ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DEL BACKEND Y LA CONSOLA DE CONTROL DEL VIDEOJUEGO MULTIJUGADOR GO PROTEST

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE SOFTWARE

KEVIN DANIEL MORALES ESTRELLA

DIRECTOR: DR. RICHARD PAÚL RIVERA GUEVARA

DMQ, febrero 2022

CERTIFICACIONES

Yo, Kevin Daniel Morales Estrella declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Kevin Daniel Morales Estrella

kevin.morales05@epn.edu.ec

kmorales201314@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Kevin Daniel Morales Estrella, bajo mi supervisión.

Dr. Richard Paúl Rivera Guevara

DIRECTOR

richard.rivera01@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Kevin Daniel Morales Estrella

DEDICATORIA

Dedico este trabajo a mi esposa, quien ha sido mi soporte espiritual y emocional en los últimos 6 años de matrimonio, a mi padre Víctor quien siempre quiso verme triunfar y a mi hijo Leonel quien ha sido mi fuente de fortaleza y motivación principal para superarme.

AGRADECIMIENTO

Agradezco a Dios por darme la fortaleza para cambiar de profesión en un momento decisivo de mi vida, a mi madre por siempre estar ahí cuando la necesitaba, a mi abuelo Víctor quien ahora no está con nosotros, pero que fue la persona que me acogió en su hogar durante mucho tiempo y fue mi modelo paterno, a mi abuela Catalina por criarme y enseñarme la ley de Dios, a mi esposa Denisse por ser mi apoyo en todo el proceso que seguí para conseguir este galardón y a mis queridos profesores de la Escuela Politécnica Nacional quienes me motivaron a ser autodidacta y a amar la complejidad de la ciencia y la ingeniería.

ÍNDICE DE CONTENIDO

RESUMEN

Un videojuego es la parte visual de toda una industria que se dedica a crear productos para el entretenimiento. *Go Protest* es un juego de jugabilidad gratuita que genera ganancias y soporta sus costos de operación a través de la venta de consumibles digitales. El juego requiere de un sistema *ERP* para manejar el negocio de manera eficiente. Como parte preliminar del proyecto de emprendimiento se procedió a desarrollar el *backend* de todo el videojuego y una consola de administración para automatizar las operaciones de la empresa, las cuales no son visibles para el usuario final pero que permite que todo funcione adecuadamente.

Se ha desarrollado una *API REST* con Node.js y Apollo Server como servidor de GraphQL para interactuar de manera eficiente con MongoDB la cual se encuentra en un clúster de base de datos en la nube, garantizando la alta disponibilidad para los usuarios del juego y para los operadores de *Go Protest*. La consola de administración se ha desarrollado con React y con CSS puro. Para garantizar la calidad de código y la seguridad del *backend* como de la consola se ha trabajado con Synk y SonarQube para corregir las vulnerabilidades de seguridad y refactorizar el código repetitivo e ineficiente.

En el presente documento se muestra el procedimiento y resultados obtenidos del desarrollo del proyecto, así como el despliegue en producción del *backend* y de la consola de control. Con el objetivo de lograr flexibilidad y rapidez en el desarrollo se usó la metodología Scrum.

PALABRAS CLAVE: *ERP, API REST, APOLLO SERVER, GRAPHQL, MONGODB.*

ABSTRACT

A video game is the visual part of an entire industry that is dedicated to creating products for entertainment. Go Protest is a free to play game that generates profits and supports its operating costs through the sale of digital consumables. The game requires an ERP system to run the business efficiently. As a preliminary part of the entrepreneurship project, we proceeded to develop the backend of the entire video game and an administration console to automate the operations of the company, which are not visible to the end user, but which allows everything to function properly.

A REST API has been developed with Node.js and Apollo Server as GraphQL server to interact efficiently with MongoDB, which is in a database cluster in the cloud, guaranteeing high availability for game users and for Go Protest operators. The administration console has been developed with React and with pure CSS.

To guarantee the quality of the code and the security of the backend and the console, Synk and SonarQube were used to correct security vulnerabilities and refactor repetitive and inefficient code.

This document shows the procedure and results obtained from the development of the project, as well as the deployment in production of the backend and the control console. In order to achieve flexibility and speed in development, the Scrum methodology was used.

KEYWORDS: *ERP, REST API, APOLLO SERVER, GRAPHQL, MONGODB.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El proyecto es la fase preliminar del videojuego Go Protest. En este trabajo se muestra el diseño e implementación de una *API* basada en Node, con un servidor de Apollo y el lenguaje de consultad GraphQL. La base de datos utilizada fue MongoDB en un clúster en la nube. La *API* puede ser consumida por los clientes del videojuego en una fase posterior y en la consola de administración del videojuego, la cual es parte de este proyecto. La consola de administración es una aplicación creada con React que tiene el objetivo de mostrar la información de los jugadores del videojuego y administrar todos los asuntos relacionados al giro del negocio de *Go Protest*.

La consola permite agregar jugadores, supervisores y administradores, editar, eliminar usuarios o desactiva/activar cuentas, agregar consumibles, agregar nueva ropa o personajes del juego, realizar compras de consumibles o de monedas digitales mediante un botón de PayPal integrado. Además, permite conocer las compras que han realizado los jugadores.

La *API* ha sido sometida a pruebas de integración para verificar su correcto funcionamiento utilizando los componentes especializados de Apollo Server para realizar pruebas de código. Adicionalmente, para garantiza la calidad del código, la aplicación fue evaluada de manera constante con el servicio en la nube de SonarQube, con el objetivo de eliminar código repetitivo y aplicar refactorización. También, se analizó de manera constante durante el desarrollo en la aplicación Synk la cual detectó vulnerabilidades y brindó soluciones sugeridas para mitigar dichas vulnerabilidades.

Un videojuego no solo es el producto que los jugadores disfrutan sino es un producto que proviene de una empresa que lo desarrolla, brinda mantenimiento y soporte a todos los jugadores a nivel mundial. La administración de la empresa es ignorada siempre y cuando esta no genere problemas al producto visible por los clientes. Una empresa de base tecnológica necesita un sistema para controlar todos los aspectos secundarios y principales del producto digital que los usuarios consumen diariamente. Los operarios del videojuego necesitan bloquear o activar cuentas de jugadores si estos infligen la normativa del juego o agregar nuevos consumibles, ropa, personajes o *skins* especiales al videojuego de manera automatizada para facilitar las actualizaciones del videojuego.

En esta fase preliminar del desarrollo del videojuego *Go Protest* se desarrollará un *backend* robusto y una consola de control que permita controlar todos los aspectos empresariales del videojuego.

Este proyecto de titulación está enfocado en el desarrollo de una API que permitirá manejar toda la información que requiere el videojuego y la empresa que monitoreará y brindará servicio a los clientes del videojuego. La API permite el almacenamiento de la información de las compras realizadas en la consola y en el videojuego, la adquisición de consumibles, personajes, el nivel y el puntaje de los jugadores y más detalles que el modelo de negocio exija. La consola de control permitirá visualizar la información más relevante para los operarios del videojuego y unificará los procesos de la empresa para optimizar los recursos de la naciente Startup.

1.1 Objetivo general

Desarrollar el *backend* y la consola de control del videojuego multijugador *Go Protest* y consola de control del videojuego.

1.2 Objetivos específicos

1. OBJ 1. Levantar los requerimientos funcionales y no funcionales del sistema.
2. OBJ 2. Diseñar la arquitectura y base de datos del sistema.
3. OBJ 3. Verificación del sistema mediante pruebas unitarias del API REST y la consola de control, además pasar los proyectos por SonarQube y Synk para corregir vulnerabilidades de seguridad y código de baja calidad.
4. OBJ 4. Desplegar el API REST y la consola de control en un servidor web público.

1.3 Alcance

El emprendimiento *Go Protest* necesita una aplicación para administrar toda la empresa que operará el videojuego y un *backend* robusto para el funcionamiento de dicho juego. El presente proyecto propone el desarrollo de una *API REST* para permitir el funcionamiento de un videojuego online y una consola de control para administrar la empresa que desarrolla el videojuego. La *API* permitirá registrar usuarios según el rol, puede ser jugador, administrador, supervisor o superadministrador, también crear, editar o eliminar consumibles, ropa, skins especiales y personajes del videojuego.

Además, permitirá crear nuevas compras de consumibles y compras de monedas mediante PayPal. La *API* permitirá recolectar información muy valiosa, la cual puede desplegarse y manipularse desde la consola de control y así administrar la

empresa desde una sola interfaz. La consola de control, así como el *backend*, serán analizadas con SonarQube y Synk con el objetivo de aumentar la calidad de código y disminuir las vulnerabilidades de seguridad.

1.4 Marco Teórico

Para el desarrollo del proyecto se requiere una metodología ágil. Una metodología ágil es un conjunto de procedimientos cuyo objetivo es proporcionar en poco tiempo piezas pequeñas funcionales de sistemas de software para conseguir la satisfacción del cliente usando enfoques flexibles, trabajo en equipo con equipos pequeños autoorganizados de desarrolladores y representantes empresariales [2]. Con este conjunto de métodos se pretende cumplir con los objetivos propuestos.

Según Chapaval [3] el *Frontend* es la parte de un sitio web que interactúa con los usuarios, por eso decimos que está del lado del cliente. En esta parte del programa es en la que el usuario interactúa directamente, esta contiene elementos multimedia como animaciones, imágenes, texto, botones y su apariencia no solo tiene el objetivo de cautivar sino de brindar una experiencia de usuario agradable mientras el usuario satisface sus necesidades en el sistema.

El *Backend* es la parte que se conecta con la base de datos y el servidor que utiliza un sitio web, esta capa corre en el servidor [3]. La lógica de la aplicación entrelazada a la lógica del negocio, la seguridad de la información y configuraciones necesarias forman parte de lo que el usuario no ve, pero brinda la mayor cantidad de funcionalidades que los usuarios disfrutan.

Una *API REST* es un servicio que expone un conjunto de datos y funcionalidades para facilitar las interacciones entre los programas computacionales y permitir que estos intercambien información [4]. En sí, es la cara de un servicio web que está escuchando a las peticiones de los clientes.

2 METODOLOGÍA

La metodología de investigación utilizada en este trabajo es un estudio de caso debido a que se busca resolver un problema de la vida real [5] mediante el desarrollo de un programa informático. El contexto es limitado, debido a que busca solventar una necesidad de un emprendimiento en el área de los videojuegos. Los requerimientos por solventar ameritan investigación para escoger y luego obtener capacitación para implementar la tecnología más conveniente para usar en el sistema. Además, este trabajo es una solución pragmática, ya que presenta una solución de software para automatizar procesos de la empresa que controlará y monitoreará al videojuego Go Protest.

Se decidió ocupar la metodología Scrum para el desarrollo del proyecto. La cual brindará la flexibilidad necesaria para desarrollar el producto de software de manera rápida y eficiente. Esta metodología involucra al dueño del producto con el desarrollador para ir ajustando el proyecto a las necesidades y expectativas del inversionista. Su enfoque basado en la evidencia es importante ya que permite la entrega continua e incremental del proyecto, con el objetivo de evaluar el proyecto de manera temprana [6] y corregir errores. Los métodos y herramientas ayudarán a la correcta gestión y organización del proyecto.

2.1 Metodología de Desarrollo

Scrum es una metodología ágil iterativa incremental que utiliza *sprints* para desarrollar actividades propuestas en el proyecto. Los *sprints* tendrán una duración entre dos a tres semanas para construir el proyecto y entregar un producto funcional en cada revisión del proyecto. El contacto con el dueño del proyecto será clave para aumentar o eliminar tareas necesarias para obtener el software requerido.

Scrum tiene roles esenciales para lograr su meta principal, la agilidad. Para ello cada rol debe cumplirse con seriedad y compromiso para generar software extensible y de calidad. Esta metodología permite centrarse en el cliente y en sus abruptas necesidades de cambio. A continuación, se describen los roles:

Product Owner: es el responsable de crear y manejar el *Product Backlog* o pila de actividades del producto. El prioriza las actividades según el riesgo y la importancia que tiene según el modelo de negocio. Este rol se ha asignado al Ingeniero Richard Rivera debido a su experiencia y gran compromiso.

Scrum Master: esta persona es clave para la implementación de la metodología Scrum, esta persona es muy organizada y ayuda al resto del equipo a recordar sus asignaciones y coordina las reuniones de revisión y planificación del *sprint*. Este rol fue asignado al doctor Richard Rivera.

Development Team: el equipo de desarrollo es responsable de implementar tecnologías adecuadas para desarrollar el proyecto. Desempeñan las iteraciones con eficiencia y eficacia para alcanzar los objetivos de los *sprints*. Reportan cualquier inconveniente al líder de proyecto con el objetivo de solventarlo y cumplir las metas propuestas. En TABLA I se observa la asignación de roles del equipo S*crum*.

TABLA I - Roles y Asignaciones

Rol	Integrante
Product Owner	Ing. Richard Rivera Ph.D.
Scrum Master	Ing. Richard Rivera Ph.D.
Development Team	Sr. Kevin Morales

Artefactos

La documentación dentro del marco de trabajo Scrum se desarrolla de manera iterativa y no está cargada de documentos largos e innecesarios sino de documentos que aportan y agregan valor al trabajo realizado por el equipo de desarrollo.

Recopilación de Requerimientos

Los requerimientos son las funcionalidades solicitadas por el cliente del proyecto. Estas deben ser detalladas y concretas ya que inciden directamente en los resultados del proyecto. La calidad de los requerimientos levantados es proporcional a la calidad del producto final.

La recopilación de los requerimientos fue realizada con el dueño de la Startup del videojuego donde se detalló con claridad todas las necesidades que tiene la organización. Se realizó el análisis pertinente para organizar los requisitos según la prioridad y el riesgo en el proyecto. Con esto se logró un entendimiento cabal de lo que se requiere desarrollar.

Historias de Usuario

Una historia de usuario es un escrito detallado de un requisito desde el punto de vista del cliente o interesados del proyecto. Cada historia de usuario corresponde a un requerimiento del sistema a desarrollar. En la TABLA II se da a conocer un ejemplo de

una historia de usuario creada para el proyecto. Todas las historias de usuario se encuentran en el ANEXO II.

TABLA II - Historia de Usuario

HISTORIA DE USUARIO	
Identificador (ID): HU004	**Usuario:** Administrador
Nombre Historia: Recuperación de Contraseña para usuarios	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Kevin Morales	
Descripción: Generar un *resolver* para recuperar la contraseña de los usuarios en el backend al ingresar el correo electrónico del usuario.	
Observación: Usar el servicio de correo electrónico para recuperar la contraseña del usuario mediante el envío de un *password* nuevo al correo electrónico registrado del mismo.	

Product Backlog
El producto backlog es una lista de todos los elementos del proyecto de software que debe ser ordenado según la prioridad propuesta por el Product Owner. Este último es el encargado de gestionar este documento para alcanzar los objetivos propuestos. Este documento se encuentra en el Anexo 2.

Sprint Backlog
Es una lista de actividades por realizar de un sprint. En dicha lista se asigna un responsable por cada tarea a ejecutar durante un periodo de tiempo establecido. El cumplimiento de estas tareas es un entregable completo que satisface a cabalidad los requerimientos funcionales o no funcionales requeridos. El Sprint Backlog se lo puede visualizar en el Anexo 2.

2.2 Diseño de interfaces (mockups)

Herramienta utilizada para el diseño

Se utilizó Figma para diseñar la interfaz de la consola de control. Esta herramienta permite crear los mockups e incluso dotar de funcionalidad representativa para facilitar la compresión de los desarrolladores sobre la funcionalidad y relaciones entre páginas de la aplicación.

Sistema Web

En la Fig. 1 se muestra un ejemplo del mockup de la consola de control, el resto se pueden visualizar en el ANEXO II.

Fig. 1: Prototipo del panel de administración de Go Protest

Diseño de la Base de Datos

Node.js es una librería de *backend* que es capaz de trabajar con varias bases de datos. En este caso particular se ha utilizado MongoDB la cual es una base de datos no relacional. Para las consultas a la base de datos se utilizó GraphQL, un lenguaje de consulta de base de datos y la librería Mongoose para facilitar las operaciones en la base de datos. La librería Mongoose permite abstraer las consultas y hacer más eficiente el trabajo en la base de datos y mostrar un código más limpio y mantenible.

Herramienta utilizada para el diseño de la base de datos

Se utilizó Lucid Chart para modelar la base datos. Esta herramienta permitió modelar la base relacional para facilitar el desarrollo de los modelos de la API. En la Fig. 2 se muestra el modelo de base de datos desarrollado.

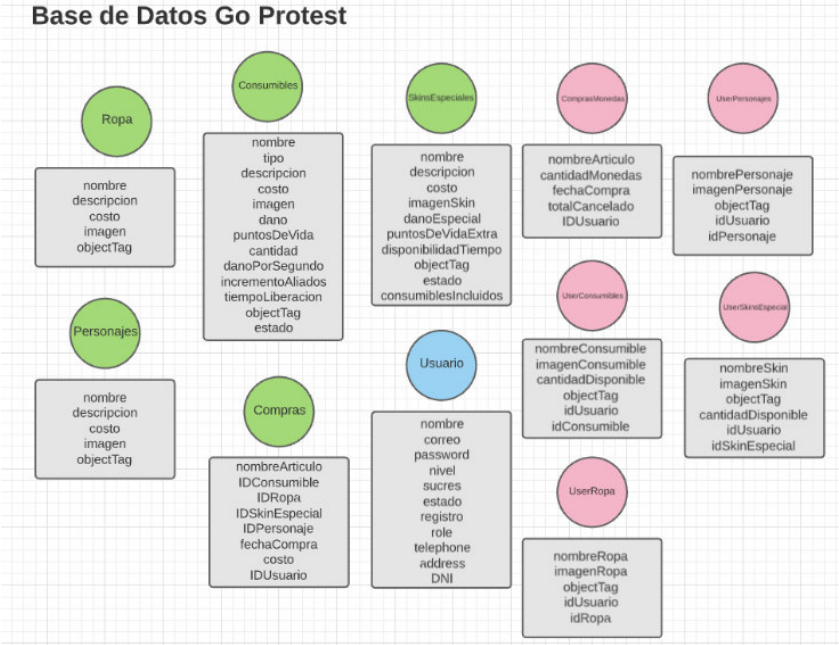


Fig. 2: Base de datos de Go Protest

2.3 Diseño de la Arquitectura

Patrón arquitectónico Modelo Vista Controlador (MVC)

Este proyecto tiene una arquitectura Modelo Vista Controlador (MVC) porque este patrón permite separar el sistema en capas. El sistema requiere el uso de varias interfaces de usuario [7] (en este proyecto solo de la consola de control, pero en el futuro de la UI del videojuego). El *backend* que se va a utilizar sirve para la consola de control como para el videojuego, por ello es necesario que sean capas independientes. A continuación, se describe brevemente el Modelo Vista-Controlador. En la Fig. 3 se muestra el diseño arquitectónico del proyecto.

Modelo: es la capa que trabaja con datos [7], en este caso particular se trabajará con una base de datos no relacional alojada en un clúster en la nube. Se escogió MongoDB y Mongoose como librería para abstraer las consultas a la base de datos.

Vista: es la capa que posee el código que permitirá renderizar los estados de la aplicación en HTML [7]. En este caso, el cliente que desplegará la información es React y facilitará la visualización de los datos provenientes de base de datos.

Controlador: contiene el código necesario para responder a las acciones que se solicitan en la aplicación [7]. Esta capa es un enlace entre las vistas y el modelo propuesta para este proyecto. El controlador en este caso es una API REST desarrollada con Node.js, la cual servirá de enlace entre la consola de control y la base de datos.

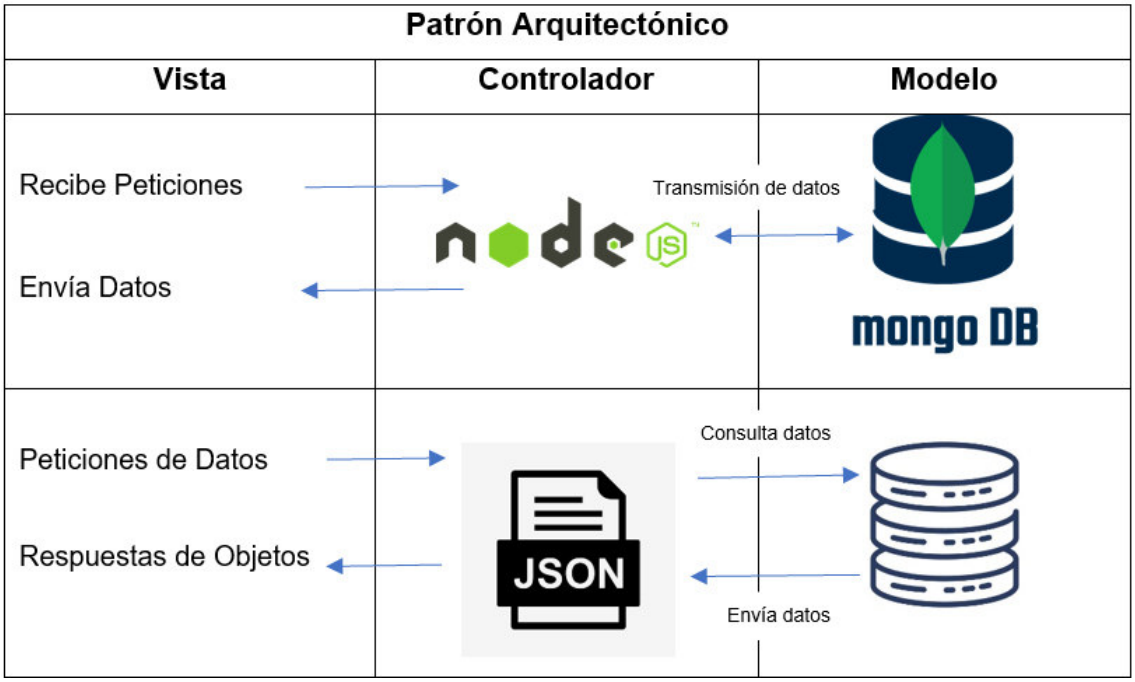


Fig. 3 Patrón Arquitectónico de la API REST

2.4 Herramientas de desarrollo

Las herramientas que se utilizaron fueron definidas en base a la arquitectura y a la compatibilidad entre ellas para funcionar eficientemente. En las tablas TABLA III, TABLA IV y TABLA V se muestran dichas herramientas utilizadas para el desarrollo del proyecto con su respectiva justificación.

API REST

TABLA III: Herramientas para el desarrollo de la API REST

Herramienta	Justificación
MongoDB	Se utilizará MongoDB como base de datos no relacional para el proyecto porque es sencilla de usar, es gratuita y porque es muy escalable [8]. *Go Protest* es un videojuego, por tanto, la base de datos de estar preparada para soportar miles de usuarios.
MongoDB Atlas	Es un servicio de base de datos en la nube, es gratuito y el costo luego de superar el límite permitido es según el uso. En este servicio es posible configurar clústeres de base de datos para garantizar la alta disponibilidad de la base de datos sin necesidad de instalar dependencias u otro tipo de software [9].
MongoDB Compass	MongoDB Compass es una GUI para MongoDB, permite conectar un servidor local o conectarse directamente al clúster de base de datos de MongoDB Atlas. El software es gratuito y permitirá verificar cambios en la base de datos durante el desarrollo y hacer monitoreo cuando el proyecto esté en producción. Permite realizar consultar y analizar patrones [10].

Apollo Server	Apollo Server es un servidor de código abierto diseñado para soportar GraphQL [11] y que es compatible con muchos clientes basados en varios lenguajes de programación. Es fácil de levantar y permite el trabajar con el lenguaje de consulta que usará el proyecto.
Node.js	Es un entorno de ejecución de JavaScript que permite correr programas desarrollados con dicho lenguaje [12]. Se usará JavaScript para desarrollar el *backend.* Es muy útil trabajar con esta tecnología porque tiene una comunidad muy grande que constantemente está realizando actualizaciones y generando paquetes compatibles con esta tecnología.
GraphQL	Se usará GraphQL como lenguaje de consultas debido a a que permite realizar consultas específicas y acceder a las anidaciones que tiene los objetos con una sola petición [13]. Esto permitirá reducir consultas en el servidor. El *backend* es para un juego de video que requerirá muchas consultas a la vez y se reducirá mucho la cuenta de los servicios en la nube al utilizar esta característica de este lenguaje.
Apollo GraphQL Studio	Es una plataforma en la nube que sirve para hacer pruebas de los *resolvers* de la API REST [14]. Es recomendable usar este programa debido a que documenta la API de manera automática y ayuda a convertir las consultas de GraphQL a la base de datos a consultas funcionales que pueden incrustarse en el cliente de GraphQL.
Heroku	Heroku es una plataforma de servicios en la nube. Se utilizará porque es fácil de usar y versátil [15]. Permite hacer despliegue continuo mediante la línea de comandos y soporta aplicaciones basadas en Node.js.

CONSOLA DE CONTROL

TABLA IV: Herramientas para el desarrollo de la Consola de Control

Herramienta	Justificación
React.js	Es una librería de código abierto para manejar el DOM de una página web. Se utilizó esta librería porque la curva de aprendizaje no es pronunciada y porque permite desarrollar aplicaciones de una sola página. Además, se usó esta librería debido a que "permite que las vistas se asocien con los datos, de modo que, si cambian los datos, también cambian las vistas" [16].
GraphQL Package	Es una dependencia que permite incrustar consultas de GraphQL en un entorno basado en Node.js.
MongoDB Package	Es una dependencia que permite configurar una base de datos de Mongo en un proyecto de Node.js.
Apollo Client	Apollo Client es una librería de manejo de estado para JavaScript que permite manejar datos con GraphQL [17]. Además, permite manejar la *cache* y modificar automáticamente la UI. Fue creada para usar con React.

Firebase	Esta es una plataforma para desarrollo de aplicaciones web [18]. En este caso se utilizó para hospedar la consola de control mediante un despliegue sencillo de la aplicación. Tiene un servicio gratuito muy bueno y solo se debe pagar por el dominio personalizado que se desee adjuntar al proyecto.

TABLA V: Herramientas para el Proyecto en General

Herramienta	Justificación
NPM	Se utilizará npm para instalar los paquetes necesarios para las aplicaciones a desarrollar. Ambos proyectos son con JavaScript entonces este administrador de paquetes permitirá gestionar las dependencias que se vayan a utilizar mediante la línea de comandos.
GitHub	Se utilizará este controlador de versiones hasta la fase de pruebas de las aplicaciones. Es gratuito y permite ver el avance y las contribuciones al proyecto.
ZenHub	Se manejará esta extensión de GitHub para gestionar el proyecto. ZenHub fue creado para manejar proyectos que usan la metodología ágil Scrum. Esta extensión permite configurar fechas, manejar problemas, *bugs* [19] y asignar prioridades a las actividades.
SonarQube	Es una plataforma para evaluar código fuente mediante análisis estático para obtener métricas que pueden ayudar a mejorar la calidad de código de un programa [20].
Synk	Es un servicio en la nube que analiza vulnerabilidades en proyectos de software. Si la vulnerabilidad está en su base de datos proporciona una solución e incluso la puede integrar al repositorio como una rama, para luego hacer un *merge* y eliminar la vulnerabilidad de seguridad. Puede emitir reportes cada vez que se realice integración continua [21]. Es decir, permite analizar las vulnerabilidades cada vez que el desarrollador realice un *commit.*
GitLab	GitLab se utilizará durante la fase de análisis de vulnerabilidades de seguridad, testeo automatizado y durante el análisis de calidad con SonarQube.
Visual Studio	Se utilizará como editor de texto ya que es gratuito y permite descargar extensiones para trabajar con proyectos con un lenguaje o *framework* determinado.

3 RESULTADOS

Después de completar los *sprints* planificados se obtienen los siguientes resultados:

3.1 Sprint 0. Configuración del ambiente de desarrollo

El sprint 0 corresponde a todas las actividades relacionadas a la configuración del entorno de desarrollo para crear todo el software necesario para satisfacer los requerimientos establecidos. En este Sprint se desarrollan las siguientes tareas:

- Instalación de Node y de las extensiones para Visual Studio.
- Creación de los repositorios en GitHub y GitLab.
- Instalación de Heroku CLI.
- Modelar la base de datos.

Instalación de Node y de las extensiones para Visual Studio

El *backend* y el *frontend* se desarrollarán con JavaScript. Por ello es necesario la descarga de nodejs para trabajar con npm y React. Las extensiones necesarias son JavaScript ESLint, JavaScript *snippets*, React Snippets, GraphQL *snippets* y Prettier para mejorar la visualización del código.

La instalación de Nodejs, fue exitosa, se instaló la versión 16.13.1 y npm, la versión 7.17.0. Con estos paquetes es posible trabajar con Nodejs y React. En la Fig. 4 se muestra las versiones instaladas.

Fig. 4 : Verificación de las instalaciones

Se logró instalar las extensiones de Visual Studio Code necesarias para trabajar con eficiencia con Node y React como se muestra en la Fig. 5.

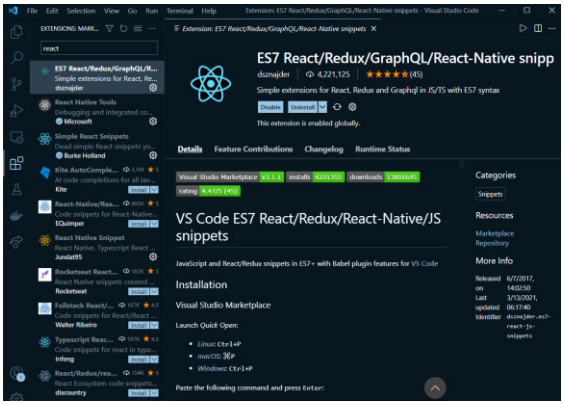


Fig. 5: Instalación de extensiones de Visual Studio

Creación de los repositorios en GitHub y GitLab.

Se crearon los repositorios, uno para la consola y otro para el backend en GitHub. El repositorio del backend tendrá un repositorio adicional para producción en GitLab. En el cual se automatizará los test y se analizará la calidad del código.

Se crearon dos repositorios en GitHub para la consola de control y uno para el backend, en la Fig. 6 se muestra el repositorio de la consola de control. Y un repositorio en GitLab para el BackEnd en el cual se va a trabajar con GitLab CI como se muestra en la Fig. 7 .

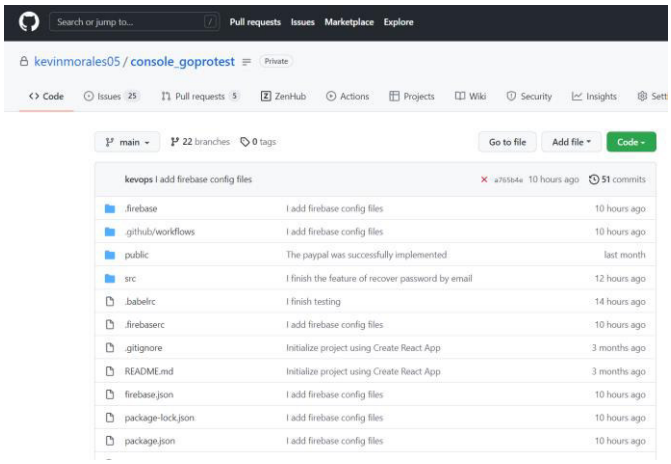


Fig. 6: Repositorio de la consola de control

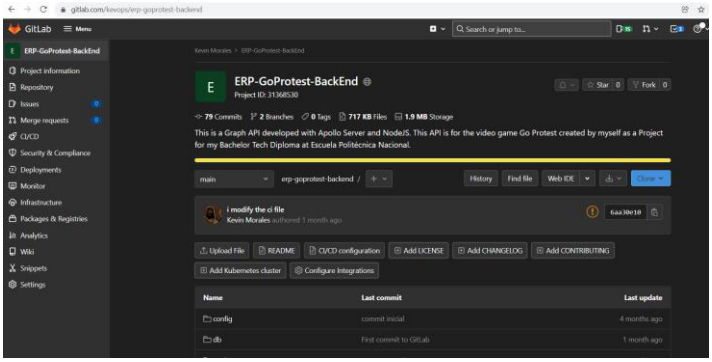


Fig. 7: Repositorio del backend en GitLab

Instalación de Heroku CLI

Se instaló Heroku CLI con el objetivo de subir el código del backend a la nube de Heroku para producción y realizar incrementos continuos del backend de manera sencilla.

Modelar la base de datos

El sistema requiere una base de datos no relacional la cual se desarrollará con MongoDB. Por tanto, se utilizó LucidChart para modelar la base de datos que se utilizará para todo el sistema del videojuego. La base de datos se implementó en MongoDB Atlas. Con la ayuda de MongoDB Compass es posible ver las colecciones creadas en la base de datos.

3.2 Sprint 1. CRUDS para Usuarios, Personajes, Ropa y Consumibles

El sprint 1 corresponde a todas las actividades relacionadas creación de una API GRAPHQL y la creación de los modelos de Usuarios, Personajes, Ropa, Compras, Compras de Monedas, Consumibles y Consumibles de Usuarios. Para luego crear los *resolvers* de cada uno de estos modelos, los cuales realizarán las actividades de Creación, Lectura, Actualización y Eliminación de información. En este Sprint se desarrollan las siguientes tareas:

- Creación del Proyecto de Node.
- Instalación de dependencias necesarias para levantar la API.
- Creación del Servidor.
- Creación de la Base de Datos y conexión al servidor.

- Creación de los Modelos, Tipos y *Resolvers*.
- Creación de los *resolvers* para operaciones CRUD.
- Autenticación de Usuarios y recuperación de contraseña.
- Creación de los modelos de Consumibles de Usuario
- Creación de los tipos.
- Creación de los resolvers para operaciones CRUD.

Creación del Proyecto de Node

Se creó el proyecto de Node con éxito, se crearon las carpetas principales para estructurar el proyecto. En package.json se crearon tres scripts: start, dev y test. Para iniciar el proyecto en producción, iniciar el proyecto en ambiente de desarrollo y test para realizar las pruebas. En la Fig. 8 se aprecia la estructuración de las carpetas del proyecto.

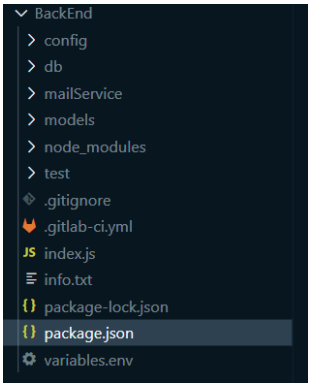


Fig. 8: Creación de proyecto de Node y estructura de carpetas.

Instalación de dependencias necesarias para levantar la API.

Se instalan dependencias para la API REST, el servidor de Apollo, GraphQL para el lenguaje de consulta, Mailslurp para trabajar con el servidor de correo para la API, jsonwebtoken para implementar tokens, dotenv para trabajar con variables de entorno, mongoose para trabajar con abstracciones de base de datos, bcrypt para encriptar los passwords antes de grabar en la base de datos. Como dependencias de desarrollo: nodemon para correr el servidor y realizar cambios en tiempo de ejecución, y Jest para realizar las pruebas unitarias y de integración. En la Fig. 9 se muestran las dependencias instaladas.

Fig. 9: Dependencias instaladas en el archivo package.json

Creación del Servidor

Se crea el servidor en el archivo index.js instanciando un objeto de Apollo Server. En el objeto del servidor se ingresaron los *resolvers*, los tipos y un contexto con token y palabra secreta como argumentos. El servidor escucha en el puerto 4000 para desarrollo local y en el puerto que se asigne en las variables de entorno. En la Fig. 10 se muestra la implementación del servidor.

Fig. 10: Servidor del proyecto

Creación de la Base de Datos y conexión al servidor

Se crea la base de datos en un clúster de MongoDB Atlas para asegurar la alta disponibilidad de los datos. En la Fig. 11 se muestran las colecciones de la base de datos Go Protest. El servicio de esta plataforma realiza réplicas de manera automática para permitir el escalado y la disponibilidad de la base de datos.

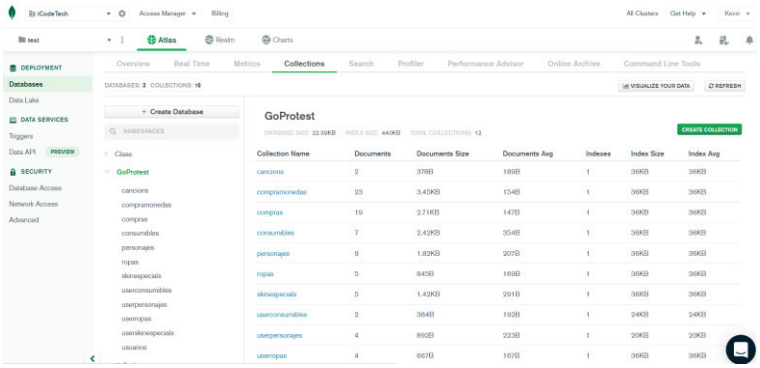


Fig. 11: Base de datos en MongoDB Atlas

Para conectar la base de datos, es necesario obtener el enlace de producción en MongoDB Atlas. En la figura 13 se muestra el enlace de conexión de la base de datos. El cual se obtiene al hacer *click* en la opción de *connect* del clúster de base de datos de la consola de control de MongoDB Atlas.

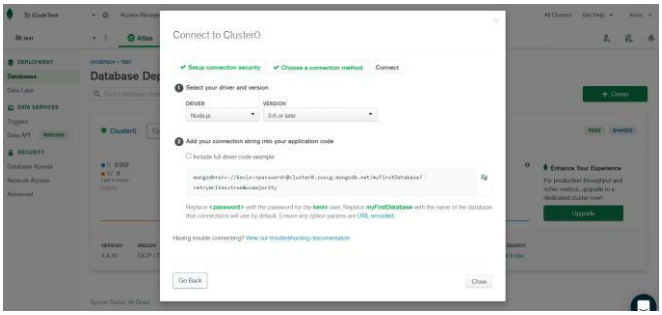


Fig. 12: Base de datos en MongoDB Atlas

Luego de obtener este enlace, es necesario agregar el *password* y escoger la base de datos deseada para luego usar ese enlace en la conexión de la API REST. Luego de realizar ese procedimiento, se colocó ese link en el archivo .env para luego colocarlo en la conexión del servidor. En la Fig. 12, se muestra la importación y llamado de la función conectarDB en index.js. la función conectarDB usa la variable de entorno DB_MONGO la cual tiene el enlace obtenido previo a este paso. Con esto se logra conectar la base de datos de manera exitosa.

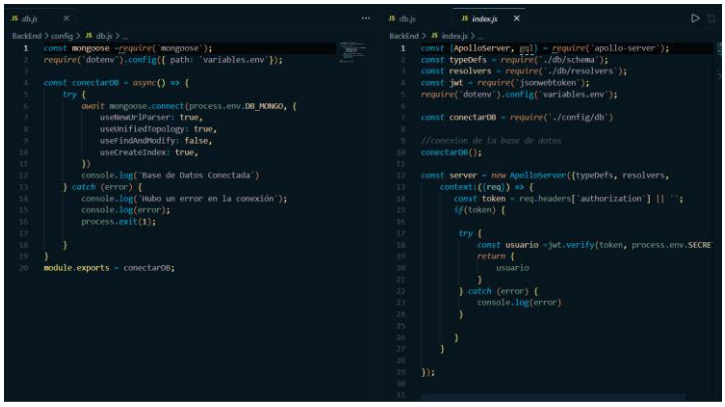


Fig. 13: Conexión de la base de datos

Creación de los Modelos, Tipos y Resolvers

Se crea archivos separados para definir los modelos, tipos y *resolvers*. Los modelos representan a las colecciones de la base de datos. Se definieron los tipos de cada parámetro en los modelos. Los tipos establen el tipo de dato que debe ingresar en el modelo. Dentro del fichero db, se crea el archivo resolvers.js, en el cual se separan los resolvers de tipo *query* (consulta) y los de tipo *mutation* (consultas que alteran la base de datos). En el mismo fichero se crea el archivo schema.js en el cual se redactaron los tipos de dato que se ocuparan en toda la API REST. En la Fig. 14 se aprecia la estructura básica fundamental de los *resolvers*, los cuales son los servicios que tiene la API y que cumplen la función de controladores, es decir manipulan o consultan la base de datos cuando son llamados.

Fig. 14: Estructura de los resolvers

En la Fig. 15 se puede apreciar el tipo Usuario, el cual tiene sus atributos que lo componen con una manifestación del tipo de dato de cada uno de ellos. Con esto se evita confusión por la falta de tipado que tiene JavaScript.

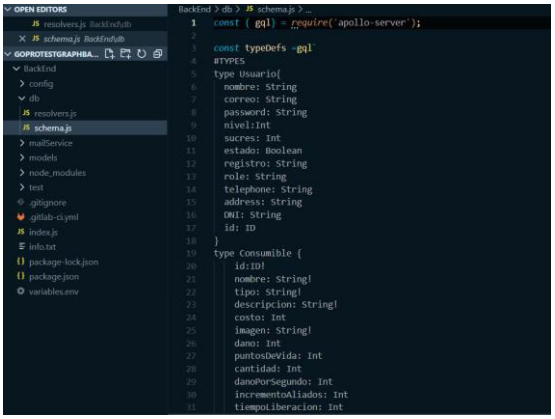


Fig. 15: Tipos del proyecto en schema.js, tipo Usuario

Creación de los *resolvers* para operaciones CRUD.

Previo a la creación de los *resolver*, se crean los modelos de Usuarios, Personajes, Ropa y Consumibles. Un modelo en este tipo de API, sirve para declarar el tipo de dato de cada atributo, también si el atributo es obligatorio o si el campo se llena por defecto. Además, es posible configurar detalles previos al ingreso de la información en la base de datos. En la Fig. 16 se aprecia el modelo Usuario, el cual es una instancia de *Schema* de la librería mongoose la cual facilita la creación de modelos a partir de un json. El modelo Usuario tiene varios atributos con tipos de dato definidos y con reglas para su llenado. Por ejemplo, el campo "*unique*" (único) en correo, establece que el correo es único para cada usuario y con esto evita duplicados.

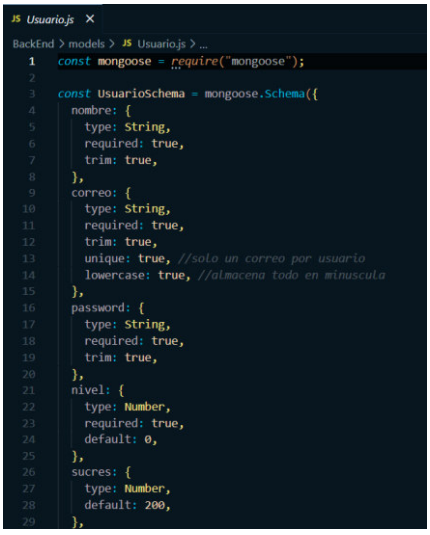


Fig. 16: Modelo de Usuario

Con los modelos listos, se crean los resolver requeridos para completar las operaciones CRUD. En la figura 18 se muestra un resolver de tipo *query*, el cual permite obtener información de los personajes del videojuego. El *query*

obtenerPersonajes es una función asíncrona que ocupa la función. *find()* de la librería moongose, la cual trae todos los datos del modelo Personaje.

```
obtenerPersonajes: async (_, { input }, ctx) => {
  const personajes = await Personaje.find();
  return personajes;
},
```

Fig. 17: Resolver para obtener Personajes

Autenticación de Usuarios y recuperación de contraseña

Se implementaron *resolvers* especiales para implementar la autenticación y la recuperación de contraseña. Lo que tenían en común era el cifrado del *password*, el cual se realizó con jwt. La autenticación requiere la creación de un token para devolver al cliente y la recuperación de contraseña requirió la implementación y configuración de Mailslurp. Para ello se ha creado una cuenta y una API Key para el backend en la página oficial de MailSlurp para usar el servicio. Para conectar el servicio y permitir el envío de una nueva contraseña por correo se ha creado un inbox id y una función para crear una contraseña nueva.

En la Fig. 18 se muestra el panel de control de MailSlurp con el *inbox* que se ha desarrollado y en la Fig. 19 se puede ver la implementación del servicio en el código de la API. En la Fig. 20 se muestra la implementación del servicio de autenticación y se muestra la generación del token creado con la dependencia *JsonWebToken* instalada previamente. El código permite realizar una validación de la existencia del Usuario que desea autenticarse y luego la validación de la contraseña.

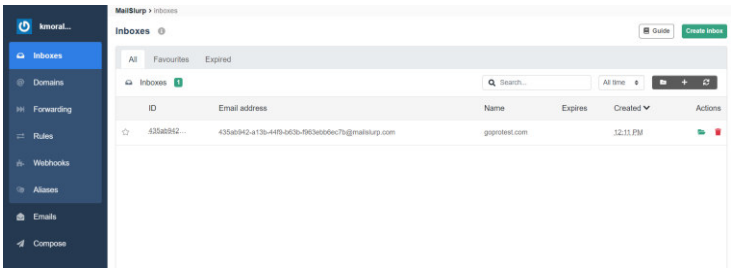


Fig. 18: Panel de Control de MailSlurp

Fig. 19: Implementación de MailSlurp en el resolver para recuperar contraseña

Fig. 20: Resolver para autenticar usuario

Creación de los modelos de Consumibles de Usuario

Se han creado los modelos requeridos tomando en cuenta los requerimientos del modelo de negocio. Los modelos de consumibles de usuario han requerido la adición del id del usuario que los adquirió como se muestra en la Fig. 21. Además, se ha aprovechado el potencial de las bases de datos no relacionales al crear un solo consumible con varios atributos para permitir un comportamiento diferente según el tipo de consumible. Por ejemplo, en la Fig. 22 se muestra el esquema de Consumible. El daño y los puntos de vida nos son obligatorios, porque se puede crear un consumible de tipo arma que baja puntos de vida como se puede crear un consumible de tipo salud el cual incrementa puntos de vida.

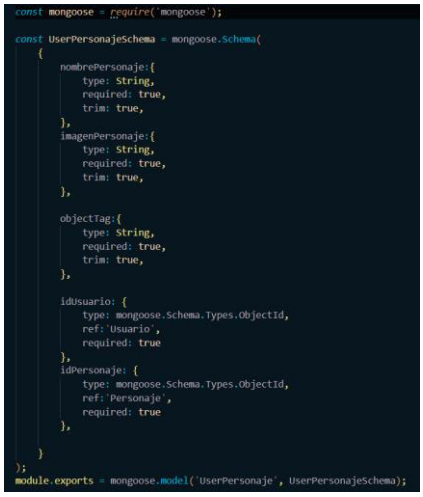


Fig. 21: Modelo de Personajes de Usuario

Fig. 22: Modelo de Consumible

Creación de los tipos

En schema.js se han creado los tipos de los modelos a usar. En la Fig. 23 se muestran los tipos creados marcando con un signo de admiración aquellos campos que son obligatorios.

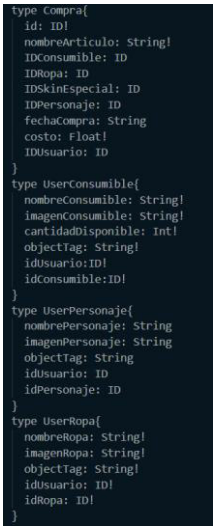


Fig. 23: Tipos de los modelos desarrollados

Creación de los *resolvers* para operaciones CRUD

Se crearon los *resolvers* para realizar todas las operaciones CRUD de los modelos requeridos en el sprint. En la Fig. 24 se muestra los *mutations* para registrar la adquisición de ropa para un usuario, actualizar una adquisición y cancelar una adquisición de ropa por parte de un usuario.

Fig. 24: CRUDS del modelo Ropa de Usuario

3.3 Sprint 2. Maqueta de Consola de Control y Menú de la Consola de Control.

El sprint 2 corresponde a todas las actividades relacionadas con la maquetación de la Consola de Control usando Figma como herramienta principal para el modelado y el desarrollo de las actividades relacionadas con la creación del menú de la consola de control. Se inicia el proyecto de React usando create-react-app y se instalan las

dependencias necesarias para el proyecto. En este Sprint se desarrollan las siguientes tareas:

- Tarea 1: Creación de los mockups.
- Tarea 2: Creación del proyecto de React e instalación de dependencias.
- Tarea 3: Creación del Menú de la consola de Control.

Creación de los mockups

Usando la herramienta Figma se desarrollaron los mockups necesarios para brindar la funcionalidad de la consola de control. Los mockups son de alta fidelidad ya que cumplen con los colores y tipografía del videojuego. En la Fig. 25 y en la Fig. 26 se muestran ejemplos de las pantallas a desarrollar.

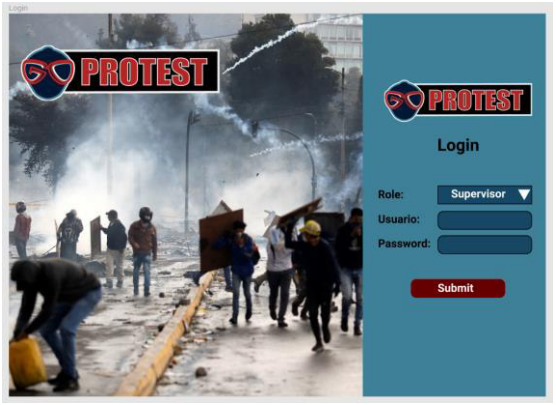


Fig. 25: Mockup de la Pantalla de Inicio de Sesión

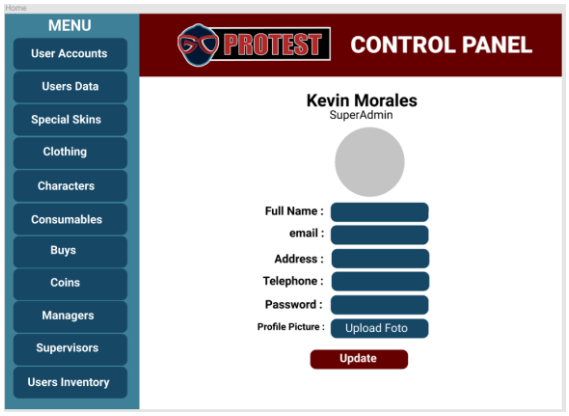


Fig. 26: Mockup de la Pantalla Home del Panel de Control

Creación del proyecto de React e instalación de dependencias

Al iniciar el proyecto de React y se desarrollaron carpetas para alojar a los componentes y páginas necesarias para el desarrollo del proyecto. En la Fig. 27 se

muestran las carpetas creadas en la carpeta *src*. En la Fig. 28 se muestra el archivo package.json en el que se muestran todas las dependencias instaladas. Cabe recalcar que se utilizará React Router para manejar las rutas de las páginas en el contexto de una aplicación de una sola página.

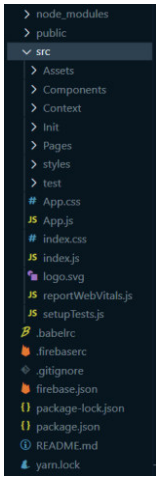


Fig. 27: Carpetas del proyecto de React

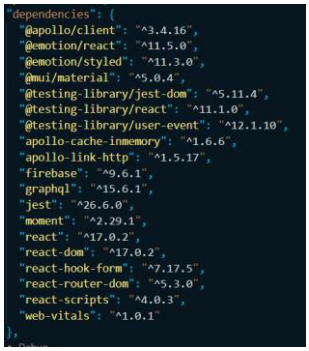


Fig. 28: Dependencias instaladas en el proyecto.

Creación del Menú de la consola de Control.

El menú se ha desarrollado y se ha implementado la funcionalidad para cambiar las páginas con los componentes Link de la librería React Router. En la Fig. 29 se muestra la implementación en de la consola en el servidor local. Además, en la Fig. 30 se muestra la configuración para renderizar los componentes Links en base al rol del usuario para cumplir con los requisitos de seguridad establecidos.

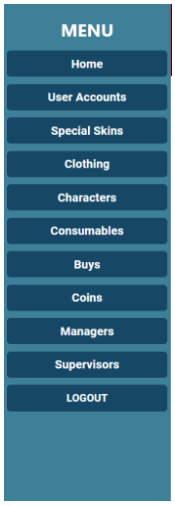


Fig. 29: Menú de la consola de Control.

```
<MenuList
  style={{ margin: "0 auto", display: "flex", flexDirection: "column" }}
>
  {routes
    .filter((route) => route.role.indexOf(userData.user.role) !== -1)
    .map((route) => {
      return (
        <MenuItem
          key={route.id}
          style={{
            backgroundColor: "#164866",
            margin: "5px",
            borderRadius: "5px",
            display: "flex",
            justifyContent: "center",
          }}
        >
          <Link
            to={route.route}
            style={{ textDecoration: "none", alignContent: "center" }}
          >
            <b style={{ color: "white", textAlign: "center" }}>
              {route.name}
            </b>
          </Link>
        </MenuItem>
      );
    }))
```

Fig. 30: Código para renderizar los Links de acuerdo al rol.

En la Fig. 31 se muestra una sección del archivo routes.js en el cual muestran los roles que pueden tener acceso a los componentes establecidos. Por ejemplo, para acceder a *User Accounts* es necesario tener los roles de superadmin, supervisor o manager.

```
const routes = [
  {
    id: 0,
    name: "Home",
    route: "/home",
    component:"HomeScreen",
    role:["superadmin","supervisor", "manager"]

  },
  {
    id: 1,
    name: "User Accounts",
    route: "/users",
    component:"UserAccounts",
    role:["superadmin","supervisor", "manager"]
  },

  {
    id: 3,
    name: "Special Skins",
    route: "/special",
    component:"SpecialSkins",
    role:["superadmin","supervisor"]
  },
```

Fig. 31: Rutas y sus permisos.

3.4 Sprint 3. Pruebas unitarias del Backend, Login y recuperación de Contraseña

El sprint 3 corresponde a todas las actividades relacionadas con las pruebas unitarias del *backend* y la creación del inicio de sesión, el cierre de sesión y la recuperación de la contraseña. En este *Sprint* se desarrollan las siguientes tareas:

- Tarea 1: Pruebas unitarias del *backend.*
- Tarea 2: Implementación del contexto global
- Tarea 3: Creación del formulario de inicio de sesión y de recuperación de contraseña.
- Tarea 4: Creación del botón de salida de sesión.
- Tarea 5: Implementación de la funcionalidad de recuperación de contraseña.

Pruebas unitarias del *backend.*

Se han realizado pruebas para todos los *resolvers* de los modelos creados previamente usando la herramienta Apollo Explorer, la cual ha permitido evaluar el funcionamiento de todos los endpoints usando datos reales. Se ha utilizado esta herramienta para hacer pruebas unitarias debido a que no es posible hacer pruebas unitarias para este tipo de tecnología. En la Fig. 32 se muestra la prueba unitaria exitosa al *resolver* para crear un nuevo usuario.

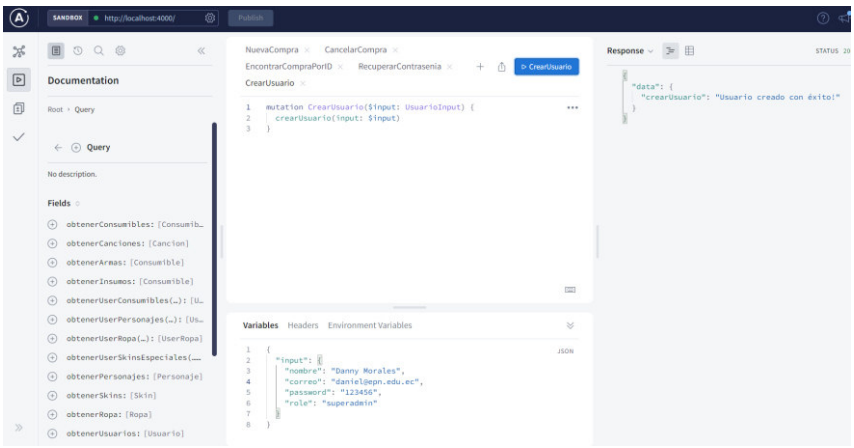


Fig. 32: Prueba unitaria del resolver crearUsuario()

Implementación del contexto global

Para implementar el contexto de la consola de control se usó React Context. En la Fig. 33 se muestra la implementación del proveedor de contexto y se pasaron los estados

globales de usuario, *token* y *login* para usarlos en todo el árbol de componentes de la aplicación.

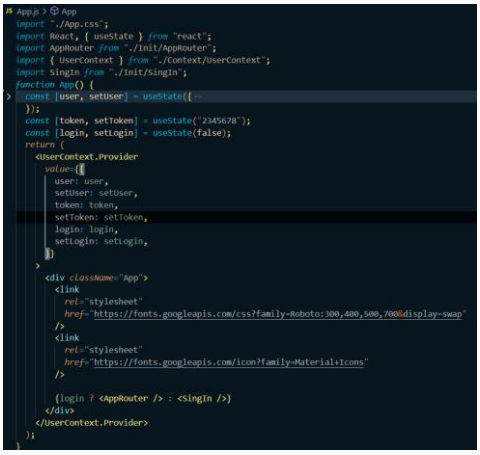


Fig. 33: Implementación del contexto global.

Creación del formulario de inicio de sesión y de recuperación de contraseña.

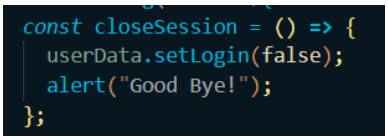
Se ha creado el formulario de inicio de sesión y de recuperación de contraseña usando HTML y CSS puro, en la Fig. 34 se muestra el resultado final de su implementación. Se ha implementado validaciones a los formularios usando React Hook Forms. En Fig. 35 se muestra la implementación de las validaciones del formulario de *Login*.

Fig. 34: Login y recuperación de Contraseña en servidor local.

Fig. 35: Formulario de Login

Creación del botón de salida de sesión.

Después de crear el botón para cerrar la sesión, se ha implementado la funcionalidad según la Fig. 36 en la cual se inactiva la sesión y se elimina el estado en el contexto. En la parte inferior de la Fig. 34 se muestra el botón de cerrar sesión.



```
const closeSession = () => {
  userData.setLogin(false);
  alert("Good Bye!");
};
```

Fig. 36: Función para cerrar sesión

Implementación de la funcionalidad de recuperación de contraseña.

Para la implementación se ha dotado de funcionalidad al botón "*Did you forget the password*", el cual despliega el formulario de recuperación de sesión como se muestra en la Fig. 37 y a la vez consume el *endpoint* para recuperar la contraseña. La implementación del *endpoint* requiere el uso del *hook* useMutation el cual pertenece a la librería de Apollo Client. El *hook* useMutation llama a la función de GraphQL específica para llamar al *callback* recuperar contraseña como se muestra en la Fig. 38.

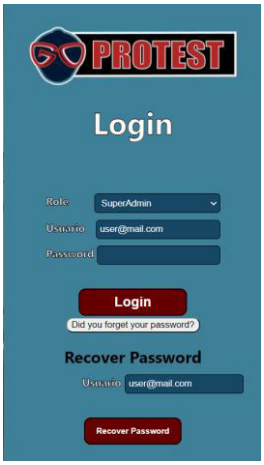


Fig. 37: Modulo para recuperar contraseña

```
const RECOVER_PASSWORD = gql`
  # Mutation to recover password
  mutation RecuperarContrasenia($recuperarContraseniaInput: CorreoInput) {
    recuperarContrasenia(input: $recuperarContraseniaInput)
  }
`;

export default function RecoverPassword() {
  const [recuperarContrasenia] = useMutation(RECOVER_PASSWORD);

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();
  const [loading, setLoading] = useState(false);
  const onSubmit = async (dataInput) => {
    console.log("dataInput", dataInput);
    try {
      setLoading(true);
      const { data } = await recuperarContrasenia({
        variables: {
          recuperarContraseniaInput: {
            correo: dataInput.correo,
          },
        },
      });
      //console.log("datos desde el servidor", data.recuperarContraseniaInput);

      setLoading(false);
      alert("Se ha enviado un correo con la nueva contraseña");
```

Fig. 38: Implementación de resolver para recuperar contraseña.

3.5 Sprint 4. Desarrollo de las secciones de Personajes, Consumibles, Skins, Ropa, Administradores y Supervisores.

El *sprint* 4 corresponde a todas las actividades relacionadas con el desarrollo de los módulos de Personajes, Consumibles, Ropa, Skins, Administradores y Supervisores. La implementación de las funcionalidades es parte de esta iteración. En este *Sprint* se desarrollan las siguientes tareas:

- Tarea 1: Creación de los formularios para los módulos de Personajes, Consumibles, Ropa, Skins, Administradores, Supervisores y Compras.
- Tarea 2: Implementación de las funcionalidades para los módulos.
- Tarea 3: Desarrollo del módulo de compras en la consola de control.

- Tarea 4: Implementación de las funcionalidades del módulo de compras.

Creación de los formularios para los módulos de Personajes, Consumibles, Ropa, Skins, Administradores, Supervisores y Compras

Se han creado los módulos siguiendo el diseño establecido. Los módulos de desarrollo contienen los formularios con sus respectivas validaciones creadas con React Hook Forms para simplificar la validación de datos en el frontal. En la Fig. 39 se muestra el diseño del módulo de Personajes. En el cual se aprecian las funcionalidades de crear, actualizar, eliminar personaje y obtener los personajes que posee un determinado usuario.

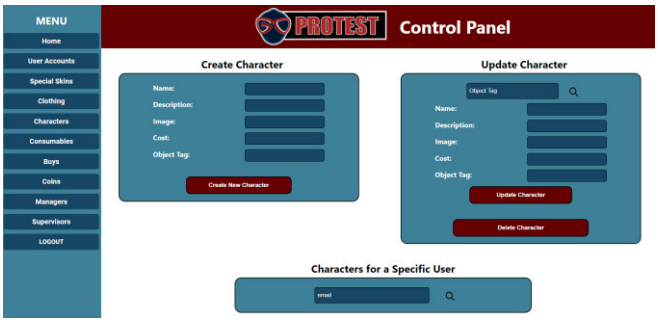


Fig. 39: Módulo de Personajes.

Implementación de las funcionalidades para los módulos.

Se han implementado las funcionalidades de todos los módulos de manera satisfactoria, en la Fig. 40 se aprecia la implementación del *endpoint* de creación de consumible. El *endpoint* recibe todos los datos del formulario y los envía al servidor para almacenarse en la base de datos.

```
const onSubmit = async (dataInput) => {
  console.log(dataInput);
  try {
    setLoading(true);
    const { data } = await nuevoConsumible({
      variables: {
        input: {
          nombre: dataInput.nombre,
          descripcion: dataInput.descripcion,
          costo: parseInt(dataInput.costo, 10),
          objectTag: dataInput.objectTag,
          imagen: dataInput.imagen,
          tipo: dataInput.tipo,
          dano: parseInt(dataInput.dano, 10),
          puntosDeVida: parseInt(dataInput.puntosDeVida, 10),
          cantidad: parseInt(dataInput.cantidad,10),
          danoPorSegundo: parseInt(dataInput.danoPorSegundo,10),
          incrementoAliados: parseInt(dataInput.incrementoAliados, 10),
          tiempoLiberacion: parseInt(dataInput.tiempoLiberacion, 10),
        },
      },
    });
    console.log("desde la funcion", data);
    setLoading(false);
    alert("Consumible created successfully!");
  } catch (error) {
    console.log(error);
    alert(error);
  }
};
```

Fig. 40: Implementación del endpoint de creación de Consumible.

Desarrollo del módulo de compras en la consola de control.

Se ha desarrollado el módulo de manera exitosa. Se concretó la creación del desplegable que permite obtener los artículos según el tipo de consumible elegido. Esto se puede apreciar en la Fig. 41. Los formularios contienen validaciones según las reglas del negocio y ha preparado para la implementación de funcionalidades.

Fig. 41: Módulo de compras de consumibles

Implementación de las funcionalidades del módulo de compras.

Se han implementado las funcionalidades al concretar el consumo de los *endpoints* en cada formulario del componente de Compras. En la Fig. 42 se aprecia una de las implementaciones de funcionalidad. El *enpoint* obtiene los datos de las compras según los usuarios específicos. En la Fig. 43 se observa el sistema en acción al obtener todas las compras de un usuario.

```
const GET_USER_BUYS = gql`
  query ObtenerComprasID($obtenerComprasIdId: ID!) {
    obtenerComprasID(id: $obtenerComprasIdId) {
      id
      IDUsuario
      costo
      fechaCompra
      nombreArticulo
    }
  }
`;

export default function RenderUserBuys({ userId }) {
  console.log("user desde renderuserBuys",userId)
  const { loading, error, data } = useQuery(GET_USER_BUYS, {
    variables: {
      obtenerComprasIdId: userId,
    },
  });
  if (loading) return "Loading...";
  //console.log("DESDE RENDER USER BUYS",data.obtenerComprasID);

  if (error) return `Error! ${error.message}`;
  return (
    <div>
      <div className="form-component">
        {data.obtenerComprasID.map((item) => {
          return (
            <div
              key={item.id}
              className="component-block"
              style={{ width: "100%", margin: "5px" }}
            >
              <div
                className="form-component"
                style={{ width: "100%", margin: "10px" }}
              >
```

Fig. 42: Implementación del renderizado de compras de un usuario.

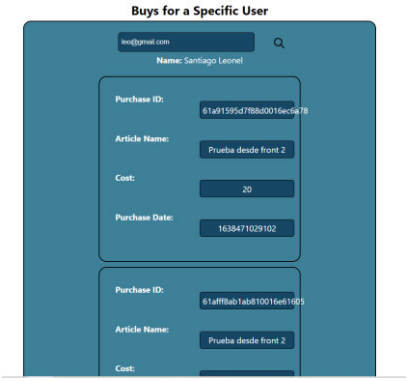


Fig. 43: Compras según el usuario

3.6 Sprint 5. Desarrollo de sección de Compras de monedas y Pruebas de Integración.

El *sprint* 5 corresponde a todas las actividades relacionadas con la creación del módulo de compras de monedas y las pruebas de integración. El módulo requiere de la implementación de un botón de PayPal para realizar compras con tarjeta de crédito o mediante saldo de PayPal y a la vez registrar dicha compra en la base de datos del sistema. Además, el sistema permite cancelar compras y renderizar las compras de monedas de un usuario específico. En este *Sprint* se desarrollan las siguientes tareas:

- Tarea 1: Desarrollo del módulo de compras de monedas en la consola de control.
- Tarea 2: Implementación de la funcionalidad del botón de PayPal.
- Tarea 3: Implementación de las funcionalidades del módulo de compras de monedas.
- Tarea 4: Configuración Previa para las pruebas de integración.
- Tarea 5: Pruebas de Integración.

Desarrollo del módulo de compras de monedas en la consola de control.

Se ha implementado todo el módulo de compras de monedas, se han creado los formularios necesarios para obtener la información c para generar una compra. Además, se instaló el botón de PayPal para llamar a la *API.* En la Fig. 44 se muestra el módulo completo.

Fig. 44: Módulo de Compras de monedas.

Implementación de la funcionalidad del botón de PayPal

El botón de PayPal se importa de la librería disponible de PayPal. Después de su importación se ha colocado en el componente *Payment*. En la Fig. 45 se muestra con claridad esa importación.

En la Fig. 46 se muestra el código del componente de creación de nueva compra el cual permite enviar el valor de la compra y llama a la *API* de PayPal para iniciar sesión y generar una compra. Para implementar el botón de compra, se creó un componente para la compra con PayPal o tarjeta de crédito. Este botón permite pasar información y métodos, lo cual es muy útil para pasar información de la compra. En la Fig. 47 se muestra la función *processToDatabase* la cual permite registrar la compra realizada con PayPal en la base de datos y la función *OnApprove* permite llamar al método *processToDatabase* cuando la transacción de PayPal es exitosa, registrando la compra en la base de datos del sistema.

```
const PayPalButton = window.paypal.Buttons.driver("react", { React, ReactDOM });

export default function Payment({ amountToPay, buyer, sucres }) {
  const [nuevaCompraMonedas] = useMutation(CREATE_COINS_BUY);
  //crear orden
  const createOrder = (data, actions) => {···
  };
  //procesar orden en base de datos
  const processBuyToDatabase = async (buyer, amountToPay, sucres) => {···
  };
  //si todo tiene exito
  const onApprove = (data, actions) => {···
  };
  return (
    <div
      style={{···
      }}
    >
      <PayPalButton
        createOrder={(data, actions) => createOrder(data, actions)}
        onApprove={(data, actions) => onApprove(data, actions)}
      />
    </div>
  );
```

Fig. 45: Módulo de Compras de monedas.

```
const createOrder = (data, actions) => {
  if (
    buyer === undefined &&
    amountToPay === undefined &&
    sucres === undefined
  ) {
    alert("You should fill all the fields");
    return;
  } else if (buyer === undefined) {
    alert("You should choose a buyer");
    return;
  } else if (sucres === undefined || sucres === 0) {
    alert("You should write a valid amount of Sucres");
    return;
  } else {
    console.log("cantidad a pagar ", amountToPay);
    return actions.order.create({
      purchase_units: [
        {
          amount: {
            value: amountToPay,
          },
        },
      ],
    });
  }
};
```

Fig. 46: Módulo de Compras de monedas

```
const processBuyToDatabase = async (buyer, amountToPay, sucres) => {
  //Registro en base de datos
  try {
    const { data } = await nuevaCompraMonedas({
      variables: {
        input: {
          cantidadMonedas: parseInt(sucres, 10),
          totalCancelado: parseFloat(amountToPay),
          IDUsuario: buyer,
        },
      },
    });
    console.log("desde la funcion", data);
    //setLoading(false);
    alert("Buy generated successfully!");
  } catch (error) {
    console.log(error);
    alert(error);
  }
};
//si todo tiene exito
const onApprove = (data, actions) => {
  console.log("comprador ", buyer);
  console.log("cantidad a pagar ", amountToPay);
  console.log("Sucres por comprar ", sucres);
  //database Process
  processBuyToDatabase(buyer, amountToPay, sucres);
  console.log("Purchase realized successfully!");
  return actions.order.capture();
};
```

Fig. 47: Funciones para registrar la información de la compra.

Implementación de las funcionalidades del módulo de compras de monedas.

Se ha implementado el consumo del *endpoint* para registrar la compra en la base de datos, además de las validaciones necesarias para evitar que se genere una compra sin cumplir los requisitos básicos para llevarse a cabo. En la Fig. 48 se muestra el funcionamiento óptimo de la compra con PayPal en la consola de control.

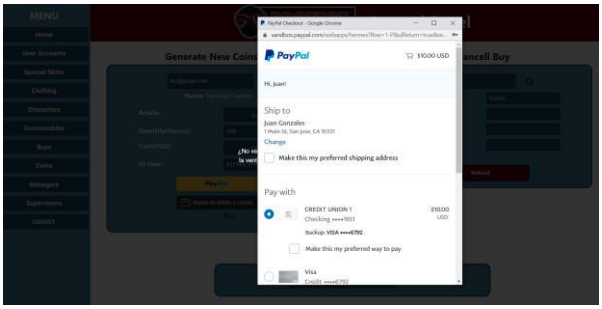


Fig. 48: Funcionalidad de compra por PayPal.

Configuración Previa para las pruebas de integración

Para realizar las pruebas de integración, se han condifurado todos los archivos de testeo por módulo a testear en el *backend*. En la Fig. 49 se muestra la importación de los typeDefs (los tipos), los *resolvers*, la función para conectar la base de datos, las variables de entorno, la librería mongoose y se importa ApolloServer para llamar una instancia del servidor de Apollo y gql para escribir las consultas a la base de datos.

```
const typeDefs = require("../db/schema");
const resolvers = require("../db/resolvers");
const { ApolloServer, gql } = require("apollo-server");
const conectarDB = require("../config/db");
require("dotenv").config("variables.env");
const mongoose = require("mongoose");
```

Fig. 49: Configuraciones para pruebas de integración.

Luego se copiaron las consultas realizadas en la consola de control en cada archivo de testeo según la necesidad. En la Fig. 50 se muestran las consultas de GraphQL del modelo de Personajes usado en el frontal, las cuales se copiaron en el archivo de testeo characters.test.js.

```
JS resolvers.js M    JS characters.test.js ×
End > test > JS characters.test.js > ...

const CREATE_CHARACTER = gql`
  mutation NuevoPersonaje($input: PersonajeInput) {
    nuevoPersonaje(input: $input) {
      nombre
    }
  }
`;
const UPDATE_CHARACTER = gql`
  mutation ActualizarPersonaje(
    $actualizarPersonajeId: ID!
    $input: PersonajeInput
  ) {
    actualizarPersonaje(id: $actualizarPersonajeId, input: $input) {
      id
      nombre
    }
  }
`;
const DELETE_CHARACTER = gql`
  mutation EliminarPersonaje($eliminarPersonajeId: ID!) {
    eliminarPersonaje(id: $eliminarPersonajeId)
  }
`;

const FIND_BY_OBJECTTAG = gql`
  # Mutation to find a Clothing by ObjectTag
  mutation EncontrarConsumibleByObjectTag($input: ObjectTagInput) {
    encontrarConsumibleByObjectTag(input: $input) {
      Personaje {
        id
        nombre
      }
    }
  }
`;
```

Fig. 50: Consultas de GraphQL del módulo Personajes (Characters)

Luego se ha configurado la conexión de la base de datos antes del inicio de las pruebas y la desconexión al finalizar las pruebas. En la Fig. 51 se muestran las funciones beforeAll la cual permite llamar una función antes de iniciar las pruebas y afterAll, la cual llama funciones luego de finalizar las pruebas. En todas las pruebas realizadas se desconectó la base de datos antes de las pruebas y desconectarlas después de finalizar con el objetivo de ahorrar recursos del servidor.

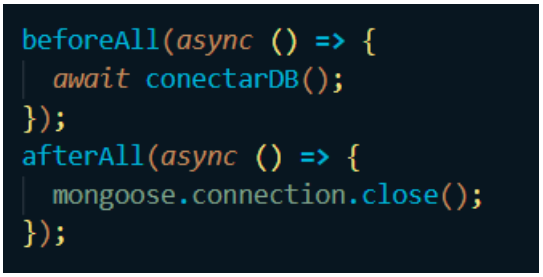


Fig. 51: Conexión y desconexión de la base de datos

Pruebas de Integración.

Para las pruebas de integración se ha instanciado e iniciado un servidor de Apollo para luego ejecutar las operaciones que se usaron en la consola de control. En la Fig. 52 se muestra cómo se implementó una prueba de creación de Personaje en el archivo de testeo characters.test.js. Al ejecutar una operación se pasó la consulta CREATE_CHARACTER en el atributo *query*, y en variables se coloca la información a enviar al servidor. Esta información debe subirse según el esquema del *payload* de la consola de control.

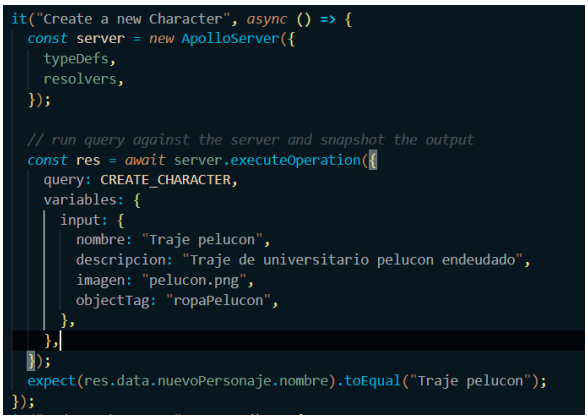


Fig. 52: Ejecución de la operación de creación de personaje.

Se han creado pruebas de integración para todos los *endpoints* utilizados en el frontal. Lo cual verificó el funcionamiento correcto del sistema. En la Fig. 53 se aprecia el testeo exitoso de las 24 pruebas de integración realizadas en el proyecto.

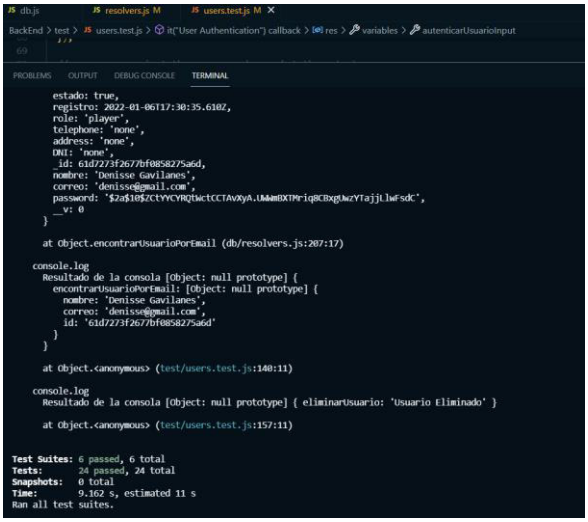


Fig. 53: Resultados de las pruebas de integración.

3.7 Sprint 6. Configuración de Sonarqube, Synk y Despliegue a Producción.

El sprint 6 corresponde a todas las actividades relacionadas con la conexión del proyecto de backend con el servicio en línea de sync.io, y también se utiliza un runner de GitLab CI para hacer evaluación de calidad de código con Sonarqube, pruebas automatizadas y despliegue a producción de la consola de control en Firebase y el backend en Heroku. En este *Sprint* se desarrollan las siguientes tareas:

- Tarea 1: Conexión de la consola de control y el backend con Synk.io.
- Tarea 2: Implementación de GitLab CI en el backend del proyecto.
- Tarea 3: Integración Continua.
- Tarea 4: Despliegue de la Consola de Control.
- Tarea 5: Despliegue del Backend.

Conexión de la consola de control y el backend con Synk.io

Se ha realizado la conexión de los repositorios de la consola de control y del *backend* con el servicio en la nube de Synk, el cual evalúa la seguridad del código y presenta informes constantes sobre las vulnerabilidades del proyecto. En la Fig. 54 se muestran los repositorios conectados al servicio. Con esto cada vez que se realiza un *commit* y también una vez por semana envía un reporte al correo electrónico como se muestra en la Fig. 55. El reporte manifiesta vulnerabilidades de seguridad y advierte

de una amenaza de denegación de servicio. Con este informe es posible alertar al equipo de desarrollo y solucionar esa vulnerabilidad.

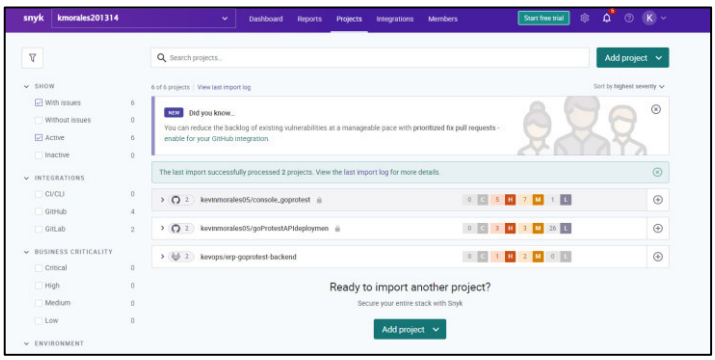


Fig. 54: Repositorios conectados Synk.io

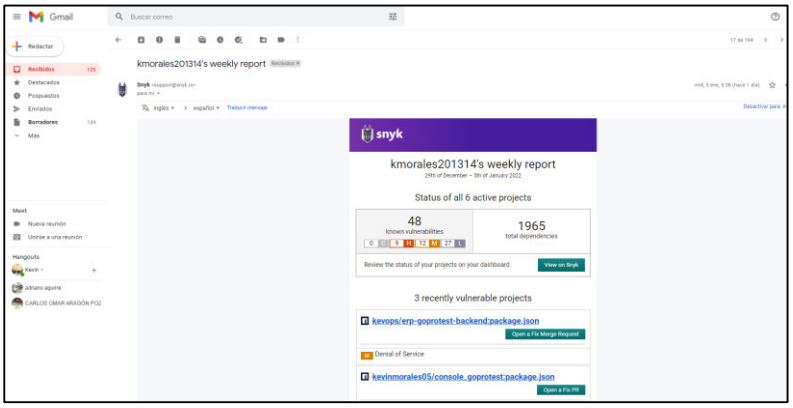


Fig. 55: Reporte semanal de Synk.io

Implementación de GitLab CI en el *backend* del proyecto.

El proyecto de *backend* se subió a un repositorio de GitLab para luego utilizar el servicio de integración continua. Para configurar el análisis continuo del código se ha creado una cuenta en Sonarcloud.io en la cual se obtiene una clave para insertar en el archivo de configuración de integración continua. En la Fig. 56 se muestra la clave creada para el proyecto. Se ha creado un archivo gitlab-ci.yml en el directorio raíz del proyecto de la API. El archivo se ha configurado para automatizar la construcción del proyecto, el testeo y la evaluación con SonarCube. En la Fig. 57 se muestra el archivo de configuración para la integración continua del proyecto. Este archivo se ejecuta cada vez que se realiza un *commit* a la rama principal del proyecto.

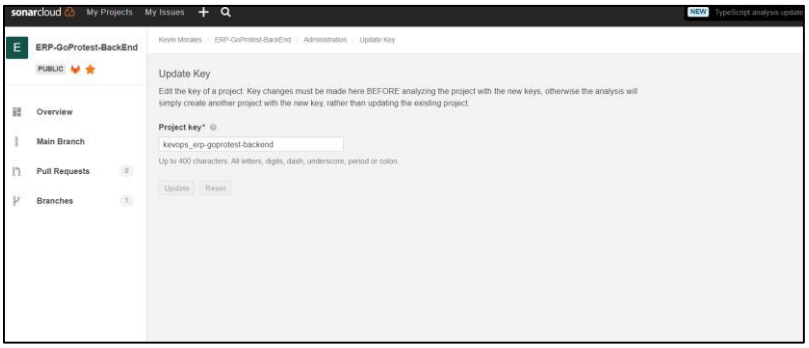


Fig. 56: Creación de una clave de SonarQube para GitLab CI

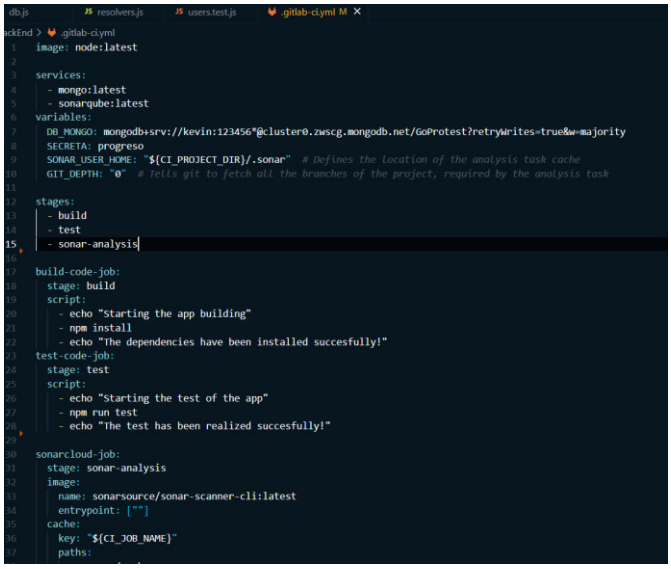


Fig. 57: Código del archivo de configuración de GitLab CI

Integración Continua

Se realizó un *commit* al repositorio de GitLab para ejecutar el archivo de integración continua. En la Fig. 58 se muestra el *pipeline* de GitLab al correr de manera secuencial los trabajos (*Jobs*) que se configuraron para evaluar el proyecto de software. En la Fig. 59 se muestra la evaluación del código en SonarCloud, mostrándose datos de alto interés para el proyecto. El proyecto tiene 0 vulnerabilidades, 1 Bug y 43 riesgos de errores o deficiencias en el diseño (*code smells*). Con esto es posible tomar acciones para corregir los problemas manifestados en el análisis.

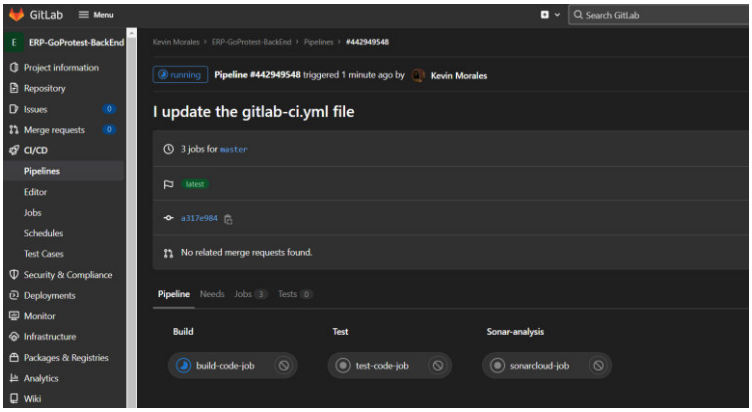


Fig. 58: Pipeline del proyecto

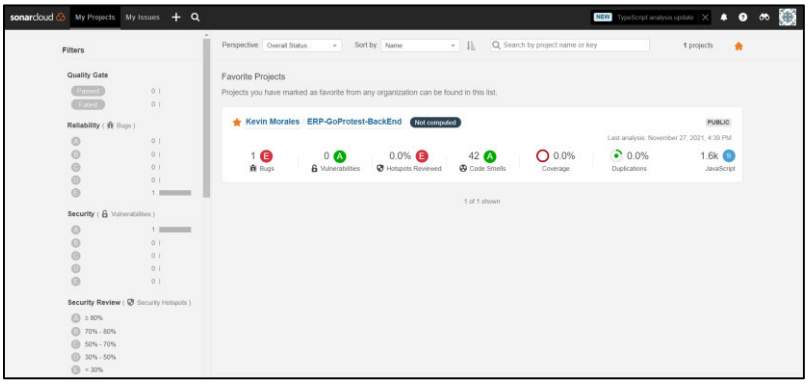


Fig. 59: Análisis de calidad de código con SonarQube

Despliegue de la Consola de Control

Se ha desplegado la aplicación en Firebase, para ello fue necesario la creación de un proyecto de Firebase como se muestra en la Fig. 60. Con la aplicación creada se procedió a realizar la construcción del proyecto de React. Después de llenar la información y anclar la carpeta "*build*" al inicializar el proyecto de Firebase en el directorio raíz del proyecto. Se utilizó el comando "*firebase deploy*". Con eso la aplicación se desplegó en los servidores de Firebase, la aplicación en producción se muestra en la Fig. 61.

Fig. 60: despliegue de la consola de control en Firebase y el backend en Heroku.

Fig. 61: Aplicación en Producción

Despliegue del Backend

El *backend* se ha desplegado en Heroku, para ello se ha creado una cuenta y se instaló Heroku CLI previamente, con ello se ha inicializado un proyecto nuevo mediante la consola de comandos y se subió a un repositorio de producción. En la Fig. 62 se puede ver el proyecto desplegado y activo en la consola de Heroku. El proyecto se llama frozen-mez-22948. Se escogió Heroku porque tiene soporte para proyectos basados en Node.

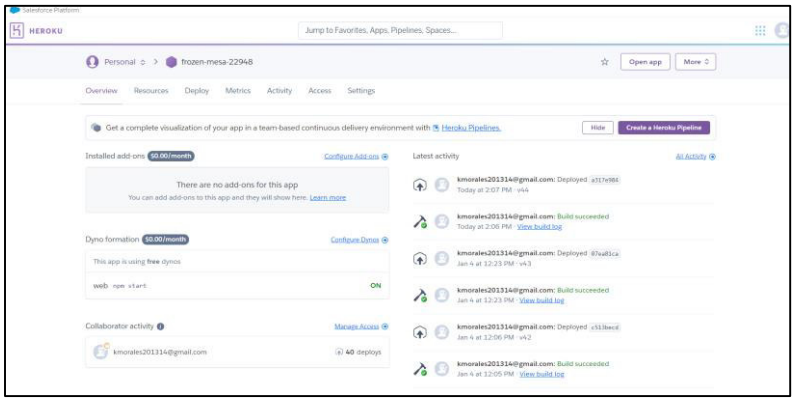


Fig. 62: Backend levantado en Heroku

4 CONCLUSIONES

- Se consiguió desarrollar una *API* para el videojuego *Go Protest*, la cual es robusta y adaptada a los requerimientos funcionales y no funcionales del sistema utilizando NodeJS, GraphQL y Apollo Server, las cuales son tecnologías de vanguardia y que permiten un acople a servicios o clientes con *frameworks* o lenguajes de programación diferentes a JavaScript.
- Se logró desarrollar una consola tipo ERP para controlar y administrar el negocio que administrará el videojuego *Go Protest*. La consola es potente y goza de rutas protegidas según el rol del usuario que ingresa al nivel.
- No se logró realizar pruebas unitarias del proyecto debido a que la tecnología utilizada no permite realizar este tipo de pruebas. Las pruebas que se realizaron fueron de integración ya que los hooks que se usan para trabajar con GraphQL deben ser renderizados dentro de un componente de React o un ejecutor de servicios. Se planificaron pruebas unitarias debido a que se desconocía de este detalle antes de iniciar la investigación.
- La evaluación del código del proyecto en Synk permitió conocer las vulnerabilidades de seguridad y proporcionó soluciones para mitigar el riesgo de dichas vulnerabilidades.
- La integración de GitLab CI permitió evaluar el código en SonarQube y correr las pruebas de manera automática cada vez que se hacía un commit al repositorio. Usar integración continua es muy útil para obtener informes de calidad de código y vulnerabilidades de seguridad.
- Se desplegó el proyecto se con éxito en los servicios en la nube planteados en el trabajo.
- El proyecto no requirió un diseño responsivo debido a que el sistema es para usarlo en una computadora de escritorio o laptop.

5 RECOMENDACIONES

- Se recomienda evaluar las tecnologías a utilizarse con más detenimiento antes de planificar las iteraciones. En este caso no se debió planificar pruebas unitarias si la tecnología no tenía soporte para ello.
- Se recomienda implementar despliegue continuo en la pila(pipeline) de GitLab CI para facilitar el despliegue de la aplicación.
- Se recomienda no realizar muchos commits innecesarios al repositorio de GitLab porque el servicio gratuito de runners es muy limitado. GitLab obliga a registrar una tarjeta de crédito para ocupar los runners en la nube. Si se ocupa en exceso el costo a pagar es considerable.
- La consola de control está desarrollada con React Router versión 5.3.0, el proyecto puede romperse en caso de actualizar a la versión 6. Esta versión tuvo varios cambios que alteran los componentes de navegación de la aplicación.
- El servicio gratuito de correo electrónico proporcionado por MailSlurp e implementado en el backend tiene restricción a envío de correos electrónicos a dominios terminados en Gmail.com. esta restricción se da porque la cuenta no es premium. Si se desea trabajar en producción se recomienda contratar el plan de pago.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] Red Hat, «¿Qué es la metodología ágil?,» 2022. [En línea]. Available: https://www.redhat.com/es/devops/what-is-agile-methodology. [Último acceso: 11 Enero 2022].

[2] N. Chapaval, «Qué es Frontend y Backend: diferencias y características - Platzi,» Platzi , 2018. [En línea]. Available: https://platzi.com/blog/que-es-frontend-y-backend/. [Último acceso: 11 Enero 2022].

[3] M. Massé, «REST APIs,» de *REST API Design Rulebook*, Sebastopol, O'Reilly, 2021, pp. 5,6.

[4] S. M. e. al., «La metodología de estudio de caso,» Revista Espacios, 27 Mayo 2019. [En línea]. Available: https://www.revistaespacios.com/a19v40n17/a19v40n17p16.pdf. [Último acceso: 10 Enero 2022].

[5] Brenton, «Los beneficios de usar Scrum en su organización,» 22 Julio 2019. [En línea]. Available: https://blog.opinno.io/es/blog/los-beneficios-de-usar-scrum-en-su-organizacion. [Último acceso: 10 Enero 2022].

[6] DesarrolloWeb, «Qué es MVC,» DesarrolloWeb.com, 28 Julio 2020. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: 24 Noviembre 2021].

[7] A. Ramirez, «7 razones para usar MongoDB en tus Proyectos,» Platzi, 2018. [En línea]. Available: https://platzi.com/blog/7-razones-mongodb/. [Último acceso: 24 Noviembre 2021].

[8] MongoDB, «MongoDB Clusters,» MongoDB, 2021. [En línea]. Available: https://www.mongodb.com/basics/clusters. [Último acceso: 24 Noviembre 2021].

[9] MongoDB, «Compass. The GUI for MongoDB.,» MongoDB, 2021. [En línea]. Available: https://www.mongodb.com/products/compass. [Último acceso: 24 Noviembre 2021].

[10] Apollo Docs, «Introduction to Apollo Server,» Apollo, 2021. [En línea]. Available:

https://www.apollographql.com/docs/apollo-server/. [Último acceso: 24 Noviembre 2021].

[11] J. Lucas, «Qué es NodeJS y para qué sirve,» Open Webinars, 4 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-nodejs/. [Último acceso: 24 Noviembre 2021].

[12] J. Acosta, «GraphQL: Qué es y qué ventajas ofrece,» Open Webinars, 29 Junio 2021. [En línea]. Available: https://openwebinars.net/blog/graphql-que-es-y-que-ventajas-ofrece/. [Último acceso: 24 Noviembre 2021].

[13] Apollo Docs, «Introduction to Apollo Studio,» Apollo, 2021. [En línea]. Available: https://www.apollographql.com/docs/studio/. [Último acceso: 24 Noviembre 2021].

[14] R. Celis, «Heroku: qué es, cómo funciona y para qué sirve - Platzi,» 2017. [En línea]. Available: https://platzi.com/blog/que-es-heroku/. [Último acceso: 24 Noviembre 2021].

[15] Desarrollo Web, «Qué es React. Por qué usar React,» DesarrolloWeb.com, 25 Febrero 2019. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html. [Último acceso: 24 Noviembre 2021].

[16] Apollo Client, «Introduction to Apollo Client,» Apollo, 2021. [En línea]. Available: https://www.apollographql.com/docs/react/. [Último acceso: 24 Noviembre 2021].

[17] «Firebase te ayuda a compilar,» Google, 2021. [En línea]. Available: https://firebase.google.com/?hl=es-419&gclid=CjwKCAiA4veMBhAMEiwAU4XRr0WuTxEG2xoxfRhG9abLkpOFc-Hk5ujVJZSscgy00LecobMtz6bYNBoC4r0QAvD_BwE&gclsrc=aw.ds. [Último acceso: 24 Noviembre 2021].

[18] Capterra, «¿Qué es ZenHub?,» 2021. [En línea]. Available: https://www.capterra.ec/software/141621/zenhub. [Último acceso: 24 Noviembre 2021].

[19] Wikipedia, «SonarQube,» Wikipedia, 10 Diciembre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/SonarQube. [Último acceso: 24 Noviembre 2021].

[20] Synk, «SCM (Git) and CI/CD integration deployment intro,» 2021. [En línea]. Available: https://docs.snyk.io/getting-started/scm-git-and-ci-cd-integration-

deployment-intro. [Último acceso: 24 Noviembre 2021].

[21] Scio, «¿Por qué es importante la arquitectura de software?,» 3 Septiembre 2019. [En línea]. Available: https://www.scio.com.mx/blog/por-que-es-importante-la-arquitectura-de-software/. [Último acceso: 24 Noviembre 2021].

[22] RAE, «Metodología,» 2022, [En línea]. Available: https://dle.rae.es/metodolog%C3%ADa?m=form. [Último acceso: 11 Enero 2022].

[23] J. F. Pareja Quinaluisa, «Evaluación de procesos de software utilizando EvalProSoft Aplicado a un caso de estudio,» 08 02 2012. [En línea]. Available: http://bibdigital.epn.edu.ec/handle/15000/4491 .

7. ANEXOS

ANEXO I. Certificado de originalidad
ANEXO II. Manual técnico
ANEXO III. Manual de usuario
ANEXO III. Manual de instalación

Anexo I. Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 24 de febrero de 2022

De mi consideración:

Yo, Richard Paúl Rivera Guevara, en calidad de director del Trabajo de Integración Curricular titulado DESARROLLO DEL BACKEND Y LA CONSOLA DE CONTROL DEL VIDEOJUEGO MULTIJUGADOR GO PROTEST, elaborado por el estudiante KEVIN DANIEL MORALES ESTRELLA de la carrera en TECNOLOGÍA SUPERIOR EN DESARROLLO DE SOFTWARE, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 9%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el informe generado por la herramienta Turnitin en el Quipux correspondiente.

Atentamente,

RICHARD PAUL RIVERA GUEVARA

Firmado digitalmente por RICHARD PAUL RIVERA GUEVARA
Fecha: 2022.02.24 14:33:22 -05'00'

Dr. Richard Rivera.
Profesor EPN-ESFOT

ANEXO II. MANUAL TÉCNICO

Recopilación de requerimientos

TABLA VI Requerimientos

RECOPILACIÓN DE REQUERIMIENTOS		
TIPO DE SISTEMA	**ID - RR**	**ENUNCIADO DEL ÍTEM**
Front End* y *Backend	**RR001**	Como usuario super administrador, supervisor o administrador puede crear, actualizar, eliminar, activar o desactivar cuentas de usuarios con el rol de jugador mediante un formulario.
	RR002	Como usuario super administrador, supervisor o administrador puede actualizar o eliminar cuentas de usuarios con el rol de jugador mediante un formulario.
	RR003	Como usuario super administrador, supervisor o administrador puede activar o desactivar cuentas de usuarios con el rol de jugador mediante un formulario.
	RR004	Como usuario super administrador o supervisor puede generar una compra de sucres (moneda del juego) mediante un formulario.
	RR005	Como usuario super administrador o supervisor puede cancelar una compra de sucres (moneda del juego) mediante un formulario.
	RR006	Como usuario super administrador o supervisor puede ver los compas de sucres (moneda del juego) que tiene un determinado jugador mediante un formulario.
	RR007	Como usuario super administrador o supervisor puede crear un nuevo skin para el juego mediante un formulario.
	RR008	Como usuario super administrador o supervisor puede eliminar o editar un skin para el juego mediante un formulario.
	RR009	Como usuario super administrador o supervisor puede ver los skins que tiene un determinado jugador mediante un formulario.
	RR010	Como usuario super administrador o supervisor puede crear ropa para el juego mediante un formulario.
	RR011	Como usuario super administrador o supervisor puede eliminar o editar ropa para el juego mediante un formulario.
	RR012	Como usuario super administrador o supervisor puede ver la ropa que tiene un determinado jugador mediante un formulario.
	RR013	Como usuario super administrador o supervisor puede crear personajes para el juego mediante un formulario.
	RR014	Como usuario super administrador o supervisor puede eliminar

	o editar personajes para el juego mediante un formulario.
RR015	Como usuario super administrador o supervisor puede ver los personajes que tiene un determinado jugador mediante un formulario.
RR016	Como usuario super administrador o supervisor puede crear consumibles para el juego mediante un formulario.
RR017	Como usuario super administrador o supervisor puede eliminar o editar consumibles para el juego mediante un formulario.
RR018	Como usuario super administrador o supervisor puede ver los consumibles que tiene un determinado jugador mediante un formulario.
RR019	Como usuario super administrador puede crear supervisores para el sistema de control mediante un formulario.
RR020	Como usuario super administrador puede eliminar o editar supervisores para sistema de control mediante un formulario.
RR021	Como usuario super administrador puede activar o desactivar supervisores para sistema de control mediante un formulario.
RR022	Como usuario super administrador o supervisor puede crear administradores para el sistema de control mediante un formulario.
RR023	Como usuario super administrador o supervisor puede eliminar o editar administradores para sistema de control mediante un formulario.
RR024	Como super administrador o supervisor puede activar o desactivar administradores para sistema de control mediante un formulario.
RR025	Como usuario super administrador, supervisor o administrador puede recuperar su contraseña mediante un formulario.
RR026	Como usuario super administrador, supervisor o administrador pueden iniciar sesión según su rol, correo y contraseña mediante un formulario siempre y cuando su cuenta esté activa.
RR027	Como usuario administrador, supervisor o administrador puede recuperar su contraseña antes de iniciar sesión mediante un formulario.
RR028	Como usuario administrador, supervisor o administrador puede ver todo el inventario de consumibles de un usuario según el tipo de consumible (weapon, food, munition, health) mediante un formulario.
RR029	Como usuario administrador, supervisor o administrador puede agregar un consumible un usuario según el tipo de consumible (weapon, food, munition, health), siempre y cuando tenga

		presupuesto mediante un formulario.

Historias de usuario

TABLA VII HUOOO

HISTORIA DE USUARIO	
Identificador (ID): HU000	**Usuario:** Todos los usuarios
Nombre Historia: Configuraciones iniciales	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 0	
Responsable: Kevin Morales	
Descripción: Configurar el entorno de Node.js para iniciar el desarrollo de la API, se desarrollará el modelo de la base de datos y la arquitectura del sistema. Se instalará Nodejs, y se instalarán las extensiones necesarias para Visual Studio Code.	
Observación: Se instalará las siguientes dependencias: Apollo Server, GraphQL, Mongoose ,se creará el repositorio en GitHub, mailslurp-client para el servicio de correo, dotenv para manejo de variables de entorno, bcrypths para encriptar los passwords, jest para hacer pruebas unitarias, nodemon para manejar un servidor local, se creará una cuenta de mail-slurp para usar el servicio de correo y se instalará Heroku CLI para preparar el ambiente de producción.	

TABLA VIII HUOO1

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario:** Todos los usuarios
Nombre Historia: Registrar usuarios	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: El administrador deberá proporcionar la funcionalidad de registro en el backend, para que el usuario pueda registrarse en el sistema.	
Observación: El registro deberá contener los siguientes campos: nombre, correo, contraseña, nivel, sucres, estado, registro, rol, teléfono, dirección y DNI. La contraseña debe almacenarse encriptado en la base de datos.	

TABLA IX HU002

HISTORIA DE USUARIO	
Identificador (ID): HU002	**Usuario:** Administrador

Nombre Historia: Autenticar usuarios.	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: El administrador deberá brindar la funcionalidad de autenticar a los usuarios finales en el backend que deseen iniciar sesión en el sistema según el rol.	
Observación: El formulario de inicio de sesión deberá contener los campos de email, de contraseña y rol para efectuar la autenticación en el sistema. Los roles son: player, superadmin, supervisor, admin.	

TABLA X HU003

HISTORIA DE USUARIO	
Identificador (ID): HU003	**Usuario:** Administrador
Nombre Historia: CRUDs para Usuarios	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver usuarios según su id, listar todos los usuarios, editar su información, inactivar cuentas de usuarios y eliminar usuarios.	
Observación: Generar todos los *resolvers* para cumplir con todas las operaciones CRUD del *modelo* usuario.	

TABLA XI HU004

HISTORIA DE USUARIO	
Identificador (ID): HU004	**Usuario:** Administrador
Nombre Historia: Recuperación de Contraseña para usuarios	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Generar un *resolver* para recuperar la contraseña de los usuarios en el backend al ingresar el correo electrónico del usuario.	

Observación: Usar el servicio de correo electrónico para recuperar la contraseña del usuario mediante el envío de un password nuevo al correo electrónico registrado del mismo.

TABLA XII HU005

HISTORIA DE USUARIO	
Identificador (ID): HU005	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo Personajes	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todos los personajes, editar la información de un personaje, crear un personaje y eliminar un personaje según su id.	
Observación: El modelo Personajes debe tener los siguientes campos: nombre, descripción, costo, imagen y objectTag.	

TABLA XIII HU006

HISTORIA DE USUARIO	
Identificador (ID): HU006	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo Ropa	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver toda la ropa, editar la información de la ropa, crear ropa y eliminar ropa según su id.	
Observación: El modelo Ropa debe tener los siguientes campos: nombre, descripción, costo, imagen y objectTag.	

TABLA XIV HU007

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo Consumible	

Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todos los consumibles, editar la información de un consumible, crear consumible y eliminar consumible según su id.	
Observación: El modelo Consumible debe tener los siguientes campos: nombre, tipo, descripción, costo, daño, puntos de vida, cantidad, daño por segundo, incremento aliados, tiempo de liberación, imagen, estado y objectTag.	

TABLA XV HU008

HISTORIA DE USUARIO	
Identificador (ID): HU008	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo Compras	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todas las compras, editar la información de una compra, crear compra y eliminar compra según su id.	
Observación: El modelo Compra debe tener los siguientes campos: nombreArticulo, IDConsumible, IDRopa, IDSkinEspecial, IDPersonaje, fechaCompra, costo y IDUsuario.	

TABLA XVI HU009

HISTORIA DE USUARIO	
Identificador (ID): HU009	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo Compras de Monedas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todas las compras de monedas, editar la información de una compra de moneda, crear compra de moneda y eliminar compra de moenda según su id.	
Observación: El modelo CompraMonedas debe tener los siguientes campos: nombreArticulo, cantidadMonedas, fechaCompra, totalCancelado y IDUsuario	

TABLA XVII HU0010

HISTORIA DE USUARIO	
Identificador (ID): HU0010	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo UserConsumible	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todas los consumibles adquiridos por un usuario según su id, editar la información de una consumible de usuario según su id, asignar un consumible a un usuario y eliminar la asignación de un consumible a un usuario.	
Observación: El modelo UserConsumible debe tener los siguientes campos: nombreConsumible, imagenConsumible, cantidadDisponible, objectTag, idUsuario y idConsumible.	

TABLA XVIII HU0011

HISTORIA DE USUARIO	
Identificador (ID): HU0011	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo UserPersonaje	
Prioridad en Negocio: Medio	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todos los personajes adquiridos por un usuario según su id, editar la información de personaje de usuario según su id, asignar un personaje a un usuario y eliminar la asignación de un personaje a un usuario.	
Observación: El modelo UserPersonaje debe tener los siguientes campos: nombrePersonaje, imagenPersonaje, objectTag, idUsuario y idPersonaje.	

TABLA XIX HU0012

HISTORIA DE USUARIO	
Identificador (ID): HU0012	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo UserSkinsEspeciales	
Prioridad en Negocio: Medio	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver los skins especiales según el id del usuario, editar la información de un skin especial, crear un skin especial y eliminar un skin especial.	
Observación: El modelo UserSkinsEspeciales debe tener los siguientes campos: nombreSkin, imagenSkin, objectTag, cantidadDisponible, idUsuario, idSkinEspecial.	

TABLA XX HU0013

HISTORIA DE USUARIO	
Identificador (ID): HU0013	**Usuario:** Administrador
Nombre Historia: CRUDs para el modelo UserRopa	
Prioridad en Negocio: Medio	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 1	
Responsable: Kevin Morales	
Descripción: Crear *resolvers* para ver todas la ropa adquirida por un usuario según su id, editar la información de la ropa de usuario según su id, asignar ropa a un usuario y eliminar la asignación de ropa a un usuario.	
Observación: El modelo UserRopa debe tener los siguientes campos: nombreRopa, imagenRopa, objectTag, idUsuario y idRopa.	

TABLA XXI HU0013

HISTORIA DE USUARIO	
Identificador (ID): HU0013	**Usuario:** Administrador
Nombre Historia: Levantar Cluster de MongoDB en MongoDB Atlas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 1	
Responsable: Kevin Morales	

Descripción: Crear un cluster de base de datos de MongoDB y asignar las variables al entorno de desarrollo. Se desplegará el backend en Heroku.
Observación: Previo a levantar la base de datos, se desarrollará el modelo de la base de datos para crear las colecciones necesarias para el funcionamiento de la API.

TABLA XXII HU0015

HISTORIA DE USUARIO	
Identificador (ID): HU0015	**Usuario:** Administrador
Nombre Historia: Desarrollar la maqueta de la consola de control.	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Baja
Iteración Asignada: 2	
Responsable: Kevin Morales	
Descripción: Se procede a crear la consola de control y a desarrollar la estructura general de la aplicación. Se desarrolla la sección de inicio de sesión y el panel de administración.	
Observación: Se levanta la maqueta de la consola de control sin funcionalidades. El enfoque es en la sección de Inicio de Sesión y el Panel de administración. En este momento se debe contar con los mockups de la consola de control.	

TABLA XXIII HU0015

HISTORIA DE USUARIO	
Identificador (ID): HU0015	**Usuario:** Administrador
Nombre Historia: Levantar el Menú de la Consola de Control	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	
Responsable: Kevin Morales	
Descripción: El menú es un componente dinámico que se despliega según el rol del usuario con sesión iniciada. En esta sección se configura el contexto de la aplicación para que el usuario pueda ingresar a la sesión siempre y cuando este con sesión activa.	
Observación: Se utilizará Context para manajar variables globales en toda la aplicación.	

TABLA XXIV HU0017

HISTORIA DE USUARIO	
Identificador (ID): HU0017	**Usuario:** Administrador
Nombre Historia: Pruebas unitarias del Backend	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 3	
Responsable: Kevin Morales	
Descripción: Se realizará pruebas unitarias de todos los resolvers del proyecto usando la librería JEST.	
Observación: Se realizará pruebas unitarias asíncronas de los resolvers.	

TABLA XXV HU0018

HISTORIA DE USUARIO	
Identificador (ID): HU0018	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Login y recuperación de la clave.	
Prioridad en Negocio: Alto	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 3	
Responsable: Kevin Morales	
Descripción: Desarrollar el formulario y realizar el consumo de los endpoints para dar funcionalidad a la sección de Login y recuperación de contraseña.	
Observación: Se va a desarrollar la funcionalidad de cierre de sesión y ajustar el context para recibir el usuario que se obtiene de la consulta y permitir que la información esté disponible en todos los componentes.	

TABLA XXVI HU0019

HISTORIA DE USUARIO	
Identificador (ID): HU0019	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Personajes y Consumibles	
Prioridad en Negocio: Medio	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 4	
Responsable: Kevin Morales	

Descripción: Desarrollar los formularios necesarios y realizar el consumo de los *endpoints* para dar funcionalidad a dichas secciones.
Observación: Desarrollar las secciones necesarias e implementar los consumos necesarios en la consola de control.

TABLA XXVII HU0020

HISTORIA DE USUARIO	
Identificador (ID): HU0020	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Special Skins y Ropa	
Prioridad en Negocio: Bajo	**Riesgo en Desarrollo:** Bajo
Iteración Asignada: 4	
Responsable: Kevin Morales	
Descripción: Desarrollar los formularios necesarios y realizar el consumo de los endpoints necesarios para dar funcionalidad a dichas secciones.	
Observación: Desarrollar las secciones necesarias e implementar los consumos necesarios en la consola de control.	

TABLA XXVIII HU0021

HISTORIA DE USUARIO	
Identificador (ID): HU0021	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Administradores y Supervisores	
Prioridad en Negocio: Alto	**Riesgo en Desarrollo:** Alto
Iteración Asignada: 4	
Responsable: Kevin Morales	
Descripción: Desarrollar los formularios necesarios y realizar el consumo de los *endpoints* necesarios para dar funcionalidad a dichas secciones.	
Observación: Desarrollar las secciones necesarias e implementar los consumos necesarios en la consola de control.	

TABLA XXIX HU0022

HISTORIA DE USUARIO

Identificador (ID): HU0022	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Compras	
Prioridad en Negocio: Medio	**Riesgo en Desarrollo:** Medio
Iteración Asignada: 5	
Responsable: Kevin Morales	
Descripción: Desarrollar los formularios necesarios y realizar el consumo de los *endpoints* necesarios para dar funcionalidad a dichas secciones.	
Observación: Desarrollar las secciones necesarias e implementar los consumos necesarios en la consola de control.	

TABLA XXX HU0023

HISTORIA DE USUARIO	
Identificador (ID): HU0023	**Usuario:** Administrador
Nombre Historia: Desarrollar la sección de Compras de Monedas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 5	
Responsable: Kevin Morales	
Descripción: Desarrollar los formularios necesarios y realizar el consumo de los *endpoints* necesarios para dar funcionalidad a dichas secciones.	
Observación: Desarrollar las secciones necesarias e implementar los consumos necesarios en la consola de control. En esta sección se va a implementar el consumo de la API de PayPal para realizar compras.	

TABLA XXXI HU0024

HISTORIA DE USUARIO	
Identificador (ID): HU0024	**Usuario:** Administrador
Nombre Historia: Configuración de SonarQube y Synk	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 6	
Responsable: Kevin Morales	
Descripción: Subir el proyecto a GitLab para realizar la configuración necesaria para implementar CI con SonarQube y Synk del backend y de la consola de control.	

Observación: Se solventará las vulnerabilidades de seguridad de las aplicaciones y se hará refactorización en caso de que la calidad del código sea baja.

TABLA XXXII HU0025

HISTORIA DE USUARIO	
Identificador (ID): HU0025	**Usuario:** Administrador
Nombre Historia: Despliegue a producción	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 6	
Responsable: Kevin Morales	
Descripción: Subir la consola de control a Firebase y el backend a Heroku.	
Observación: Se desplegará el proyecto a producción para que esté disponible en internet.	

Product Backlog

TABLA XXXIII - Product Backlog

ELABORACIÓN DEL PRODUCT BACKLOG				
ID –HU	**HISTORIA DE USUARIO**	**ITERACIÓN**	**ESTADO**	**PRIORIDAD**
HU000	Configuraciones iniciales	**0**	**Finalizado**	**Alta**
HU001	Registrar usuarios	1	**Finalizado**	**Alta**
HU002	Autenticar usuarios	1	**Finalizado**	**Alta**
HU003	CRUDs para Usuarios	1	**Finalizado**	Alta
HU004	Recuperación de Contraseña para usuarios	1	**Finalizado**	Alta
HU005	CRUDs para el modelo Personajes	1	**Finalizado**	Media
HU006	CRUDs para el modelo Ropa	1	**Finalizado**	Media
HU007	CRUDs para el modelo Consumible	1	**Finalizado**	Alta
HU008	CRUDs para el modelo Compras	1	**Finalizado**	Alta
HU009	CRUDs para el modelo Compras de Monedas	1	**Finalizado**	Alta
HU010	CRUDs para el modelo UserConsumible	1	**Finalizado**	Medio
HU011	CRUDs para el modelo UserPersonaje	1	**Finalizado**	Medio
HU012	CRUDs para el modelo UserSkinsEspeciales	1	**Finalizado**	Medio
HU013	CRUDs para el modelo UserRopa	1	**Finalizado**	Medio
HU014	Levantar Cluster de MongoDB en MongoDB Atlas	1	**Finalizado**	Alto
HU015	Desarrollar la maqueta de la consola de control.	2	**Finalizado**	Media

HU016	Levantar el Menú de la Consola de Control	2	**Finalizado**	Alto
HU017	Pruebas unitarias del Backend	3	**Finalizado**	Alto
HU018	Desarrollar la sección de Login y recuperación de la clave.	3	**Finalizado**	Alto
HU019	Desarrollar la sección de Personajes y Consumibles	4	**Finalizado**	Medio
HU020	Desarrollar la sección de Special Skins y Ropa	4	**Finalizado**	Bajo
HU021	Desarrollar la sección de Administradores y Supervisores	4	**Finalizado**	Alto
HU022	Desarrollar la sección de Compras	5	**Finalizado**	Medio
HU023	Desarrollar la sección de Compras de Monedas	5	**Finalizado**	Alto
HU024	Pruebas de Integración	5	**Finalizado**	Alto
HU025	Configuración de SonarQube y Synk	6	**Finalizado**	Alto

HU026	Despliegue a Producción.	6	**Finalizado**	Alto

PROTOTIPO DE INTERFACES DE LA CONSOLA DE CONTROL

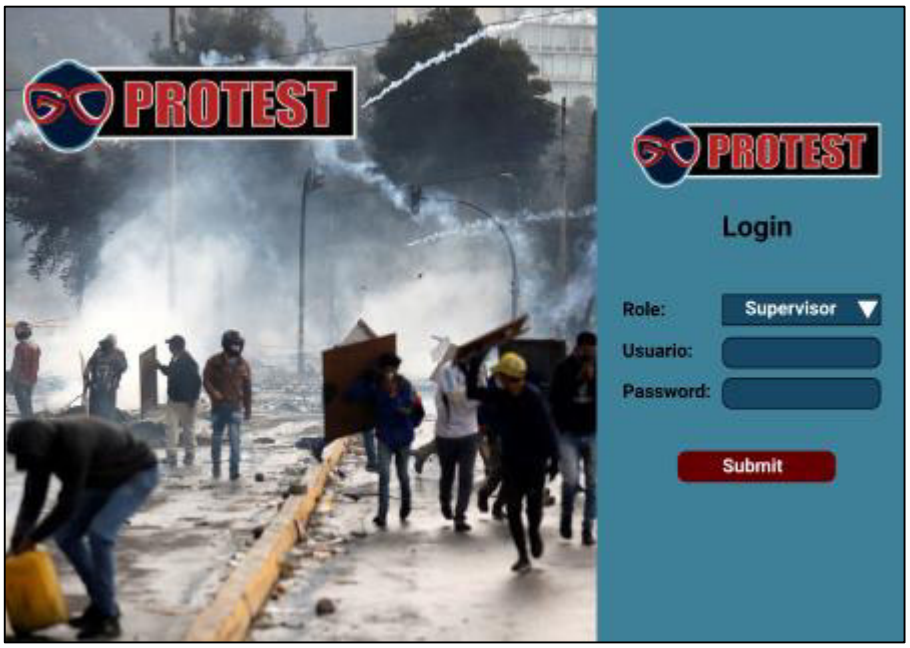


Fig. 63: Login

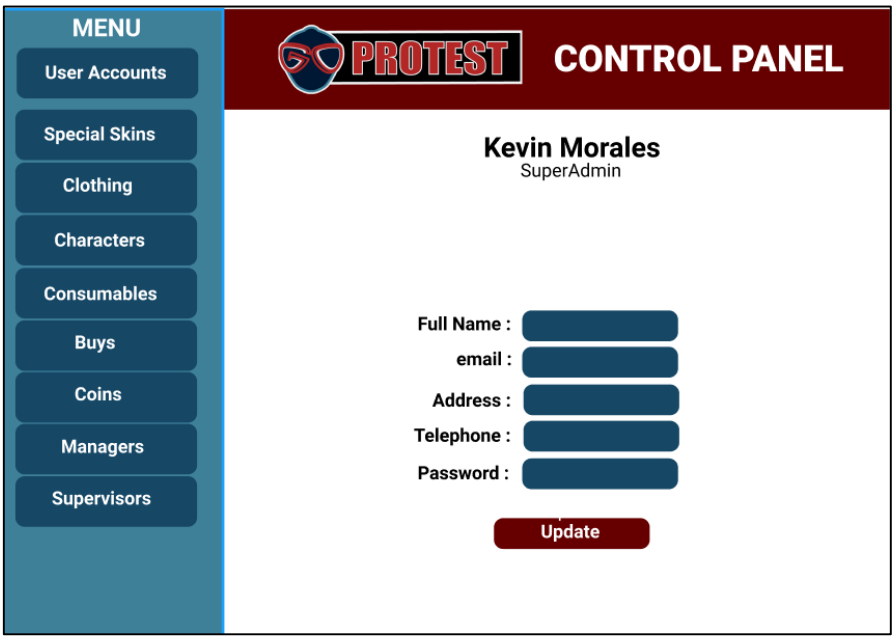


Fig. 64: Página Principal

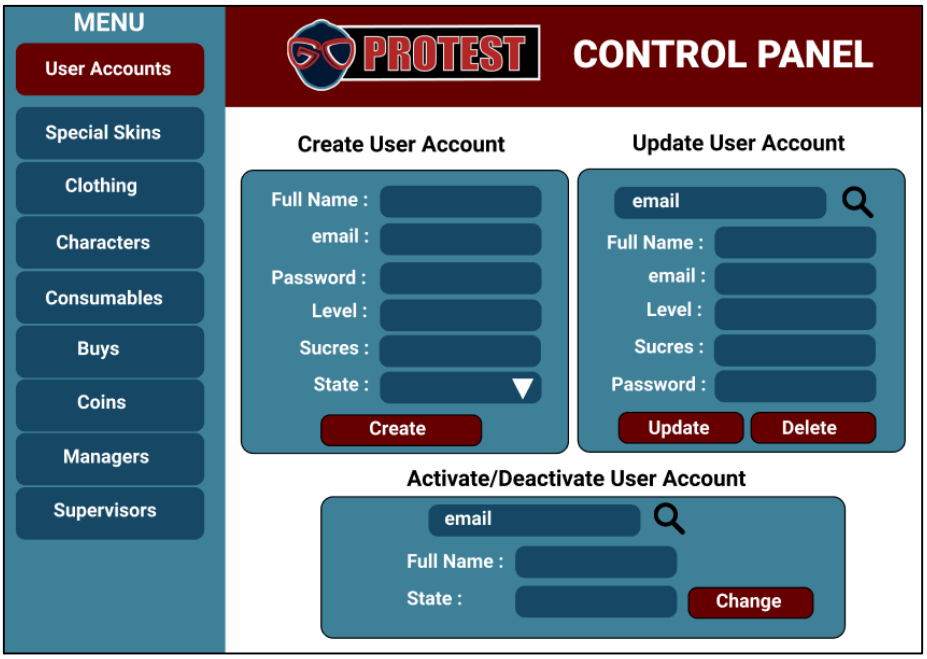


Fig. 65: Cuentas Usuarios

Fig. 66: Módulo de Ropa

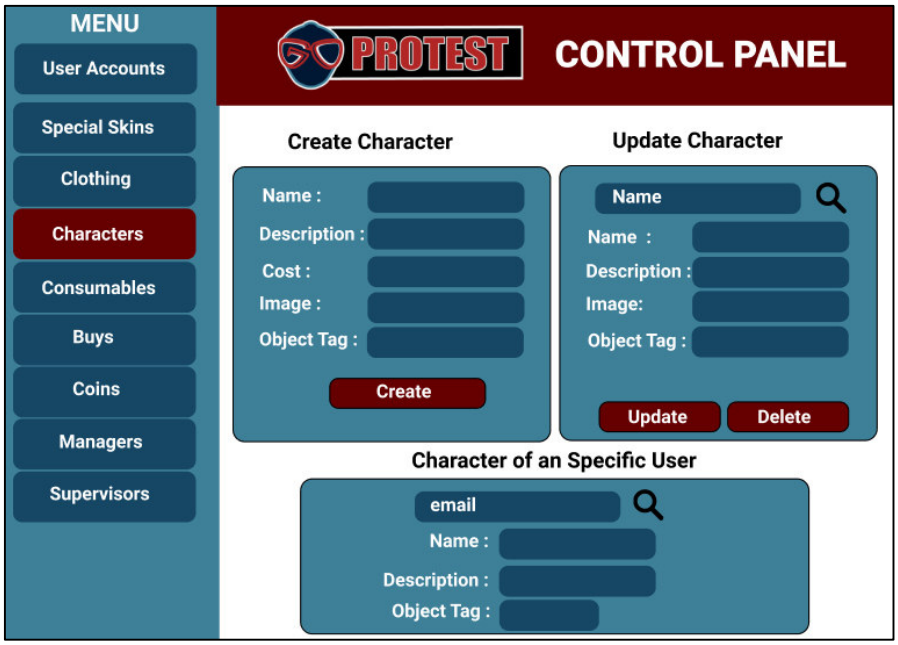


Fig. 67: Módulo de Personajes

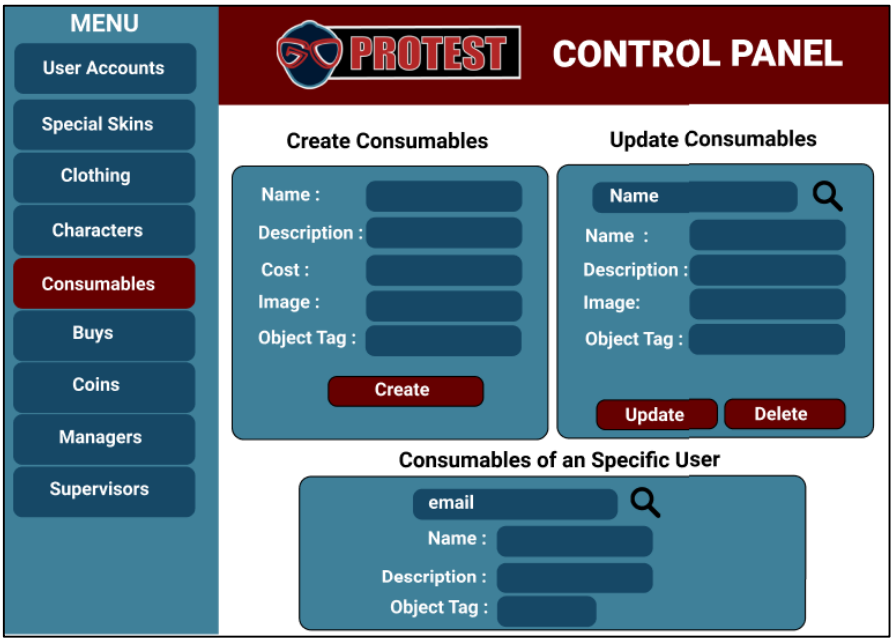


Fig. 68: Módulo de Consumibles

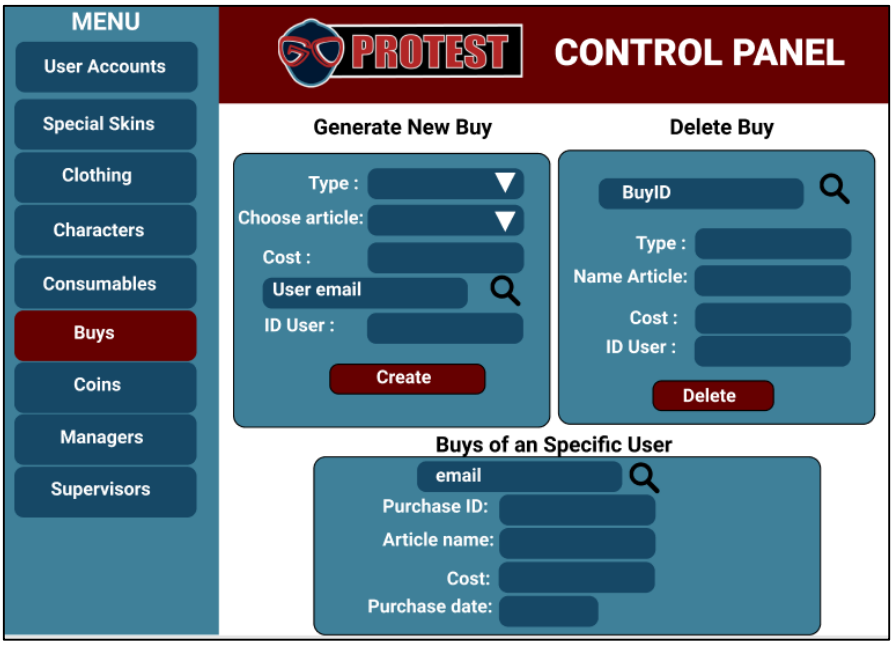


Fig. 69: Módulo de compras

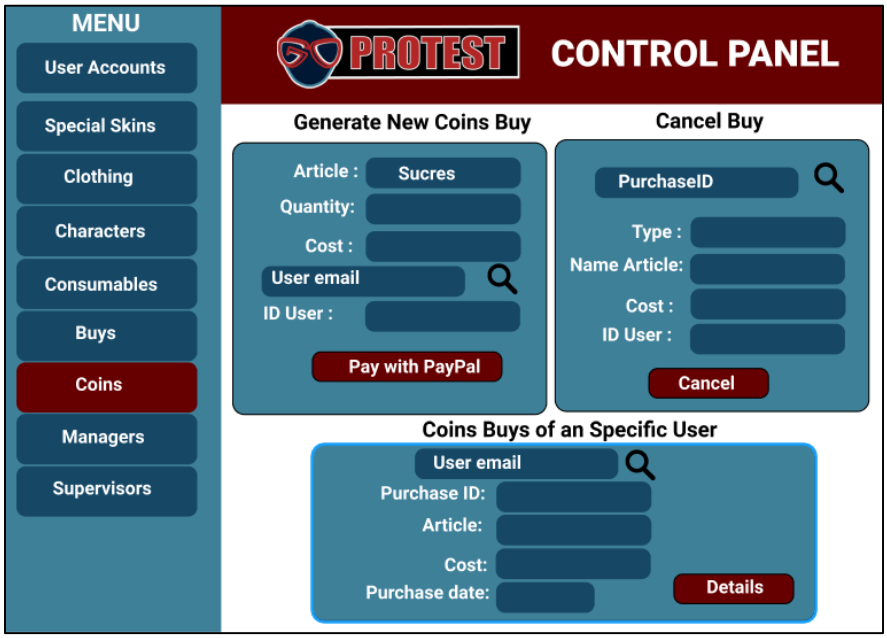


Fig. 70: Módulo de Compras de Sucres

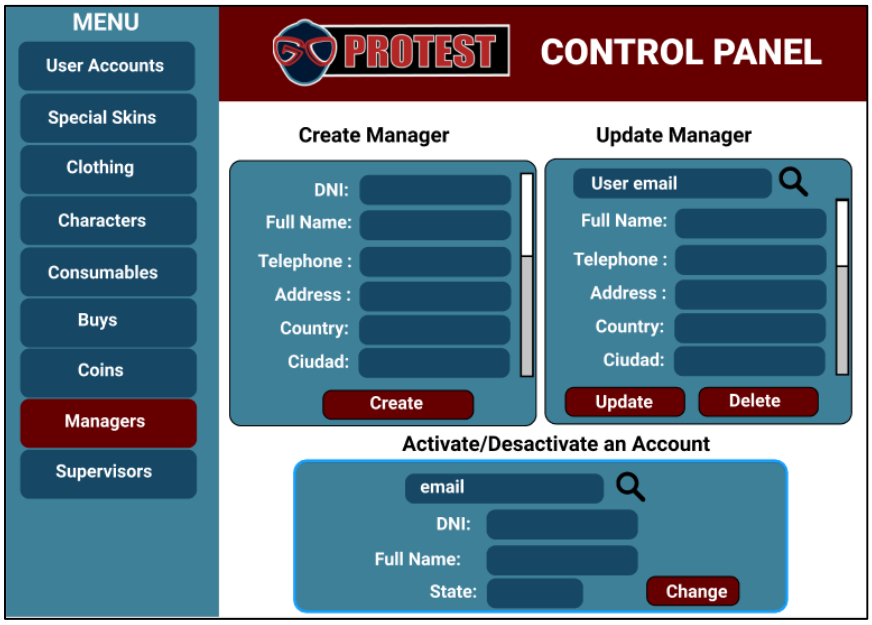


Fig. 71: Módulo de Gestion de Administradores

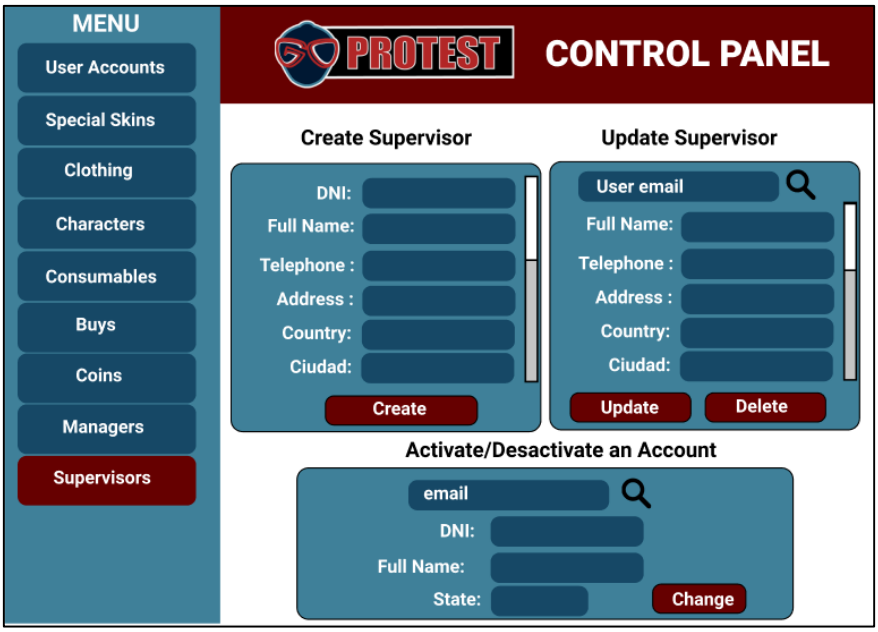


Fig. 72: Módulo de Gestión de Supervisores

ANEXO III. Manual de usuario

- Link del video: https://youtu.be/HpEMwZEx6eE
- Enlace de los mockups del proyecto. https://www.figma.com/file/K1gDPUYZ7eBdZuODU6u9ht/Go-Protest-Control-Panel?node-id=0%3A1
- Enlace de acceso a la aplicación en producción: https://tesis-tsds.web.app/

ANEXO IV. MANUAL DE INSTALACIÓN

- Enlace del repositorio de la Consola de Control:

https://github.com/kevinmorales05/console_goprotest.git

- Enlace del repositorio del *backend*:

https://gitlab.com/kevops/erp-goprotest-backend.git