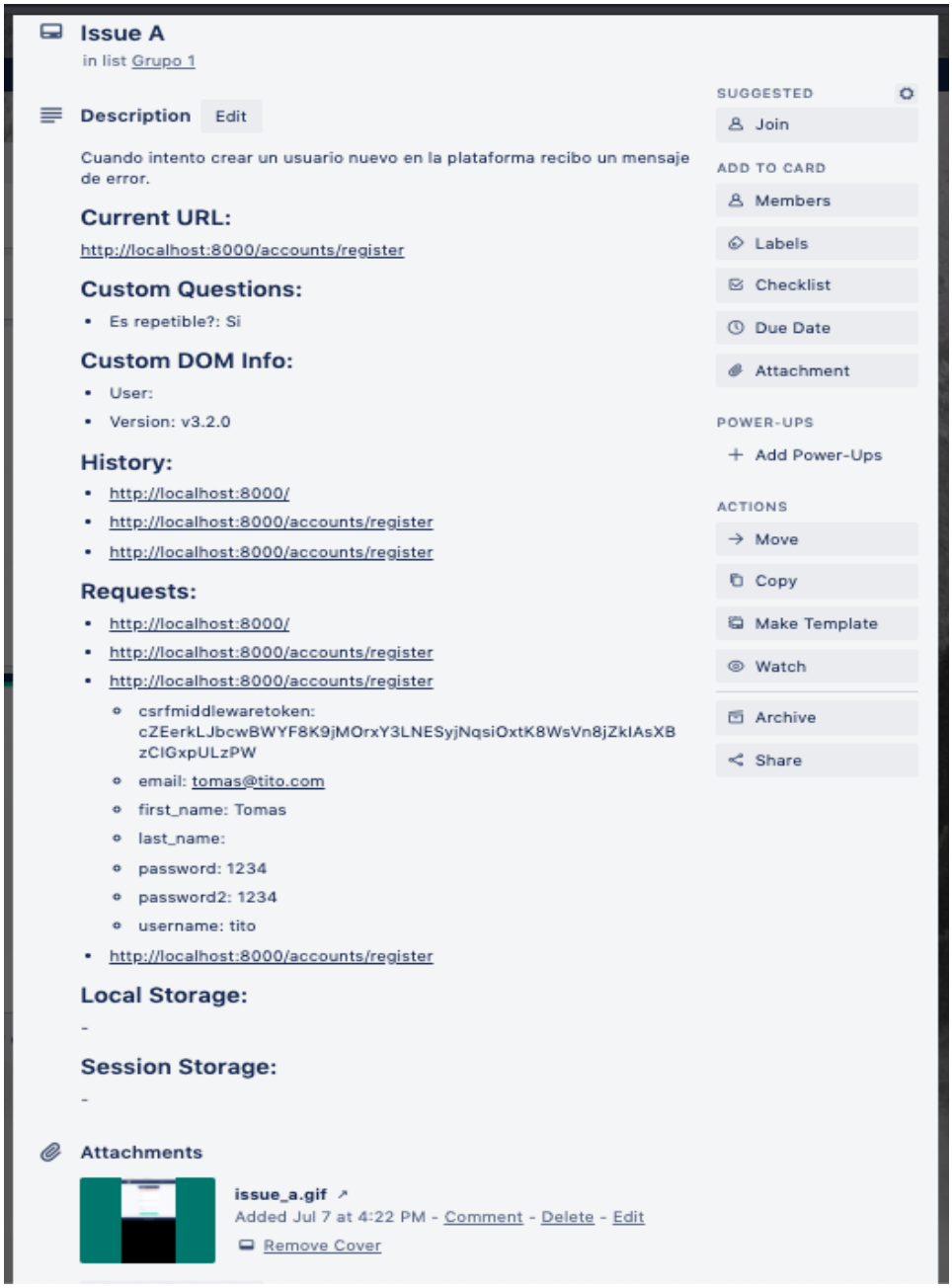

Figura 4.16: Card de Trello con la información cargada del reporte.

Capítulo 5

Evaluación

5.1. Introducción

Para comprobar que la utilización de la herramienta propuesta mejora el tiempo de respuesta de los desarrolladores a la hora de arreglar un bug/issue, realizamos un experimento en el cual le pedimos a un grupo de colaboradores que instalen un proyecto de software en donde se introdujeron una serie de errores. Dichos errores fueron reportados en un gestor de proyectos en el cual estos colaboradores acceden con el rol de desarrollador.

Los colaboradores se separararon en dos grupos y se los hizo identificar dos bugs reportados con la herramienta y dos reportados solo con titulo y descripción, con el fin de obtener métricas de si fue o no de utilidad la herramienta.

Además completaron dos encuestas, una inicial previa a la realización del experimento y una al finalizar el mismo.

5.2. Experimento / tarea propuesta

5.2.1. Encuesta inicial

Esta encuesta se realizó antes de comenzar con la preparación del proyecto y consta de cuatro preguntas con las que buscamos conocer el tipo de usuario que realiza los reportes en el día a día del colaborador, la calidad con la que lo hace y por último saber qué información le gustaría que siempre se brindará en los reportes y no lo hace.

1. En la escala de 1 a 5 (donde 1 es ninguno y 5 es mucho) que tantas dificultades o problemas soles tener a la hora de identificar los bugs o errores reportados?

2. En la escala de 1 al 5 (donde 1 es nunca y 5 muy frecuente) que tan frecuentemente soles necesitar de un intercambio de mensajes con el reportador para identificar los problemas?

3. ¿Qué tipo de usuario es el que realiza los reportes de errores en tu trabajo?

   a) Desarrollador
   b) Tester
   c) Usuario del sistema
   d) Otro

4. ¿Cuáles de los siguientes items te gustaría que siempre se incluyan en los reportes de error?

   a) Pasos para reproducir el error
   b) URL donde se genero el error
   c) Datos ingresados en formulario si aplica
   d) Stack Trace (URLs por las que paso hasta llegar al error)
   e) Capturas de pantalla
   f) Navegador y versión
   g) Sistema operativo
   h) Comportamiento esperado
   i) Severidad/urgencia
   j) Otro

5.2.2. Proyecto

Tomamos un proyecto open source hecho en Django ya que es una de las tecnologías que más usamos y al ser Python [1] un lenguaje fácil de aprender y entender a simple vista, ayuda a los participantes del experimento a concentrarse en la identificación de los errores y no tanto en las características propias del lenguaje.

El proyecto en cuestión es un sitio web para la publicación de inmuebles, creación de martilleros y consultas de usuarios registrados o anónimos sobre inmuebles. El proyecto a grandes rasgos posee:

- Login
- Registro de usuarios
- Listado y filtros de inmuebles
- Listado de martilleros
- Formulario de consulta sobre un inmueble
- Detalle de inmueble
- Administrador

Repositorio: `https://github.com/TheCaffeineDev/Real-Estate-Django-Web-App`

Homepage: es la página principal y cuenta con un filtro de propiedades que permite realizar búsquedas por ciudad, estado, palabras claves, número de habitaciones y precio máximo. Seguido de este se encuentra el listado de propiedades luego de aplicar el filtro.

[1] es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

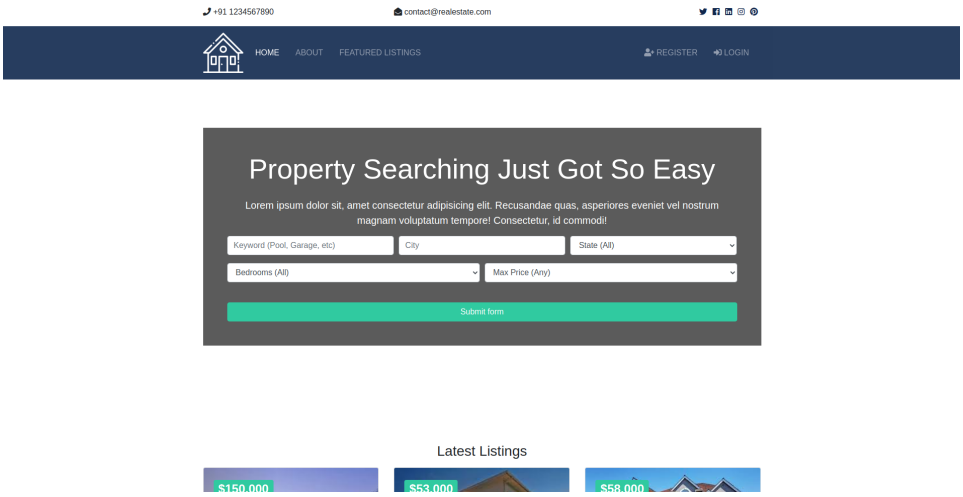

Figura 5.1: Pagina principal del proyecto.

Register: un formulario sencillo de registración de usuario con nombre, apellido, nombre de usuario, email y contraseña.

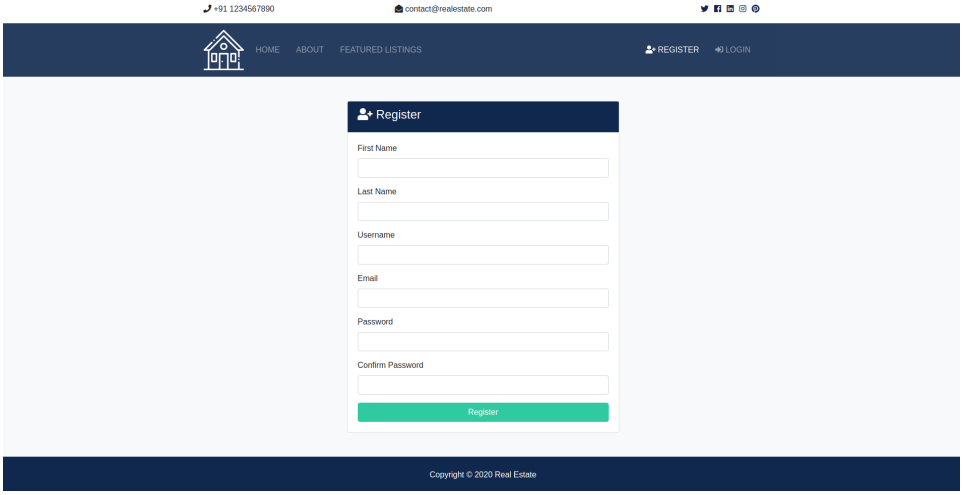

Figura 5.2: Pagina de registro del proyecto.

Listado de inmuebles: listado completo de propiedades con sus características principales y su precio.

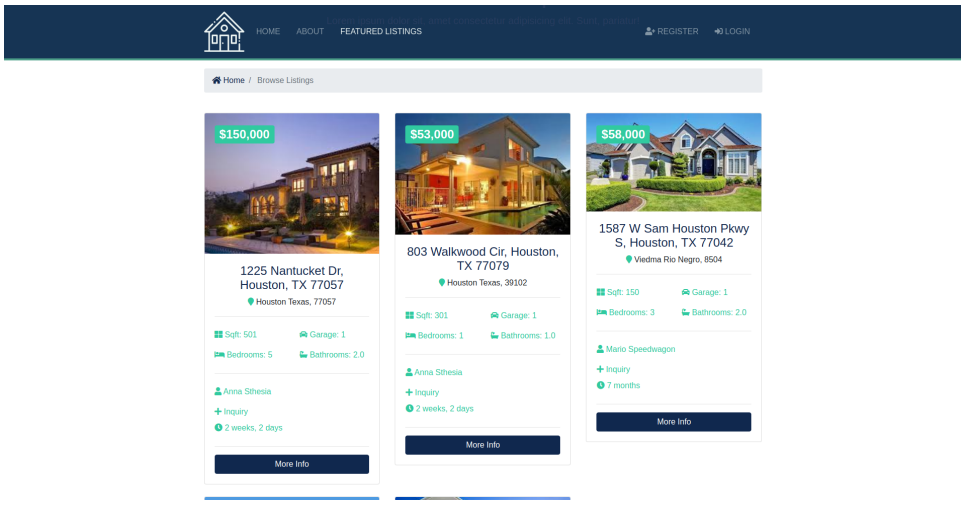

Figura 5.3: Pagina de listado de inmuebles.

Detalle de inmueble: detalle completo de la propiedad con características principales, texto descriptivo de la misma y martillero encargado.

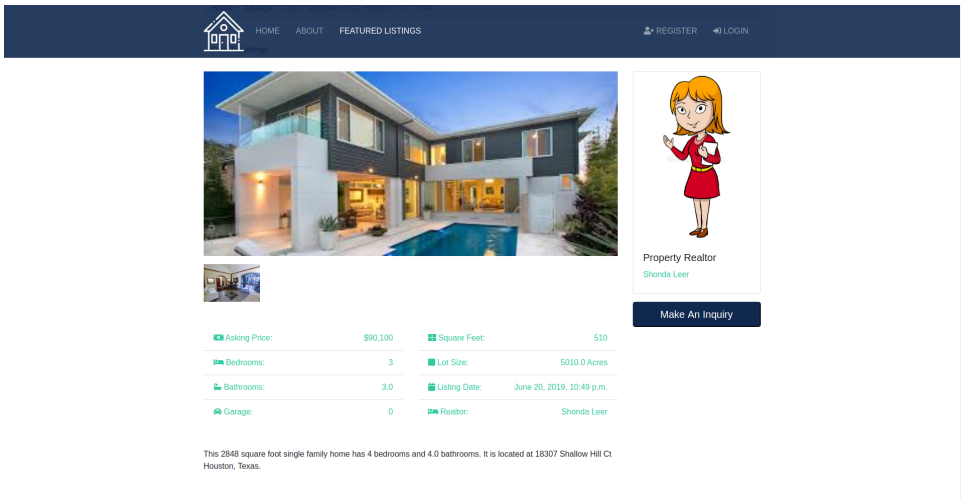

Figura 5.4: Pagina de detalle de un inmueble.

Consulta por inmueble: formulario de consulta sobre inmuebles con propiedad, nombre, email, teléfono y un mensaje.

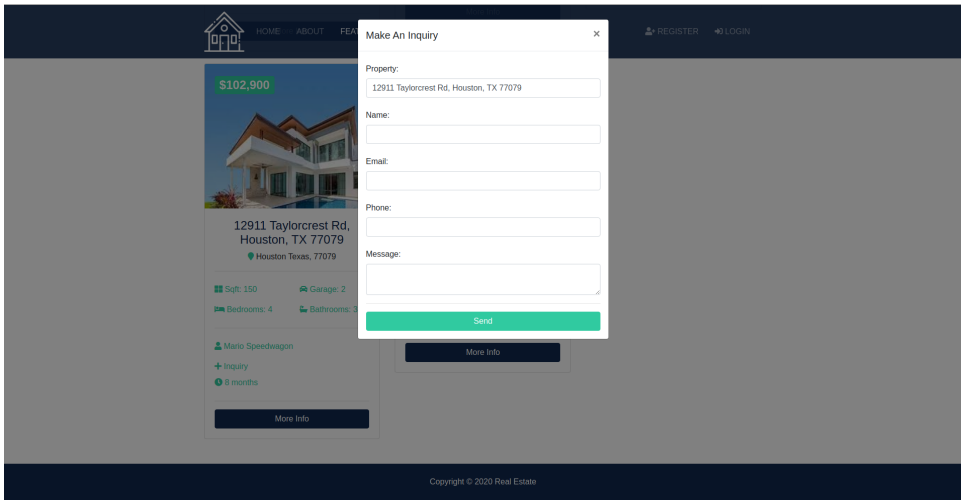

Figura 5.5: Pagina de consulta de un inmueble.

El propósito de utilizar un proyecto open source que no fuera programado por nosotros, es para mostrar que la herramienta no es dependiente de la arquitectura del sitio web, sino que sirve para cualquier desarrollo de estas características, agilizando la identificación de errores incluso si el desarrollador es nuevo en el proyecto.

5.2.3. Bugs/Issues

A este proyecto se le introdujeron 4 bugs (A, B, C, D), cada uno de estos bugs se incluyeron en una rama de GIT separada sin que quede registro en el historial de commits (para evitar que los desarrolladores hagan trampa al revisar el historial de commits de GIT y vean los bugs que introdujimos). Como algunos bugs no son sencillos de resolver y para mantener un esquema de tiempos de resolución similar para todos los desarrolladores, al transcurrir 6 minutos desde el inicio de la toma de un issue se simuló la petición de más información por parte del desarrollador con una respuesta predeterminada para cada issue brindando más detalle del porque del mismo. Esta respuesta predeterminada busca simular los casos de la vida real donde con la información que se brinda inicialmente, el desarrollador no logra identificar el

error y pide mas detalles de como reproducirlo y el reportador contesta con más información de utilidad.

1. **Issue A:**
   Descripción del reportador: "Cuando intento crear un usuario nuevo en la plataforma recibo un mensaje de error."
   Motivo: si el usuario no ingresa el nombre y/o apellido en el formulario de registración, un mensaje con el texto "Error" es lo que obtiene como respuesta. El reportador no da detalle de cuáles fueron los datos ingresados.
   Respuesta predeterminada: "Complete todos los datos menos el apellido."

2. **Issue B:**
   Descripción del reportador: "Cuando quiero realizar un Inquiry para una propiedad no me abre el popup/modal."
   Motivo: una Inquiry/consulta se puede realizar desde dos lugares, el detalle de una propiedad o en cada una de estas en el listado. Solo en el listado no anda y el reportador no especifica dónde ocurre.
   Respuesta predeterminada: "El botón que no anda es el del listado de propiedades en la sección de Featured Listing."

3. **Issue C:**
   Descripción del reportador: "A veces cuando filtro por max-price recibo un error."
   Motivo: En la definición del backend de las opciones para filtrar, una de estas contiene caracteres inválidos que producen un error al intentar listarlos. El reportador no especifica cual de todas es la que produce el error.
   Respuesta predeterminada: "Cuando filtro por $800.000 recibo el error."

4. **Issue D:**
   Descripción del reportador: "Cuando genero un Inquiry no me esta guardando el Teléfono de contacto."
   Motivo: En el template se encuentra repetido el input de teléfono pero la segunda vez no está visible, esto produce que al enviarse los datos del formulario al backend se envíe el teléfono como una lista con dos valores en lugar de un único string. El reportador no puede agregar mucha más información para facilitar la resolución.

Respuesta predeterminada: "Si utilizo el formulario del listado de propiedades anda bien, pero si utilizo el del detalle de una propiedad no se guarda.".

Estos cuatro issues fueron introducidos buscando lograr un equilibrio en cuanto a la complejidad que requiere cada uno para resolverlos para no poner en desventaja a ninguno de los dos grupos. Si bien para todos los issues el historial de navegación y el GIF son de utilidad para hacerse de una idea rápidamente de como llegar a la pantalla donde ocurre, en el issue A y D la información de mayor utilidad es la enviada en el formulario y en el B y C la URL donde ocurrió el error.

5.2.4. Reportador

Le pedimos a un usuario ajeno a la informática que utilice la aplicación y reporte los 4 bugs encontrados utilizando la herramienta, ya que si lo realizábamos nosotros u otro desarrollador, podríamos haber introducido información relevante para la resolución del bug que un usuario regular omitiría. Usualmente los usuarios de las aplicaciones que no se dedican al desarrollo de software al reportar un bug agregan información innecesaria que no ayuda a la hora de intentar replicar el error.

5.2.5. Tracking de tiempo y Board

Para poder medir el tiempo de respuesta en la resolución de los bugs y poder comparar, seleccionamos 20 desarrolladores. A estos desarrolladores los separamos en dos grupos (Grupo 1 y Grupo 2) de 10 personas cada uno. Debido a que la métrica se basa en tomar los tiempos de resolución de cada bugs, hemos armado los grupos en base a los años de experiencia de cada desarrollador para lograr un equilibrio, ya que al ser bugs típicos de desarrollo web los desarrolladores más experimentados suelen identificarlos y resolverlos en un tiempo menor.

Para poder registrar el tiempo de resolución de cada bug y brindarle al desarrollador los reportes, a cada uno se le asignó un board de Trello que contiene tres listas de cards/tareas:

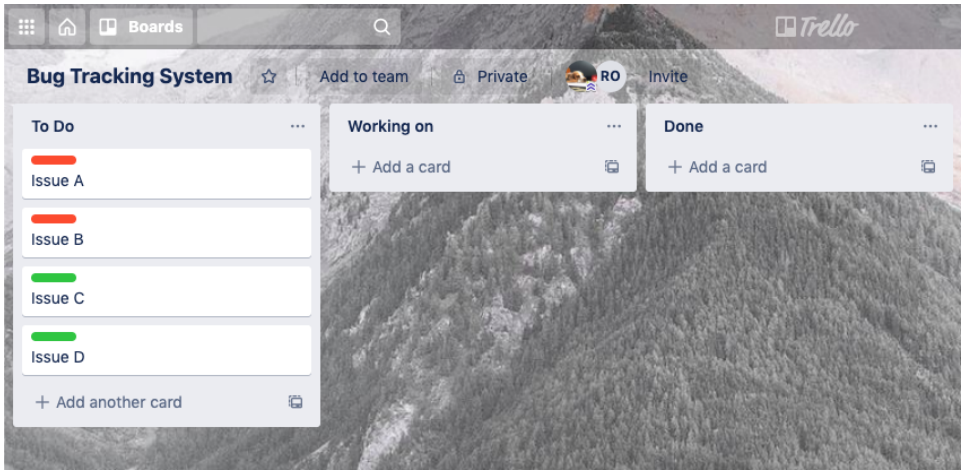

Figura 5.6: Board de Trello.

- **To Do:** esta lista contiene los cuatro issues a resolver, dos con el reporte completo realizado por nuestra herramienta y los dos restantes sin ella.
- **Working On:** contiene la card del issue en el que se encuentra trabajando actualmente.
- **Done:** contiene las cards de los issue que ya fueron resueltos.

El desarrollador tomó uno de los bugs de la lista de **"To Do"** y lo movió a la lista **"Working On"**.

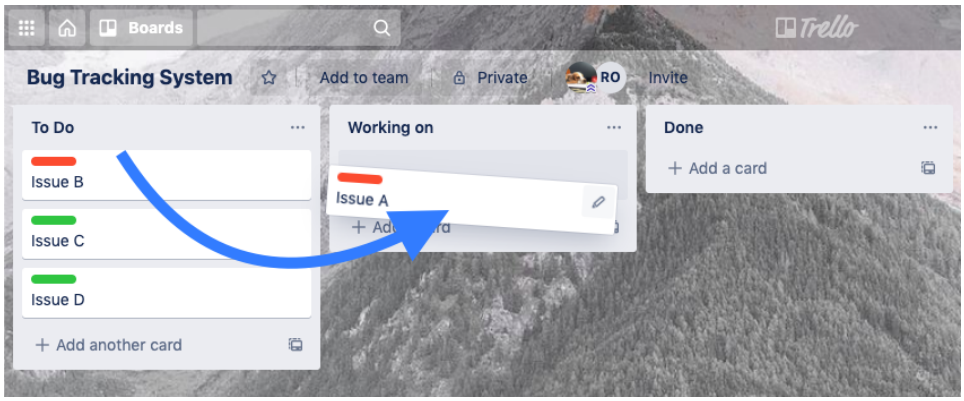

Figura 5.7: Movimiento de tarea de un listado a otro en Trello.

Al realizar esta acción Trello guarda automáticamente un registro con la fecha y hora exacta del momento en el que la card fue trasladada de la lista de **"To Do"** a la lista **"Working On"** que podemos ver en la sección actividad dentro del detalle de la card.

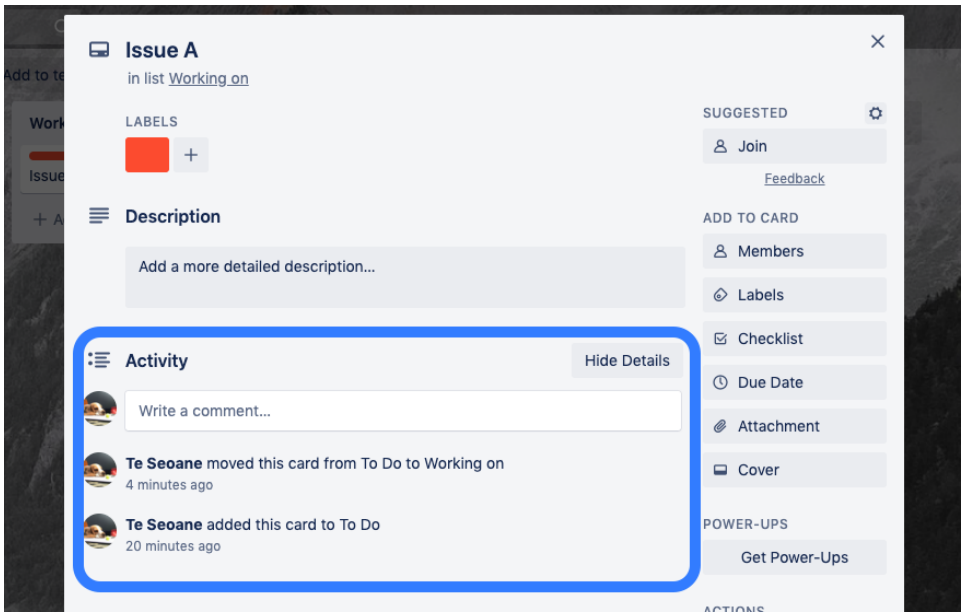

Figura 5.8: Log de Trello al mover cards.

Una vez resuelto el bug, el desarrollador mueve su correspondiente card de la lista **"Working On"** a la lista **"Done"**, generando así otro registro con la fecha y hora.

Para hacer el cálculo del tiempo de resolución del issue y poder aplicar las métricas correctamente, utilizamos los dos registros de la sección de actividad de Trello para saber cuando comenzó a trabajar en el issue y cuando terminó de resolverlo.

Ejemplo Issue A reportado con la Herramienta:

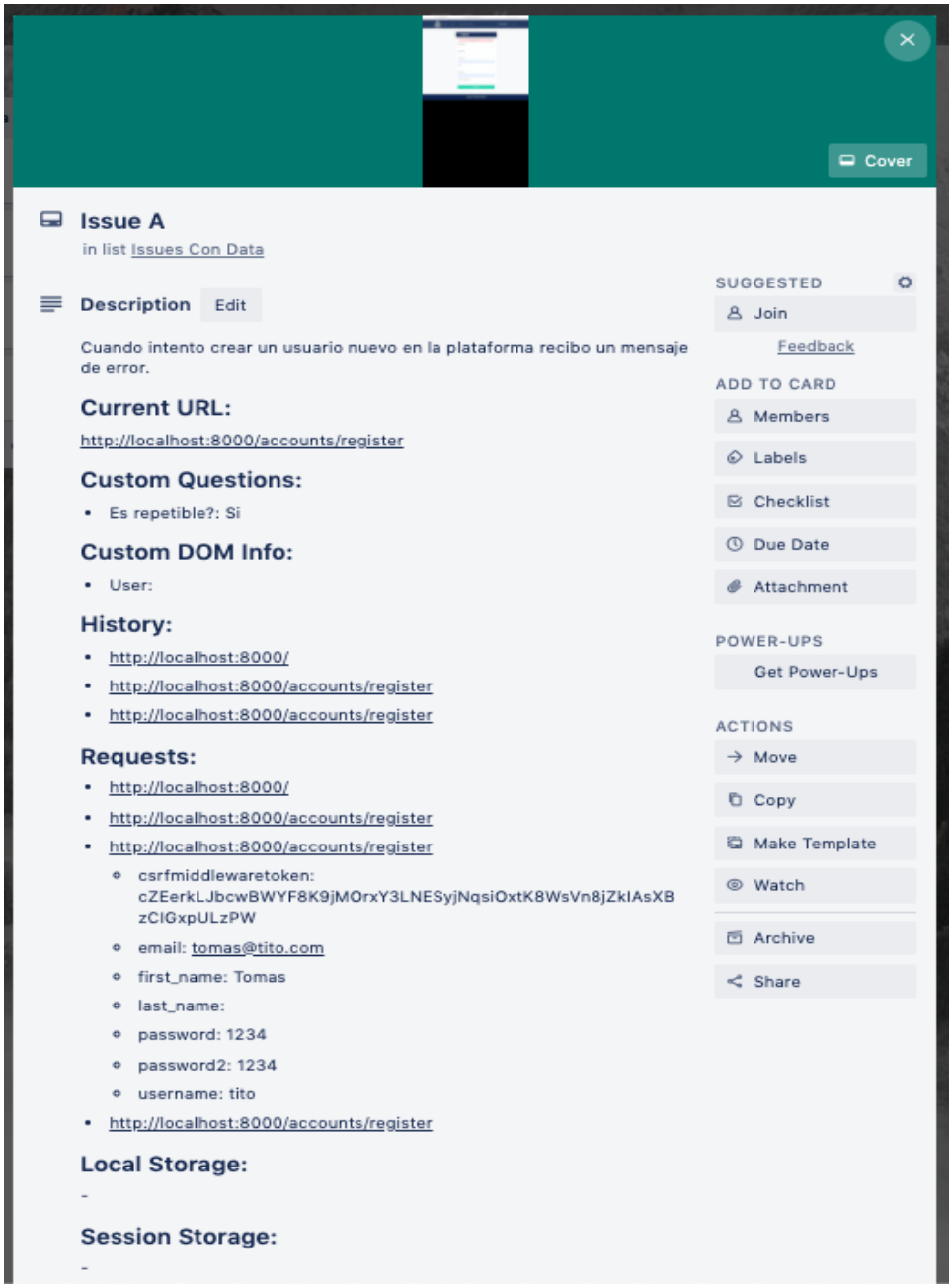

Figura 5.9: Card de Trello generada por la herramienta.

Entre la información que se puede observar en la imagen se encuentra:

- Descripción ingresada por el reportador.
- URL en la cual fue generado el issue.
- Preguntas custom configurables desde las opciones de la extensión.
- Información del DOM configurable desde las opciones de la extensión.
- Historial de navegación del reportador.
- Historial de requests al servidor con su data.
- Información almacenada en LocalStorage y SessionStorage.
- GIF construido con imágenes tomadas por cada click realizado.
- Capturas de pantalla realizadas por el reportador con la herramienta.

5.2.6. Asignación de los bugs

Grupo 1: se les asignaron los bugs A y B con el reporte completo generado por la aplicación y para los bugs C y D solamente la descripción generada por el usuario encargado de reportar los bugs.

Grupo 2: se les asignaron los bugs C y D con el reporte completo generado por la aplicación y para los bugs A y B solamente la descripción generada por el usuario encargado de reportar los bugs.

5.2.7. Métricas

Para las métricas realizamos un cuadro comparativo con el tiempo de resolución para cada grupo de cada bug, realizando la sumatoria por grupo del total de minutos que le tomó a cada desarrollador resolver cada bug.

Grupo	Issue A	Issue B	Issue C	Issue D
1	N minutos	N minutos	N minutos	N minutos
2	N minutos	N minutos	N minutos	N minutos

Reporte generado por la herramienta.

Reporte realizado sin la herramienta.

Figura 5.10: Tabla comparativa de tiempo de resolución por Issue.

Tiempo de resolución Grupo 1 para los bugs reportados con la herramienta contra el tiempo de resolución de los del Grupo 2 de los mismos bugs reportados sin la herramienta y viceversa.

Otra métrica a evaluar es de cuantos desarrolladores necesitaron de la respuesta predeterminada brindando más información.

Con Reporte	Issue A	Issue B	Issue C	Issue D
Si	N personas	N personas	N personas	N personas
No	N personas	N personas	N personas	N personas

Reporte generado por la herramienta.

Reporte realizado sin la herramienta.

Figura 5.11: Tabla comparativa de intercambio de mensajes por Issue.

5.2.8. Encuesta final

Al concluir la resolución de los bugs se le realizó una última encuesta al desarrollador para saber si la herramienta fue de ayuda, cual de las funcionalidades le fueron de utilidad para resolver el problema, si le gustaría contar con ella en su trabajo diario, etc.

1. En la escala del 1 al 5 (donde 1 es poco útil y 5 es muy útil), ¿qué tan útil te pareció el reporte completo hecho con la herramienta?

2. De la información del reporte, ¿qué datos te fueron de utilidad para identificar el error?

   a) GIF

   b) Historial de navegación

   c) Local storage

   d) Session storage

   e) POST y GET data

   f) Preguntas custom

   g) Elementos del DOM

   h) Otra

3. ¿Te gustaría contar con un reporte de este estilo en el día a día de tu trabajo?

4. La herramienta en lugar de facilitar la identificación de algún error/bug ¿te lo dificulto?

5. ¿Qué información no contemplada te gustaría ver en los reportes?

6. ¿Cambiarías algo en cuanto a la presentación del reporte?

5.3. Resultados

A continuación mostraremos los resultados del experimento explicado anteriormente, en donde tomamos el tiempo de identificación de cada desarrollador para cada uno de los issues. Para esto separamos a los desarrolladores en dos grupos balanceándolos en base a su experiencia laboral. Cada grupo recibió los mismos cuatro issues en donde:

Grupo 1:

- Issue A **(Reportado con la herramienta)**
- Issue B **(Reportado con la herramienta)**
- Issue C
- Issue D

Grupo 2:

- Issue A
- Issue B
- Issue C **(Reportado con la herramienta)**
- Issue D **(Reportado con la herramienta)**

Con esto buscamos comprobar si el reporte realizado con la herramienta sirve para mejorar tanto el tiempo de identificación de los errores como así también también la comunicación.

5.3.1. Conformación de los Grupos

Para la creación de los grupos, sus integrantes fueron seleccionados según sus años de experiencia como desarrolladores.

Grupo 1		Grupo 2	
Desarrollador	Experiencia(años)	Desarrollador	Experiencia(años)
Dev 1	5	Dev 11	7
Dev 2	6	Dev 12	6
Dev 3	7	Dev 13	3
Dev 4	4	Dev 14	4
Dev 5	7	Dev 15	8
Dev 6	4	Dev 16	2
Dev 7	7	Dev 17	5
Dev 8	2	Dev 18	4
Dev 9	3	Dev 19	5
Dev 10	3	Dev 20	3
Total	48	**Total**	47

Figura 5.12: Tabla de conformación de grupos en base a sus anós de experiencia.

En la tabla se puede apreciar como están balanceados los dos grupos con un promedio de años de experiencia por desarrollador de 4.8 para el Grupo 1 y 4.7 para el Grupo 2.

5.3.2. Resultados por Tiempo

En las siguientes tablas podemos ver el tiempo representado en minutos que le tomó a cada desarrollador identificar el motivo del error.

Grupo 1				
Dev	A	B	C	D
1	4	5	8	12
2	5	5	7	9
3	4	5	7	9
4	3	5	9	14
5	4	3	6	7
6	2	4	8	7
7	4	6	7	11
8	4	4	7	10
9	5	3	6	9
10	4	5	8	10
Total	**39**	**45**	**73**	**98**

Grupo 2				
Dev	A	B	C	D
11	12	11	6	4
12	4	10	3	7
13	5	9	3	6
14	11	10	3	5
15	10	9	4	7
16	11	11	6	7
17	15	8	3	5
18	10	10	4	7
19	9	11	4	6
20	10	9	4	7
Total	**97**	**98**	**40**	**61**

Figura 5.13: Tabla de resultados de experimento tiempo de identificación en minutos.

Issue A: Para el Grupo 1 el tiempo de identificación fue 59,79 % menor al Grupo 2.

Issue B: Para el Grupo 1 el tiempo de identificación fue 54,08 % menor al Grupo 2.

Issue C: Para el Grupo 2 el tiempo de identificación fue 45,21 % menor al Grupo 1.

Issue D: Para el Grupo 2 el tiempo de identificación fue 37,76 % menor al Grupo 1.

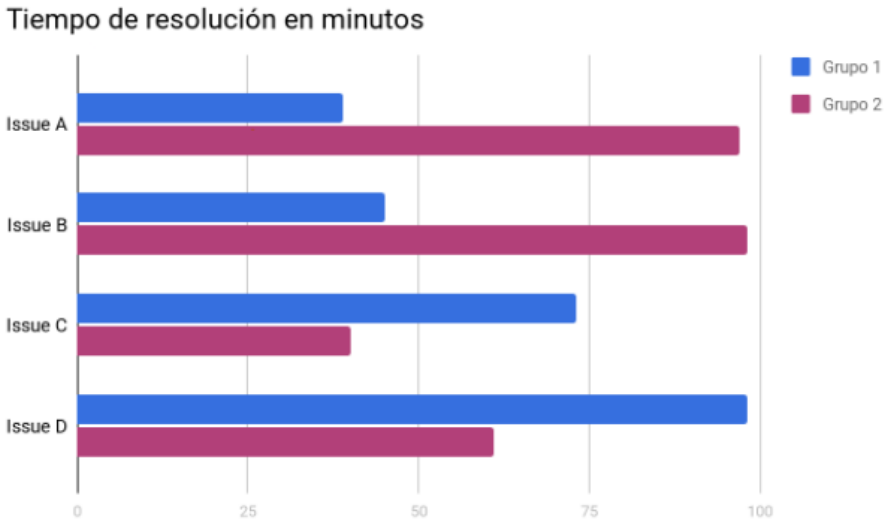

Figura 5.14: Gráfico de barras de resultados de experimento tiempo de identificación en minutos.

Si totalizamos los tiempos de identificación de los issues con reporte realizado con la herramienta, independientemente del grupo, y los comparamos contra la suma de los tiempos de identificación de los issues sin la herramienta nos da:

Tiempo de los issues con la herramienta: **185**
Tiempo de los issues sin la herramienta: **366**

Podemos observar que los issues reportados con la herramienta fueron identificados casi en la mitad de tiempo (49,45 % menor).

5.3.3. Resultados por intercambio de mensajes

En el caso de la ayuda predeterminada que se otorgaba al transcurrir seis minutos sin identificar los errores, los resultados fueron:

Grupo 1				
Dev	A	B	C	D
1	No	No	No	Si
2	No	No	No	Si
3	No	No	Si	Si
4	No	No	Si	Si
5	No	No	No	No
6	No	No	Si	Si
7	No	No	No	Si
8	No	No	Si	Si
9	No	No	No	Si
10	No	No	Si	Si
Total	**0**	**0**	**5**	**9**

Grupo 2				
Dev	A	B	C	D
11	Si	No	No	No
12	No	Si	No	Si
13	No	No	No	No
14	Si	Si	No	No
15	Si	Si	No	No
16	Si	Si	No	No
17	Si	Si	No	No
18	Si	Si	No	No
19	Si	Si	No	No
20	Si	Si	No	No
Total	**8**	**8**	**0**	**1**

Figura 5.15: Tabla de resultados de experimento en base a otorgamiento de ayudas predeterminadas.

Nota: solo se dio la ayuda a aquellos que tardaron más de 6 minutos en replicar el error. Existieron casos en donde pudieron replicar el error antes de los 6 minutos por lo tanto no aportaba nada a la hora de identificar el error el brindarles la ayuda predeterminada.

Issue A: El 80 % de los desarrolladores del Grupo 2 necesitó de la ayuda predeterminada, mientras que ninguno del Grupo 1 la necesitó.

Issue B: El 80 % de los desarrolladores del Grupo 2 necesitó de la ayuda predeterminada, mientras que ninguno del Grupo 1 la necesitó.

Issue C: El 50 % de los desarrolladores del Grupo 1 necesitó de la ayuda predeterminada, mientras que ninguno del Grupo 2 la necesitó.

Issue D: El 90 % de los desarrolladores del Grupo 1 necesitó de la ayuda predeterminada, mientras que solo el 10 % del Grupo 2 la necesitó.

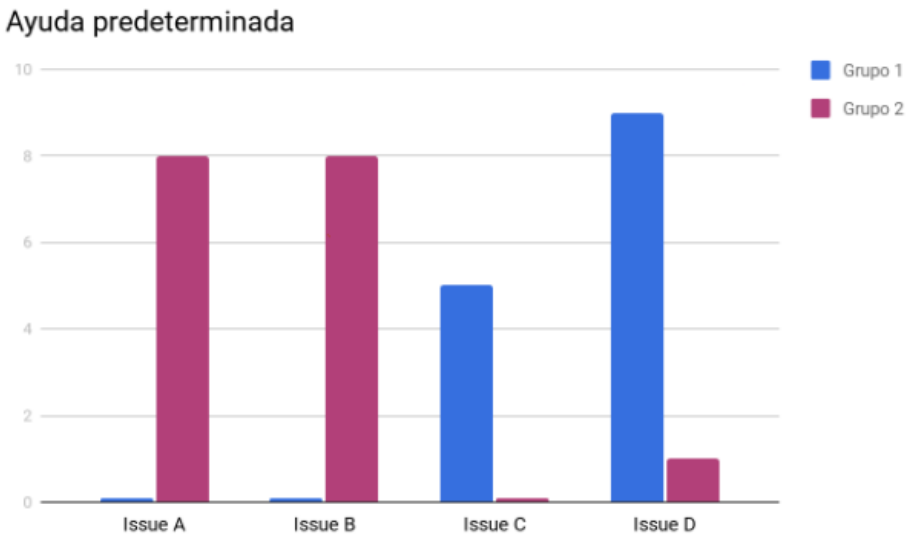

Figura 5.16: Gráfico de columnas comparativa de otorgamiento de ayudas predeterminadas.

Podemos observar a simple vista que a la hora de requerir la ayuda predeterminada, solo fue necesario en los casos donde no se brindaba el reporte completo a excepción de un solo caso (Issue D - Grupo 2).

5.3.4. Encuesta Inicial

Previo a la resolución/identificación de los Issues, a cada desarrollador se le hizo una serie de preguntas independientemente de su grupo en donde las respuestas fueron los siguientes:

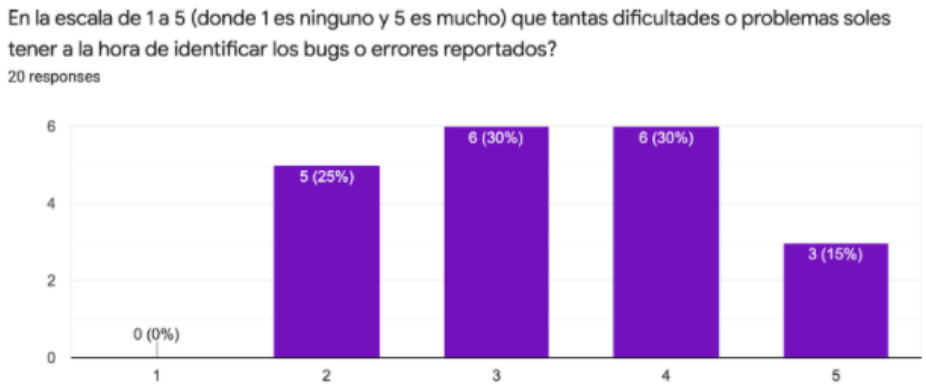

Figura 5.17: Gráfico de columnas de resultados de encuesta inicial primera pregunta.

Si bien un 25 % indicó que tiene pocos problemas a la hora de identificar errores, podemos observar que el 100 % tiene problemas y el 75 % considera que tiene problemas medios a graves.

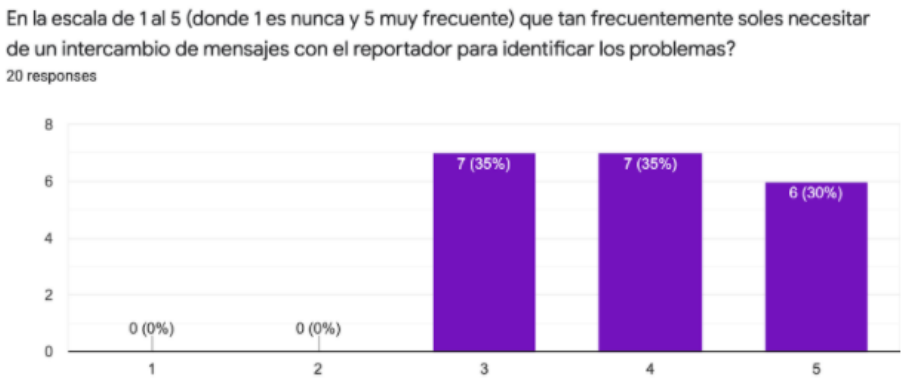

Figura 5.18: Gráfico de columnas de resultados de encuesta inicial segunda pregunta.

Cuando hablamos de la comunicación entre el desarrollador y el/los reportadores, podemos observar que el 100 % suele requerir, con mayor o menor frecuencia, de más información que la brindada inicialmente en el reporte

de error para identificar el problema.

Figura 5.19: Gráfico de torta de resultados de encuesta inicial tercera pregunta.

En cuanto al tipo de usuarios que realiza los reportes podemos observar que en general son usuarios del sistema los que realizan los reportes, seguido por testers totalizando estos un 90 % de los reportadores, siendo estos en general usuarios ajenos al ámbito de desarrollo.

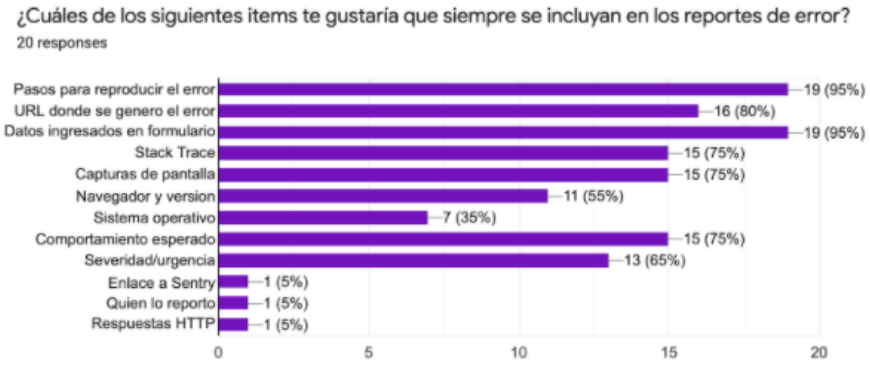

Figura 5.20: Gráfico de columnas de resultados de encuesta inicial cuarta pregunta.

Podemos ver a simple vista que hay mucha información que los desarrolladores quisieran que siempre se incluya en los reportes, pero que si toda

esta información fuera obligatoria para la creación de un reporte, sería un trabajo tedioso y hasta imposible en algunos casos para el reportador. Es posible pedirle al reportador que siempre describa cual es el comportamiento esperado o cual es la URL donde se generó el error, pero esto no asegura que lo haga en cada reporte o que lo haga correctamente.

5.3.5. Encuesta Final

Posteriormente a la identificación/resolución de los Issues, se les volvió a realizar otra encuesta a los desarrolladores con el fin de conocer su opinión sobre la herramienta desarrollada.

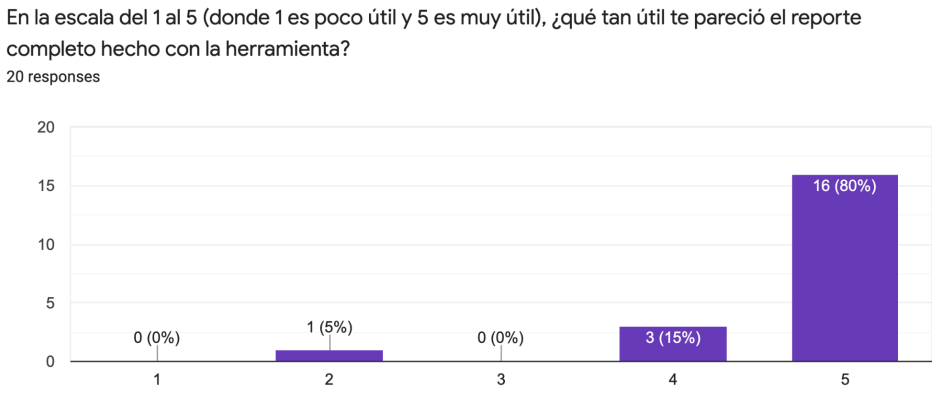

Figura 5.21: Gráfico de columnas de resultados de encuesta final primera pregunta.

Casi la totalidad de los desarrolladores encontraron muy útil el reporte realizado por la herramienta, el cual les permitió identificar los errores a partir de la información que brinda el mismo.

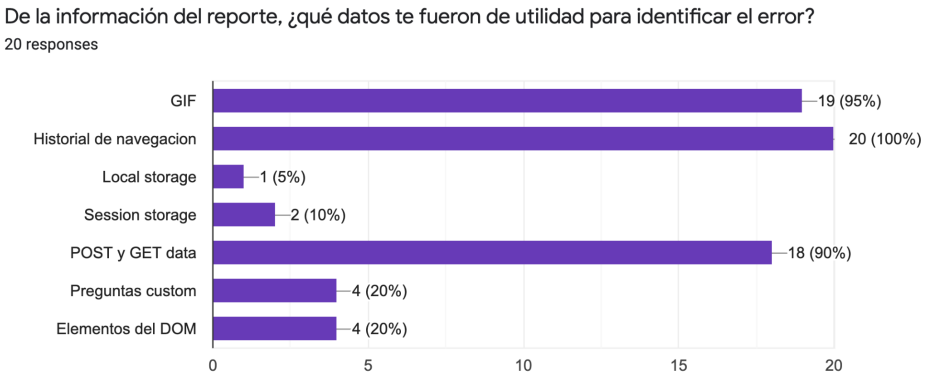

Figura 5.22: Gráfico de columnas de resultados de encuesta final segunda pregunta.

Si bien las respuestas están sujetas a los errores reportados, es casi unánime que el GIF, el historial de navegación y la información de los request (POST y GET) fueron de gran utilidad a la hora de identificar los errores reportados.

Figura 5.23: Gráfico de torta de resultados de encuesta final tercer pregunta.

Todos los desarrolladores preferirían que los reportes de errores se hicieran con una herramienta que capturara y mostrará información de la navegación como la reportada por la herramienta propuesta.

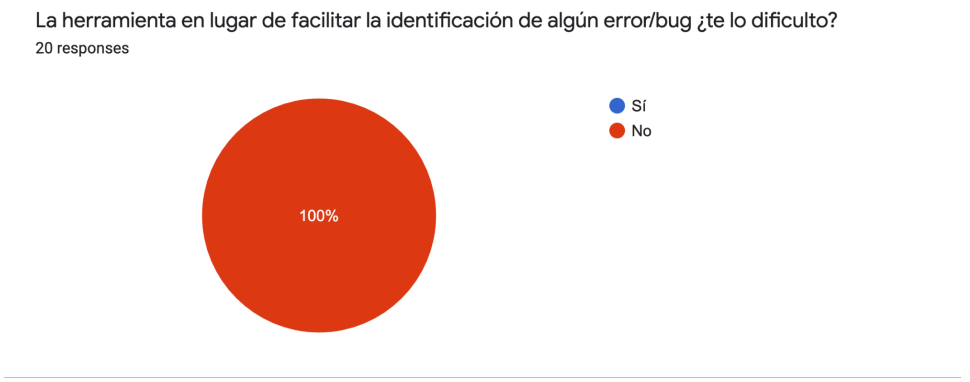

Figura 5.24: Gráfico de torta de resultados de encuesta final cuarta pregunta.

Si bien se podría creer que un exceso de información innecesaria puede confundir o dificultar la identificación de los errores, ningún desarrollador la encontró problemática.

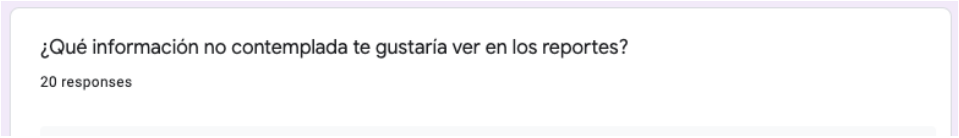

Figura 5.25: Quinta pregunta encuesta final.

- Console errors del navegador.
- Comportamiento deseado.
- Data de las responses con status 400/500.
- Creación de mockups.
- No, me pareció correcta para identificar los errores.
- Response del servidor.
- Logs de la consola js.
- No para estos ejemplos simples. Pero para casos más complejos sería bueno indicar navegador, versión, sistema operativo, etc.
- Errores de los assets que se cargan, errores de conexiones de websockets, información de indexeddb.

- Horario de los requests.
- El estado del paquete http - Si hubiese headers de autenticación - Aclarar si el request fue GET o POST o que verbo - Si se detecta un console.log u console.error hacer alguna mención del mensaje.
- No le agregaría nada.
- La información que me brindó me pareció muy completa, no sabría que agregarle.
- Historial detallado de requests y dom clickeado.
- Tipo de request y respuesta.
- En el caso de que se suba un archivo, poder contar con el mismo.
- Si el usuario está logueado o no.

Entre la información más pedida que podría ser agregada en un trabajo a futuro encontramos los logs de la consola, más información de los paquetes HTTP como su verbo, estado/código y fecha y hora.

Figura 5.26: Sexta pregunta encuesta final.

- No.
- Hacer que la descripción sea más llamativa para no perderla en el resto de la información.
- Mostrarme el verbo HTTP de las request hechas.
- No tener que scrollear el gif.
- No me pareció bien, capas el estilo pero es lo de menos.
- que se ofusque la contraseña ingresada.
- No se me ocurre nada en concreto. Pero supongo que se puede afinar el formato del mismo.

- Ofuscar datos sensibles en los contenidos de las requests o bien poner un warning grande grande.
- No.
- No, considero que debe ser útil mas que estetico.
- Me pareció muy bueno el reporte, no le cambiaría nada de momento.
- No debería loguear datos personales.
- No, me pareció correcta.
- Que el gif sea más lento.

Dentro de las sugerencias lo más destacable para mejorar sería el GIF, en algunos casos es muy rápido, mejorar la el formato del reporte para hacer más llamativa la descripción y ofuscar datos sensibles como tokens y contraseñas.

Capítulo 6

Conclusión

6.1. Conclusión general

Es muy común que los desarrolladores tengan problemas o dificultades para entender e identificar rápidamente los errores reportados en sistemas web llegando incluso a requerir de un intercambio de mensajes con el reportador solicitando más detalles e información. Además creen que hay cierta información que siempre debería incluirse en los reportes, la cual suele ser entregada luego de que el desarrollador la pide en lugar de ser incluida inicialmente en el reporte, como por ejemplo la URL donde se generó el error y la información ingresada en los formularios.

No alcanza con enseñarle al reportador qué información debe siempre ser incluida, porque sigue estando sujeta al error humano y no toda esta información es de fácil acceso para él [2]. Por lo tanto, contar con una herramienta de este estilo, quita esta responsabilidad sobre el reportador al recopilar la información automáticamente y procesarla para generar un mejor reporte de errores.

Como se puede apreciar en la sección Resultados del capítulo Evaluación, no solo es de utilidad para el reportador, sino también para los desarrolladores a la hora de lidiar con los errores, mejorando los tiempos de identificación en un 50 % y reduciendo en casi un 100 % el intercambio de mensajes entre estos y el reportador. Si los equipos de desarrollo invirtieran algo de tiempo en desarrollar o instalar y configurar una herramienta de este estilo, podrían mejorar drásticamente los tiempos no solo de resolución de errores sino también el de desarrollo.

6.2. Trabajo Futuro

Si bien la herramienta desarrollada cumple con su propósito y es efectiva, existen varios complementos y mejoras que se le pueden desarrollar, entre estos encontramos:

- Extender la información provista
- Mejorar la visualización de la información
- Integración con otros gestores de proyectos

6.2.1. Sugerencias de los desarrolladores

De las sugerencias realizadas por los veinte desarrolladores encuestados podemos destacar:

- **Verbo HTTP:** agregar una etiqueta en el reporte para cada request que indique el verbo HTTP utilizado (GET, POST, PUT, DELETE, etc.).
- **Console Errors:** capturar y listar de manera compacta y prolija los errores que surjan en la consola del navegador.
- **Comportamiento esperado:** agregar un campo obligatorio para el reportador donde deba explicar el comportamiento deseado.
- **Código de estado de respuesta HTTP:** estado de la solicitud (201 Created, 301 Moved Permanently, 400 Bad Request, 500 Internal Server Error, etc.).
- **Navegador:** nombre y versión del navegador.
- **Sistema Operativo:** nombre y versión del sistema operativo.
- **Archivos subidos:** en el caso de que se suba un archivo a través de un formulario poder contar con él o con información relevante del mismo (extensión, tamaño, nombre, etc.).
- **Assets:** proveer un listado con los nombres de los assets que fueron cargados.
- **Fecha y hora de los requests:** agregar junto al verbo la fecha y hora de los requests.
- **IndexedDB:** adjuntar al reporte información almacenada en IndexedDB.

6.2.2. Mejoras del reporte y la extensión

Además de las sugerencias de los desarrolladores la herramienta sería extendida con las siguientes funcionalidades:

- Los reportes podrían ser generados con un número automático de identificación para un mejor seguimiento de los mismos en el futuro dentro de la herramienta de gestión de proyectos.
- Posibilitar la edición de las capturas de pantalla para realizar anotaciones y dibujar recuadros y señalizaciones que permitan al reportador ser más explícito con lo que quiere mostrar con dicha captura.
- Para brindarle más utilidad a la extensión y que sirva no solo para reporte de errores, sino también para la creación de nuevos features en el proyecto, se agregaría una sección con las herramientas necesarias para la creación de mockups [1].
- Capturar y generar un archivo JSON con cada interacción del usuario con el navegador para una posterior ejecución del desarrollador en herramientas de automatización de pruebas como por ejemplo Selenium [2] [21].

6.2.3. Integración con gestores de proyecto

Actualmente la herramienta trabaja con dos de los gestores de proyecto más populares (Trello, Basecamp). Creando los adaptadores necesarios con la interfaz explicada en la sección Arquitectura del capítulo Desarrollo Propuesto podría darse soporte a otros gestores de proyectos del mercado tales como Jira, GitLab, etc.

También se podría extender la funcionalidad de la herramienta en base a las opciones que brindan comúnmente los gestores de proyecto como:

- Asignación de usuario
- Prioridad/urgencia de la tarea
- Deadline de la tarea

[1] Un mockup o maqueta es un diseño digital de una web y/o aplicación.
[2] Es un entorno de pruebas de software para aplicaciones basadas en la web.

Si bien no todos los gestores de proyecto brindan las mismas funcionalidades, en los casos en que estos no lo hagan pueden ser adaptados, por ejemplo en Basecamp 2 no es posible asignar una prioridad a una tarea pero podría ser agregada a la descripción de la misma.

Bibliografía

[1] *Breu, S., Premaj, R., Sillito, J., Zimmermann, T. (2009). Frequently asked questions in bug reports. University of Calgary.*

[2] *Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T. (2008, November). What makes a good bug report?. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 308-318).*

[3] *Zimmermann, T., Premraj, R., Sillito, J., Breu, S. (2009, May). Improving bug tracking systems. In 2009 31st International Conference on Software Engineering-Companion Volume (pp. 247-250). IEEE.*

[4] *Hooimeijer, P., Weimer, W. (2007, November). Modeling bug report quality. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (pp. 34-43).*

[5] *Just, S., Premraj, R., Zimmermann, T. (2008, September). Towards the next generation of bug tracking systems. In 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (pp. 82-85). IEEE.*

[6] *Davies, S., Roper, M. (2014, September). What's in a bug report?. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-10).*

[7] *Kolluri, A. B., Tameezuddin, K., Gudikandula, K. Effective Bug Tracking Systems: Theories and Implementation. IOSR Journal of Computer Engineering (IOSRJCE) ISSN, 2278-0661.*

[8] *Boehm, B., Basili, V. R. (2007). Software defect reduction top 10 list. Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research, 34(1), 75 .*

[9] *Kim, S., Whitehead Jr, E. J. (2006, May). How long did it take to fix bugs?. In Proceedings of the 2006 international workshop on Mining software repositories (pp. 173-174). .*

[10] *Soltani, M., Hermans, F., Bäck, T. (2020). The significance of bug report elements. Empirical Software Engineering, 1-40. ISO 690 .*

[11] *Baysal, O., Holmes, R., Godfrey, M. W. (2013, May). Situational awareness: personalizing issue tracking systems. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 1185-1188). ISO 690 .*

[12] *Raymond, E. (1999). The cathedral and the bazaar. Knowledge, Technology Policy, 12(3), 23-49 .*

[13] *Reis, C. R., de Mattos Fortes, R. P. (2002, February). An overview of the software engineering process and tools in the Mozilla project. In Proceedings of the Open Source Software Development Workshop (pp. 155-175) .*

[14] *Jesse Ruderman, Gervase Markham. (2020). Guía para escribir un Bug .*

[15] *Anvik, J., Hiew, L., Murphy, G. C. (2006, May). Who should fix this bug?. In Proceedings of the 28th international conference on Software engineering (pp. 361-370) .*

[16] *Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S. (2008, September). Duplicate bug reports considered harmful... really?. In 2008 IEEE International Conference on Software Maintenance (pp. 337-345). IEEE .*

[17] *Jalbert, N., Weimer, W. (2008, June). Automated duplicate detection for bug tracking systems. In 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN) (pp. 52-61). IEEE .*

[18] *Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., Zimmermann, T. (2007, October). Quality of bug reports in Eclipse. In Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange (pp. 21-25). .*

[19] *Singh, V. B., Chaturvedi, K. K. (2011). Bug tracking and reliability assessment system (btras). International Journal of Software Engineering and Its Applications, 5(4), 1-14. .*

[20] *What are extensions?, https://developer.chrome.com/extensions .*

[21] *Artzi, S., Kim, S., Ernst, M. D. (2008, July). Recrash: Making software failures reproducible by preserving object states. In European conference on object-oriented programming (pp. 542-565). Springer, Berlin, Heidelberg. .*