APLICACIÓN WEB PARA RECOMENDACIÓN DE LIBROS PARA APOYAR LA ELECCIÓN DE COMPRA

JUAN PABLO GOMEZ QUINTERO

Institución Universitaria Politécnico Grancolombiano
Ingeniería de sistemas
Bogotá D.C., Colombia
2021

Índice

RESUMEN DEL PROYECTO:

Este proyecto describe el proceso de planeación , desarrollo y testeo de una aplicación web destinada para el apoyo de compra de libros, dicha aplicación , se describe que herramientas, lenguajes y tecnologías usadas(entre ellas están Mongodb, Reactjs, Java, etc), también se describe la metodología usada y como se desarrolló, se comparte el cronograma de actividades en el cual se observa cuanto tiempo tardo en implementarse cada una de las tareas asignadas que componen el desarrollo del software.

Palabras Claves: API, Scrum, Rating, Base de datos no relacional, Aplicación web, front-end, back-end.

1. TITULO DE LA PROPUESTA:

APLICACIÓN WEB PARA RECOMENDACIÓN DE LIBROS PARA APOYAR LA ELECCIÓN DE COMPRA

1. PLANTEAMIENTO DEL PROBLEMA:

Hoy en día los medios de entretenimiento representan una importancia considerable en la vida de las personas, ya sea música, videojuegos, películas, series o libros, sin embargo, debido a obligaciones personales las personas cuentan con un tiempo limitado para consumir dichos contenidos y son más dadas a invertir su tiempo en contenido del que conozcan información de entrada y referencias de otros usuarios para validar su calidad.

Para ello se propone una aplicación web que permite que los usuarios consulten y publiquen sus opiniones y calificaciones sobre una variedad de libros, adicionalmente

habrá información de entrada de cada uno de ellos con el fin de facilitar la decisión de los usuarios para que puedan escoger fácilmente qué libro se acomoda más a sus gustos.

2. OBJETIVOS

3.1 **OBJETIVO GENERAL**:

Desarrollar una aplicación Web que permita la recomendación de libros a partir de las opiniones de lectores

3.2. **OBJETIVOS ESPECÍFICOS**

- Implementar una Base de datos que contenga información básica de libros sugeridos por los usuarios de la aplicación.
- Implementar capas de lógica y de presentación para permitir el registro y la consulta de opiniones de los lectores sobre los libros.
- Realizar pruebas unitarias

ALCANCE: El aplicativo tendrá la capacidad de almacenar información de entrada de cada uno de los libros, dicha información consiste en el nombre, autor, género, plataformas en las que está disponible, sinopsis y su portada, la adición de nuevos libros a la base de datos dependerá de sugerencias por parte de los usuarios, dicha sugerencia tendrá como información el nombre del libro que se quiere agregar el sistema, en caso de ser aceptado el administrador deberá añadir el libro en el sistema

Los usuarios podrán dejar un comentario con sus opiniones de cada uno de los libros existentes, junto a ese comentario debe ir una calificación de 0 a 5. Los libros cuentan con un “rating” que se calculará como el promedio de calificaciones dadas por los

usuarios. Dicha información se usará para mostrar los libros mejor recibidos por los usuarios de cada género.

3. JUSTIFICACIÓN:

El software propuesto le da al usuario un conjunto de información que le puede ayudar a elegir en qué libro debería invertir su tiempo, además le permite comparar calificaciones para saber qué libros tuvieron mejor acogida, adicionalmente información como las calificaciones y opiniones de los demás usuarios puede ser usada para propósitos de marketing por los escritores ya que la calificación de los usuarios puede medir la acogida que ha tenido su libro entre los lectores que usen el aplicativo.

Gracias a las tecnologías usadas, será posible construir un sistema robusto que cumpla con las necesidades del proyecto, en dichas tecnologías se incluye el uso de Spring framework (*Spring Framework*, 2021) node.js,express, MongoDb (*¿Qué es MongoDB*, 2021) y Reactjs (*React*, 2021.). Adicionalmente el manejo de la metodología ágil SCRUM garantizará entregas incrementales y de valor (*What is Scrum*, 2021).

4. MARCO TEÓRICO

Aplicación web

Una aplicación web es un programa el cual requiere de un servidor remoto para ser desplegado y se ejecutan usando exploradores de internet como por ejemplo Google Chrome, Opera, Etc, lo que representa una ventaja ya que el software se podrá ejecutar sin importar el sistema operativo que el dispositivo maneje. Cualquier aplicación que realice una función que sea más que mostrar información al usuario, es una aplicación web, dichas funcionalidades pueden ser diversas como por ejemplo calculadoras online, aplicaciones de mail, tiendas online, redes sociales, etc *Web application(Web app)*(AGO/2019)

Este tipo de software usualmente usa lenguajes de programación como JavaScript y HTML y los elementos que los suelen componer son un servidor web que se encarga de recibir las peticiones del cliente, un servidor de aplicación que se encarga de realizar las operaciones que se requieran y dependiendo de los requerimientos del software, una base de datos. Las aplicaciones web pueden ser dinámicas, lo que las hace requerir un servidor del cual reciben información y recursos necesarios para el funcionamiento de la aplicación o no dinámicas las cuales no requieren de un servidor para acceder a todas sus funcionalidades. La implementación de aplicaciones web ha incrementado debido al uso tan frecuente que tanto como empresas como usuarios le dan al internet, también se ha dado gracias a que las empresas han optado por desarrollos más relacionados con la nube, incrementa eficiencia , disminuye costos y adicionalmente al no ser exclusivo de ningún sistema operativo y al solo necesitar acceso a internet, aumenta considerablemente su alcance.*What is a web application?* (31/May/2016)

API REST

(Application Programming Interface) Es un mecanismo de acceso a recursos compartidos entre componentes de software, a diferencia de otros tipos de API's como SOAP o XML-RPC este tipo de API es compatible con una gran variedad de lenguajes y de tipos de datos, su comunicación se basa en distintos métodos que usan peticiones HTTP, los cuales permiten realizar una variedad de operaciones sobre un conjunto de datos *API REST(06/04/2021)*, dichos métodos se describen a continuación:

- **POST:** Se usa para insertar datos desde un cliente a un servidor.
- **GET:** Se usa para consultar **datos desde un cliente a un servidor.**
- **PUT:** Se usa para consultar **datos desde un cliente a un servidor.**
- **DELETE:** Se usa para eliminar **datos desde un cliente a un servidor.**

Para la correcta implementación de este tipo de sistemas es importante tener en cuenta los siguientes principios:

1. **Interfaz uniforme:** Los datos deben corresponder a una URI(Uniform Resource Identifier) única, todas las solicitudes a un recurso sin importar su origen deben ser iguales.
2. **Separación entre cliente y servidor:** los componentes de software que representan al cliente y servidor deben de ser capaces de comunicarse usando únicamente la URI del recurso solicitado**,** dichos componentes deben ser totalmente independientes, las únicas vías de comunicación deberían ser los datos y las URI's.

3. **Sin estado:** las API REST no almacenan información de ninguna petición que se realice, lo que significa que cada petición debe contener toda la información necesaria para llevar a cabo la operación.
4. **Capacidad de almacenamiento en memoria caché**: Es preferible que el almacenamiento en memoria cache de los datos tanto en el cliente como en el servidor esté activado
5. **Arquitectura de sistema por capas**: Los sistemas construidos usando esta API están conformados por una variedad de capas que procesan los datos de diversas maneras, dichas capas se deben poder implementar sin afectar el funcionamiento del cliente o servidor. *API REST(06/04/2021).*

Este tipo de mecanismo cuenta con ciertos beneficios a considerar a la hora de decidir si se incluye en proyectos de software. Las API's REST son muy flexibles y son capaces de manejar una gran variedad de tipos de petición en distintos tipos de formatos, además son escalables ya que conforme el sistema de información crece en complejidad y cantidad, el rendimiento de su funcionamiento no se ve afectado *REST APIs*(27/Jul/2020)

PRINCIPIOS SOLID

Son un conjunto de buenas prácticas a la hora de escribir código, seguir dichas prácticas nos ayuda a construir proyectos de software más eficientes, mantenibles, de fácil extensión y de menor complejidad. Adicionalmente son un conjunto de principios que ayudan a los desarrolladores a mejorar aspectos del desarrollo de software referente a mantenimiento y extensibilidad SOLID: The First 5 Principles of Object Oriented Design(21/Sep/2020)

Su nombre está conformado por siglas que representan cada uno de los 5 principios a seguir:

- **Single responsability:**

  Cada clase que compone nuestro software debe tener una sola responsabilidad, se debe evitar crear clases que se encarguen de más de una tarea a la vez, seguir este principio nos ayuda a que los procesos de testing sean más sencillos, reduce el acoplamiento lo que quiere decir que tendremos menos dependencias entre nuestros componentes y hace el proceso de organización de archivos en proyectos grandes más fácil

- **Open for extention, closed for modification:**

  Todas las clases que creemos deben estar diseñadas para que cuando se requiera añadir o expandir una funcionalidad, no exista la necesidad de modificar una clase existente y se pueda implementar mediante la adición de código nuevo, la única excepción es la resolución de errores en código existente.

- **Sustitución de Liskov:**

  Cuando se tiene una clase que es un subtipo de una clase padre, deberíamos ser capaces de intercambiarlas sin interrumpir el funcionamiento de un programa, esto garantiza que todas las clases que hereden sus comportamientos de una clase padre se comporten de la misma manera.

- **Segregación de interfaces**

  las interfaces que contengan una gran cantidad de métodos deberían ser dividas en varias interfaces, con el fin de asegurarnos que las clases que

implementas dichas interfaces solo se preocupan por los métodos que hacen parte de su funcionalidad.

- **Inversión de dependencias**

  Los módulos de alto nivel no deberían de depender de módulos de bajo nivel, en cambio ambos deben depender de abstracciones, con el fin de aumentar la modularidad y poder intercambiar componentes sin irrumpir en el funcionamiento del programa.*A solid Guide to SOLID* (18/05/2021)

Patrones de diseño

Son conceptos de programación que tienen como propósito solucionar problemas comunes que se suelen presentar a la hora de implementar código. Un patrón de diseño no describe un código que sea una solución concreta a algún problema, ya que la implementación de la solución depende del desarrollador, si no que describe su funcionalidad y como debe ser el resultado.

Los patrones de diseño se clasifican en 3 tipos en función de para qué se usan:

- Patrones creacionales: incluye todos los patrones que se encargan de la creación de objetos, se suelen usar para optimizar y reutilizar código.
- Patrones estructurales: incluye los patrones que se encargan de definir la estructura entre objetos y clases para disminuir el acoplamiento.
- Patrones de comportamiento: incluye los patrones que se encargan de asignar responsabilidades a los componentes. PATRONES *DE DISEÑO (2021)*

Un ejemplo de patrón de diseño es el patrón singleton, el cual es un patrón creacional que permite al desarrollador asegurar que una clase determinada tenga solo una única instancia durante la ejecución del código, así como también crear un punto de acceso a ella, esto con el fin de evitar la sobrecarga de llamadas a un servicio como puede ser una base de datos además de evitar que alguna otra clase modifique su estado. Para implementar esta solución es necesario que la clase cuente con dos elementos, un constructor privado que evite que otras clases creen nuevas instancias de la clase singleton y un método de creación que se encarga de crear la instancia de la clase la primera vez que sea llamado y de devolver la instancia guardada(la cual es solo una durante toda la ejecución) el resto de veces que se necesite la clase. Shvets Alexander *(2019)*

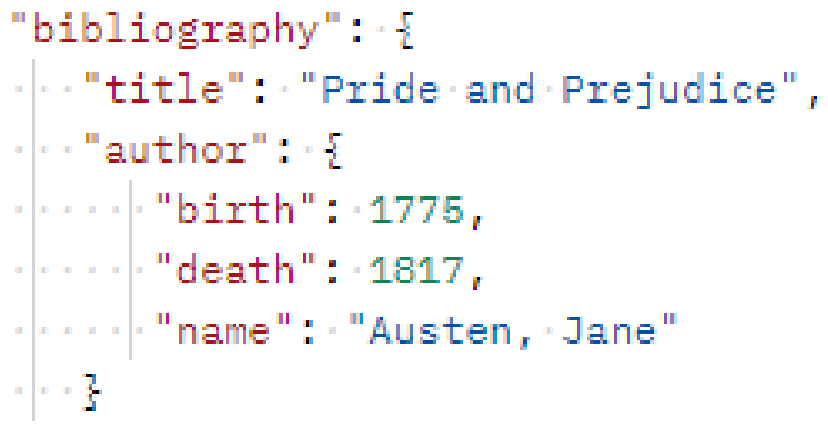
Bases de datos no relacionales

Las bases de datos no relacionales almacenan información de una manera muy distinta a la que lo hacen las bases de datos relacionales, por ejemplo en las bases de datos relacionales se almacenan datos en forma de tablas las cuales se relacionan mediante el uso de llaves foráneas, las cuales requieren un mayor uso de memoria, y si se requieren escalare, solo se puede hacer verticalmente, la que puede representar una limitación de almacenamiento por otro lado, las bases de datos no relacionales guardan su información en forma de documentos que generalmente están en formato JSON los cuales pueden almacenar información de varios formatos y al no depender de ningún otra entidad, permiten mayor flexibilidad y ofrecen mayor eficiencia a la hora de hacer consultas ya que mientras una consulta en una base de datos relacional debe acceder a varias tablas, en el caso de una

base de datos no relacional solo debe consultar un documento en el cual se debe encontrar toda la información de la entidad que se esté modelando.

Figura 1

Estructura de un documento en formato JSON



Nota. El gráfico muestra cómo se usa un documento en formato JSON para almacenar datos de una entidad. Fuente: Elaboración propia.

Las bases de datos no relacionales cuentan con una serie de ventajas que son importantes considerar a la hora de construir software. Algunas de estas ventajas se mencionan a continuación:

- **Organización masiva de conjunto de datos**: hoy en día en la industria del desarrollo del software es normal encontrar requerimientos que incluyan manejo de un gran volumen de datos, las bases de datos no relacionales son capaces de almacenar grandes cantidades de datos de muchas variedades, adicionalmente son capaces de realizar peticiones a dichos datos de una forma eficiente.
- **Tipos de datos flexibles:** Es posible hacer que nuestro sistema de información crezca conforme la información lo haga de una manera fácil.

- **Múltiples estructuras de datos:** En una base de datos no relacional es posible guardar datos de una gran variedad de tipos, desde Strings, hasta imágenes y video. *What is a non-Relational Database?*(2021)

Git

- Es un sistema de control de versiones que permite llevar un registro de todas las versiones que existan de un software, esto con el propósito de facilitar la resolución de problemas y facilitar el trabajo en equipo, en git es posible crear ramas que representan un "estado" del código que estemos manejando y son independientes a el resto del proyecto, lo que permite realizar cambios, añadir funcionalidades y experimentar con ideas y soluciones nuevas sin preocuparse por interferir en el funcionamiento de todo el proyecto, así como también volver a versiones anteriores en caso de ocurrir algún error. Ya que se puede decidir si una rama se "mezcla" con el resto del trabajo o simplemente se elimina en caso de no haber funcionado. *About*(s.f) git

Unit Testing

Es una práctica en la cual se usa para hacer pruebas relacionadas con el correcto funcionamiento de una unidad de código, se define unidad de código como una parte de código que se pueda separar lógicamente, lo que por lo general suele ser una función o método. Es objetivo del unit testing es facilitar el proceso de pruebas, ahorrando tiempo al no tener que verificar funcionamiento manualmente. Existen librerías como Junit y chai, las

cuales son usadas en el proyecto, que contienen métodos de comparación y otras utilidades de testing. *What is unit Testing*(2021)

Figura 2

Ejemplo básico del concepto de unit testing

```
fun sumar(a,b){
   return a+b;

}

fun testSumar(a,b){
   a=1;
   b=2;
 assertEquals(3,sumar(a,b))
}
```

Nota. Ejemplo de unit testing básico en el cual se verifica el funcionamiento de una función que suma dos números, el método assertEquals determina si el resultado retornado por la función sumar es igual a 3. Fuente: Elaboración propia

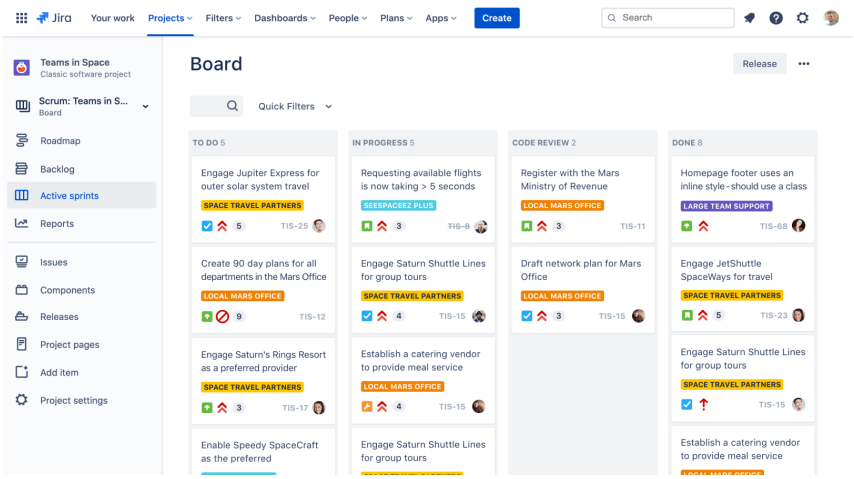
Con el fin de implementar correctamente unit testing se debe entender lo que es un objeto mock, de su traducción del inglés “imitar algo” ,dichos objetos se encargan de dar a las funciones las variables y objetos necesarios para su funcionamiento, son objetos y variables que simulan un caso de uso específico para saber si el programa responde correctamente en ciertos escenarios. Aplicar unit testing a métodos que hacen parte de una API es muy útil ya que nos ayuda a entender el funcionamiento de cada uno de ellos y en caso de existir algún cambio que genere un error, se puede identificar rápidamente para ser corregido. *Unit Testing Techniques and Best Practices* (28/OCT/2021)

Jira

Es una herramienta de gestión que permite crear historias de usuario, planificar sprints y distribuir tareas entre el equipo de desarrollo, cuenta con un tablero el cual organiza las tareas en función de su estado (por hacer, en progreso, hecho) con la posibilidad de crear estados personalizados, la herramienta adicionalmente crea informes con información útil a cerca del rendimiento del equipo. *La herramienta de desarrollo de software líder de los equipos ágiles (2021)*

Figura 3

Página en la cual se puede administrar el tablero de tareas.



Nota JSW-tour-board. (2021). [Gráfico]. Jira Software.

https://www.atlassian.com/es/software/jira

Node.js

Es un entorno de ejecución de JavaScript creado en el 2009, que se caracteriza por ser rápido, escalable y que cuenta con una gran variedad de librerías gratis para implementar en los proyectos, es un entorno de ejecución que permite tener código javascript funcional afuera de un navegador, lo cual era imposible antes de su creación, su arquitectura usa v8, el motor de javascript de Google.

Node se ejecuta sin crear varios hilos a diferencia de otros entornos de ejecución como java, en cambio cuando se realiza una operación de entrada o salida, node continúa con su operación una vez su solicitud anterior devuelva una respuesta, esta característica lo hace una buena opción cuando se construyen aplicaciones en tiempo real. Una ventaja adicional de node es que al escribirse en código JavaScript el cual es utilizado comúnmente por desarrolladores front end, les permite transicionar de una manera muy fácil a escribir código back end sin la necesidad de aprender varios lenguajes de programación. *A brief history of Node.js* (2020)

5. METODOLOGÍA

SCRUM

Es un framework que ayuda a equipos a desarrollar cualquier tipo de producto, se basa en soluciones adaptativas que generan entregas incrementales, no se debe ver como un conjunto de instrucciones específicas que seguir, en cambio debe verse como una guía de trabajo en la cual se pueden adoptar los elementos que los equipos que lo implementen consideren útiles.

Scrum se basa en 4 principios indispensable para implementar este framework correctamente:

- **Transparencia:** Información sobre el progreso y el trabajo que se acuerde realizar debe ser clara para todos los involucrados, esto ayuda a que no se tomen decisiones sin tener en cuenta aspectos que puedan incrementar el riesgo.
- **Inspección:** Scrum está diseñado para aumentar la inspección mediante los eventos que se llevan a cabo, con el fin de llevar un buen registro de problemas y avances importantes para el proyecto.
- **Adaptación:** Es muy común que al realizar alguna actividad necesaria para completar algún proyecto se encuentren aspectos a mejorar, para lo cual es

importante que el equipo cuente con la capacidad de identificar y poner a prueba los nuevos aspectos aprendidos que se deben tener en cuenta para mejorar la efectividad de trabajo

Equipo Scrum

Es un equipo conformado habitualmente por 10 o menos personas las cuales cuentan con las capacidades necesarias para añadir valor a cada entrega que se realice, son capaces de tomar decisiones y de determinar que se hace y quien lo hace, son responsables de las actividades relacionadas con el proyecto ya sea mantenimiento, operación, investigación y desarrollo, está conformado por un producto owner, scrum master y los desarrolladores.

Desarrolladores

Son los responsables de crear entregas con valor para cada sprint, adicionalmente deben ocuparse de otras tareas importantes para el desarrollo como la planeación de cada sprint y la creación del product backlog

Product Owner

Su responsabilidad es maximizar el valor de las entregas mediante el correcto manejo de la información que hace referencia a las tareas del proyecto, para llevar esto acabo es importante que el product owner sea claro con cada uno de los elementos del backlog y con el objetivo de la entrega.

Scrum master

Su responsabilidad es asegurar que todo el equipo aplique y entienda correctamente el framework SCRUM y también eliminar impedimentos que pueda tener el equipo de desarrollo que les impida avanzar con efectividad también es conocido como orquestador por su capacidad de liderar un equipo complejo.

Eventos scrum

Sprint planning: Es la reunión que da inicio a un sprint en la cual se establecen las tareas que se realizaran en el sprint que está apunto de empezar. Dura 8 horas para un sprint de un mes, pero puede durar menos de ser necesario.

Sprint: Es el intervalo de tiempo en el cual las ideas plasmadas en el product backlog son hechas realidad, es posible modificar dicho tiempo dependiendo de lo que se aprenda conciliándolo con el product owner, una vez iniciado no se pueden cambiar las tareas establecidas. Se recomienda sprints cortos que permitan mantener los objetivos claros y la complejidad constante.

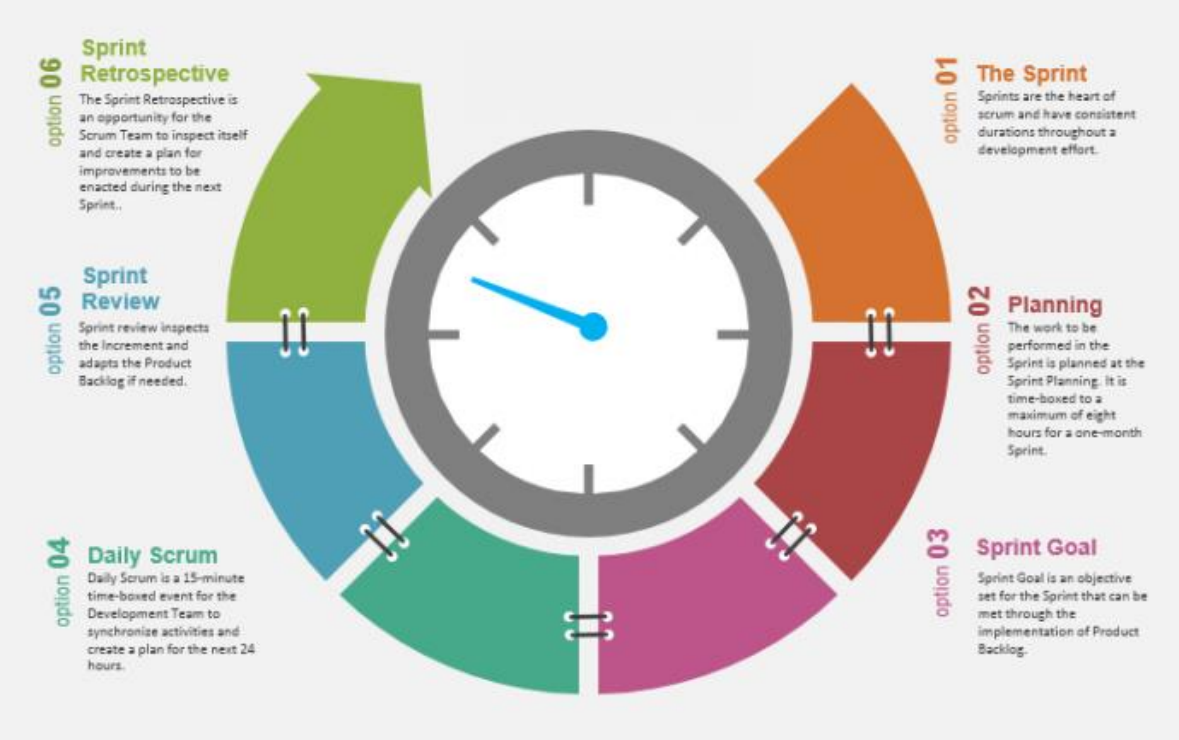
Daily Scrum: Es una reunión breve que dura máximo 15 minutos en la cual cada integrante del equipo da una pequeña actualización de el estado de su progreso en las tareas del sprint. Se realiza todos los días a la misma hora.

Sprint retrospective: En esta reunión se realiza al finalizar cada sprint y se reflexiona sobre el rendimiento del equipo a lo largo del sprint. Se habla acerca de lo que fue bien, de lo que hay que mejorar y que planes de acción se llevarán a cabo para solucionar los

problemas que se hayan presentado. Tiene una duración de máximo 3 horas, pero suele ser más corta. *The 2020 Scrum Guide* (2020)

Figura 4

Gráfica mostrando los diferentes eventos de Scrum.



Nota. Scrum-Events. (2017, 31 octubre). [Gráfico]. Eventos en scrum.

https://www.saraclip.com/eventos-en-scrum/

6. RESULTADOS:

La primera etapa de la práctica consistió en una serie de capacitaciones realizadas por expertos de la empresa en base a ciertos temas y principios valiosos para el desarrollo de software, algunos de estos temas fueron incluidos en la sección de marco teórico, a continuación, se tiene la bitácora de sesiones, la cual contiene un resumen de cada uno de los temas tratados en las sesiones realizada en conjunto por los asistentes.

Bitacora de sesiones.docx

BITACORA DE SESIONES PROGRAMA DE PRACTICANTES INTERTEC COLOMBIA

FECHA	OBSERVACIONES	REVISADO POR
Julio 8, 2021	Primera versión del documento	Jose Fabio Jaramilo Castro

SESION #1, Principios SOLID

- Aprendimos la importancia de la responsabilidad única y la fuerte relación que tiene esta con el acoplamiento, y como es importante buscar una relación de bajo acoplamiento y alta cohesión.

- Aprendimos a diferenciar los mejores y peores casos posibles relacionando la cohesión con el acoplamiento.
- Aprendimos las principales características de cada principio:
    - El principio de responsabilidad única nos indica que una clase debe tener solo una responsabilidad a cargo.
    - El principio abierto-cerrado nos indica que las clases o módulos deberían estar abiertas a extensión/expansión, pero cerrados a ser modificados.
    - En la sustitución de Liskov nos dejan aun mas claro la diferencia entre herencia y polimorfismo.

- El día de hoy hablamos sobre los principios SOLID, donde cada letra representa una buena práctica en programación que nos brinda una serie de beneficios a la hora de desarrollar software, estos principios se aplican sin importar el lenguaje de programación que se esté manejando:
    - El principio de responsabilidad única se refiere a que cada clase debería ser responsable de una sola tarea y debe tener solo una razón para ser modificada, los beneficios que nos trae es que al tener solo una razón para ser modificada, nos reduce el acoplamiento(la dependencia entre módulos),nos ayuda a la organización y nos disminuye los casos de prueba
    - El principio de abierto-cerrado nos dice que una clase debería únicamente ser extendida o expandida, pero no modificada, uno de sus beneficios es disminuir el efecto de la paradoja del pesticida, la cual se refiere a que si modificamos un método y no cambiamos las pruebas que se realizan, las pruebas pueden perder efectividad, así como las ratas se adaptan a un tipo de veneno y este les afecta menos.
    - El principio de sustitución de liskov, nos dice que las clases hijas deberían poder hacer lo mismo que su clase padre y adicionalmente algo especifico de ellas, nos ayuda a aplicar correctamente la herencia y el polimorfismo(hay que recordar que no son lo mismo, con la herencia pasamos todos los comportamientos y cualidades mientras que con polimorfismo definimos un comportamiento dependiendo de el objeto)
    - El principio de segregación de interfaces se refiere a disminuir la cantidad de interfaces dividiéndolas en interfaces más pequeñas y específicas así no obligamos a las clases a utilizar métodos que no necesitan, esto nos beneficia a la hora de hacer testing.
    - El principio de inversión de dependencias se refiere a que en vez de tener módulos de alto nivel dependiendo de módulos de bajo nivel, deberíamos tener módulos de alto y bajo nivel que dependan de abstracciones de dichos módulos, esto nos ayuda a conseguir un bajo acoplamiento ya que si debemos hacer un cambio en alguna clase, solo debemos modificar esa clase y no todas las clases relacionadas con ella.
- Los principios SOLID son un conjunto de reglas y buenas prácticas. Estos son:
    - Single Responsability Principle: Entregar a cada módulo solo las tareas que le corresponden.

- Open/Close Principle: Una clase u objeto debe estar disponible para su extensión, pero no para su modificación.
- Liskov Substitution Principle: Si se sutituye una clase por su clase hija, el comportamiento del programa no debería cambiar.
- Interface Segregation Principle: Cuando se definen interfaces, estas deben cumplir con una finalidad concreta. Es preferible tener muchas interfaces que definen pocos métodos a tener una interfaz con muchos métodos.
- Dependency Inversion Principle: El código no debería depender de los detalles de implementación, como puede ser el framework o la base de datos.

SESION #2, Design Patterns, parte 1

- Al principio de esta sesión hablamos de cómo organizar el código de un proyecto haciendo uso de paquetes, también de cómo funciona su organización y nomenclatura
- Realizamos un repaso general de la definición de Patrones de diseño (Creacional, Estructural y de comportamiento) centrándose en el uso general que se les da, las razones pasa usarlos y la estrategia para saber reconocer cual es el idóneo para el proyecto en el que estemos trabajando.
- En la segunda parte de esta sesión hablamos sobre que son los patrones de diseño y en qué consisten dos de ellos.
  - Los patrones son soluciones para problemas comunes en el desarrollo de software, debemos saber cuál usar dependiendo de nuestro contexto actual como si fuera una caja de herramientas
  - El Factory pattern es un patrón de creación y lo que busca es facilitar la creación de objetos nuevos en un sistema, y que no tengamos que modificar las clases cliente para lograrlo.
  - El singleton patter es un patrón de creación y lo que busca es que dependiendo de nuestro contexto solo tengamos una instancia de una clase concreta (por ej una clase encargada de administrar las conexiones a una base de datos. La implementación puede cambiar si estamos trabajando con hilos en nuestro proyecto

SESION #3, Design Patterns, parte 2

- Al inicio de esta sesión se realizó una revisión del código realizado en base al patrón Singleton, hicimos un análisis de dos versiones diferentes, feedback sobre ciertos detalles en cada código y como paso final la revisión utilizando la depuración de programas con el fin de confirmar su correcto funcionamiento.
- Facade pattern: nos sirve para exponer una funcionalidad de una manera simplificada, esto nos sirve cuando tenemos un conjunto de clases con sistemas complicados y ya definidos y queremos extraer parte de su funcionalidad de manera simplificada para usarla en otro módulo.

- Adapter: Este patrón permite que dos clases puedan trabajar juntas pese a que tienen una interfaz diferente. Básicamente convertirá la interfaz en otra siempre y cuando sea lo que el cliente espera.
- Continuamos con la explicación de los patrones, esta vez centrándonos en otros tres (Facade, Adaptor y Strategy) con ejemplos prácticos, además de buscar la manera como ejercicio practico de implementarlo en el trabajo en grupo que estamos desarrollando.

SESION #4, Design Patterns, parte 3

- En la sesión de hoy hablamos de los dos últimos patrones de diseño: Patrón observer en cual se usa mucho en frontend y el cual nos sirve para tener objetos "observadores" los cuales serán notificados si ocurre un cambio en una clase "objeto", adicionalmente hablamos del patrón Decorator nos ayuda a agregar nuevas funcionalidades un objeto poe medio de agregar extensiones a una clase base.
- Aprendimos los detalles y definiciones de los patrones Decorador y Observador, así mismo implementamos en el proyecto el decorador, lo que ayudo a ver mejor su funcionamiento y la forma correcta de implementarlo.
- En base a lo anterior pudimos corroborar que:
    - El patron Decorator permite añadir responsabilidades a un objeto en concreto, con el fin de generar cierta flexibilidad a las subclases.
    - El patron Observer es un tipo de suscripción existente entre el sujeto y un observador, con el fin de crear una dependencia entre objetos para que uno cambie su estado.

SESION #5, POO

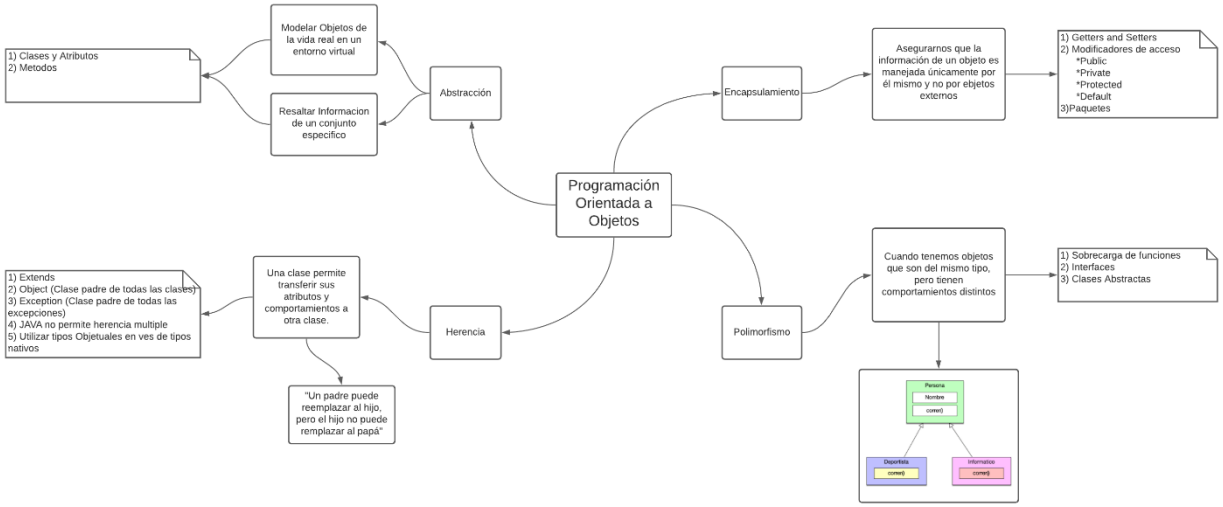


Ilustración 1 Link referencia: https://lucid.app/lucidchart/invitations/accept/inv_e371d603-d758-4455-bf15-07739ec84f42?viewport_loc=-787%2C56%2C2731%2C1117%2C0_0

- Iniciamos el tema de programación orientada a objetos (POO) centrándonos en los aspectos mas generales y las representaciones que se tienen en el lenguaje JAVA, creando de forma grupal el mapa conceptual.
- Revisamos el código realizado en clases anteriores y corregimos el manejo de atributos en las clases de las cuales heredan sus hijos , en dichas clases los atributos deben tener el modificador de acceso “protected” el cual permite que no sean modificados por clases externas pero que sus hijos si puedan heredar esos atributos. Adicionalmente en las clases hijas se debe hacer uso del constructor de su clase padre haciendo uso de la palabra reservada “super”.
- Revisamos como se pueden aplicar los 4 principios de la POO (Encapsulamiento, abstracción, herencia y polimorfismo) en Java. Además, se recomienda trabajar con tipos objetuales (Integer, Double, Float), que heredan de la clase Object, para poder hacer validaciones de null con los objetos.

SESION #6, collections

- Revisamos algunas de las colecciones más usadas en el lenguaje, así como características que ayudan a la hora de escoger cual es la indicada para un desarrollo, entre ellas vimos Vector, ArrayList, LinkedList y Queue.
- Una característica importante a la hora de elegir una colección es la velocidad a la que trabaja, ya que, si pensamos en manejar una gran cantidad de datos, puede que nos convenga más usar una que otra

SESION #7, API

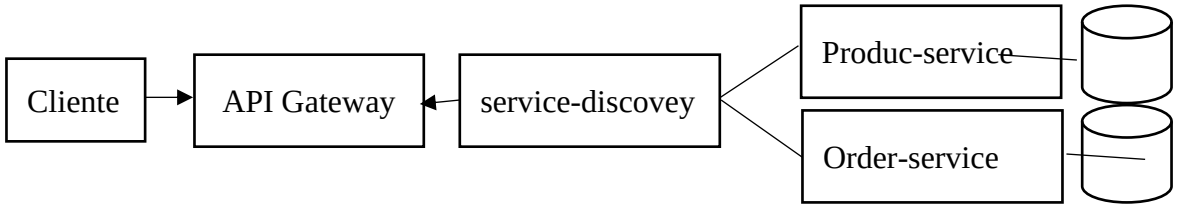
- Revisamos las definiciones generales de API y API REST, así mismo se repasó el tema de las peticiones http y como estas son planteadas en las historias de usuario.
- Se revisó como definir las historias de usuario, las historias deben referirse a tareas específicas de nuestro proyecto, por lo tanto, es necesario definir una historia para todo lo relacionado con la interfaz de usuario y otra historia para todo lo relacionado con operaciones backend, de esta forma es mucho más fácil repartir y dar por completadas las diferentes tareas de nuestro proyecto

SESION #8, JAVA MULTITHREADING

- Vimos los principales conceptos de los hilos, así como su implementación y diferencia con procesos, también revisamos las diferentes implementaciones que se pueden crear en Java.
- Revisamos en aspectos generales el uso y codificación de los hilos, tanto los creados de forma manual como los que usan la API de java “Executor” para facilitar no solo el manejo de estos sino optimizar de mejor forma el código.
- Se habló sobre los beneficios de usar hilos, entre estos se encuentra el uso óptimo de los cores de un pc, esto nos permite ejecutar nuestros programas de una manera mucho más rápida
- Para implementar un programa concurrente se debe tener cuidado de no incurrir en alguna de las faltas que pueden existir como por ejemplo el “deadlock” el cual ocurre cuando dos hilos se quedan bloqueados ya que se están esperando mutuamente, otra falta típica de la programación concurrente es “starvation” el cual se refiere a cuando un hilo se queda esperando un recurso el cual nunca le llega, siendo imposible que complete su tarea.

SESION #9, MICROSERVICES

- para la construcción de microservicios hablamos un poco del contexto, el uso y sus buenas prácticas, razón por la que se hablo de los Monolithic applications, el cual es el diseño más común que hemos tratado hasta el momento. Diferenciamos los microservicios por su estructura y función, ayudándonos así a hacer más atómico nuestros desarrollos lo cual ayuda en el mantenimiento y actualización de funcionalidades.
- La importancia del CICD para la actualización de estos microservicios, así mismo el como un equipo suele trabajar en este tipo de desarrollos, haciendo más fácil el aplicar los principios SOLID en nuestras funcionalidades.
- Generamos un proyecto base con algunos microservicios básicos, el cómo comunicarlos y usarlos correctamente.



- Así mismo revisamos el patrón de diseño de los microservicios, al menos los mas usados como saga por coreografía o saga por orquestación, con sus ventajas y desventajas correspondientes.
- También mencionamos el uso de los CQRS (Command Query Responsability Segregation) para las funcionalidades de un microservicio.

SESION #10, REACTjs

- React es una librería para la construcción de interfaces de usuario, no se trata de un framework de frontend. Existen dos tipos de componentes en React:

  componente tipo contenedor: en estos componentes se incluye únicamente la lógica y los cálculos necesarios para obtener información importante para el componente

  componente de presentación: en estos componente se incluye únicamente los elementos que hacen referencia a la presentación de información en la interfaz de usuario, estos componentes no tienen acceso al funcionamiento de la lógica de negocio, simplemente preparan la información para mostrarla en pantalla

  react se basa en la creación de componentes que conforman las diferentes

funcionalidades de una aplicación, con el fin de tener la posibilidad de reutilizar dichos compontes en funcionalidades diferentes, en nuestro proyecto de react deberíamos tener separado en distintos paquetes los componentes de los contenedores, con el propósito de separar la lógica del negocio de los componentes de presentación, mejorando así la legibilidad del código.

- Algunas ventajas de React:
  - Genera un DOM virtual.
  - Es enfocado en componentes, por lo cual cuando se modifica una parte de una página web, solo se recarga el componente correspondiente a la parte modificada.
  - Posee una amplia comunidad.

- Características de React:
  - JSX - JavaScript Syntax Extension: Se usa con React para describir cómo debería verse la interfaz de usuario. Con JSX podemos escribir estructuras HTML en el mismo archivo que contiene código JavaScript.
  - Virtual DOM - VDOM: Es la representación del DOM "real" que React tiene en la memoria. La ventaja de esto es que es más rápido manipular el VDOM que el DOM, entonces, cuando el estado de un objeto cambia, el VDOM se actualiza y luego actualiza en el DOM real solo los objetos que cambiaron.

LEVANTAMIENTO DE REQUERIMIENTOS

La primera etapa de desarrollo consistió en generar las historias de usuario las cuales representan funcionalidades que debe llevar a cabo el software, deudas técnicas que surgen durante el desarrollo y tareas relacionadas con unit testing. Se han creado 92 historias que componen el desarrollo de varias versiones del software que manejan otro tipo de entretenimiento como videojuegos series y películas, las historias anteriormente mencionadas se dividen en 4 épicas, las cuales se mencionarán a continuación.

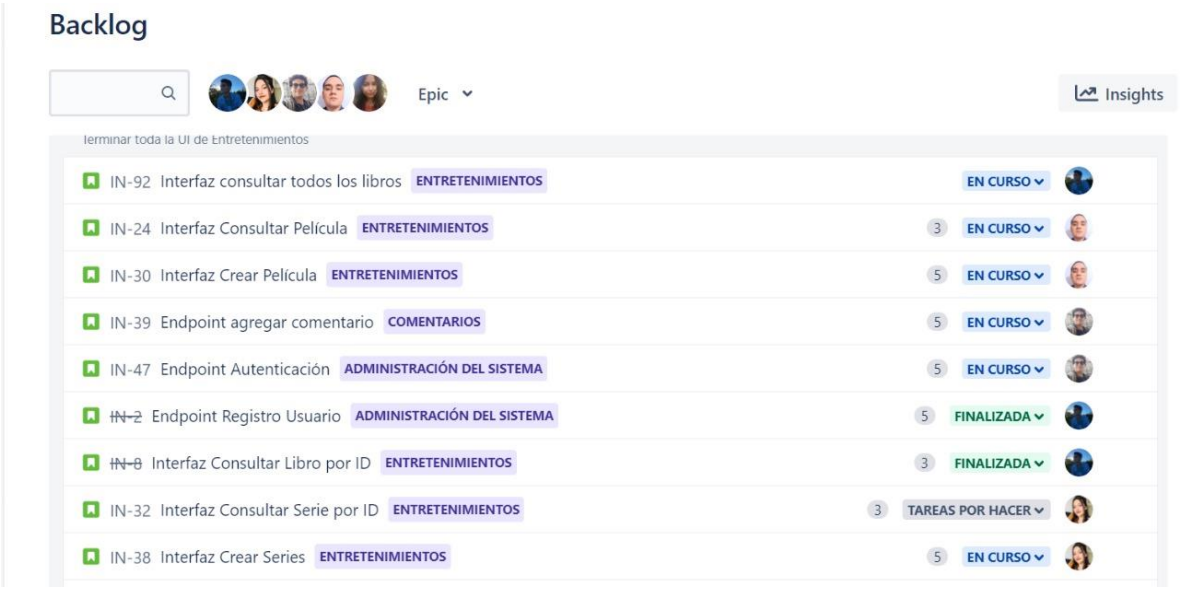
- **Deuda técnica:** Abarca historias que hacen referencia a correcciones en código existente.

- **Entretenimientos:** Se usa para registrar todas las funcionalidades relacionadas con algún entretenimiento en el aplicativo.
- **Administración del sistema:** Abarca todas las historias que hacen referencia a la administración del sistema como lo son la autentificación y el registro.
- **Sugerencias**: Abarca todas las historias de usuario relacionadas con el funcionamiento de las sugerencias de adición que los usuarios realicen.
- **Comentarios**: Abarca todas las historias de usuario relacionadas con el funcionamiento de los comentarios que los usuarios pueden realizar en los libros.

Adicionalmente se establecieron las tecnologías a usar para implementar cada uno de los componentes del aplicativo, las cuales son React.js para el frontend del aplicativo, java-Springboot para la implementación del Backend y Mongodb como servicio de base de datos no relacional. Debido a nuevos requerimientos que surgieron a lo largo del desarrollo se requirió implementar la API usando otra tecnología, node.js- express

Figura 5

Lista en la herramienta de gestión Jira de algunas de las historias de usuario que componen el backlog.



Nota. En la imagen se muestran algunas de las historias de usuario activas del sprint junto con su estado de finalización y puntos de historia, también se muestra el responsable de cada tarea. Fuente: Elaboración propia

Figura 6

Estructura de una de las historias de usuario asignadas.

·**El objeto JSON para la petición es:**

```
{
  "usuarioId": "",
  "contraseña": "",
  "apodo": "",
  "correo": ""
}
```

·El objeto JSON para la respuesta es**:**

```
{
  "mensaje": ""
}
```

·Si fue creado correctamente, deberá devolver el http code 201 y un mensaje de "Usuario creado correctamente"

·Si el usuario ya existe en el sistema, deberá devolver el http code 200 y un mensaje de "Usuario ya existente"

·Si hubo un error que genero una excepción, deberá devolver el http code 500 y un objeto JSON con la siguiente estructura:

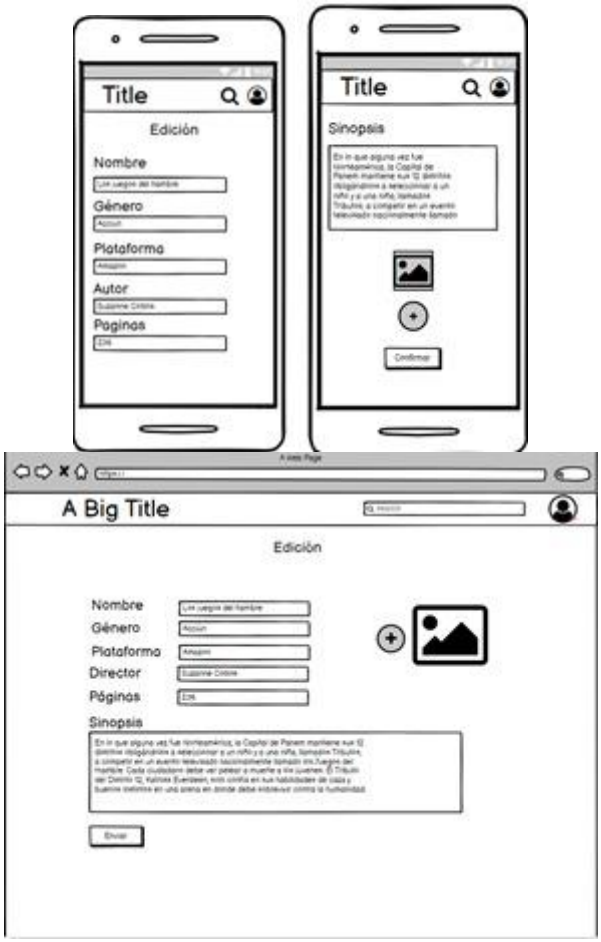
```
{
  "logID"": <Id del log generado con la excepción>
}
```

Nota. Descripción de una historia de usuario que detalla los requerimientos y características clave para determinada funcionalidad.

Adicionalmente se realizaron mock-ups que representan la apariencia de cada una de las páginas que compone la aplicación:

Figura 7

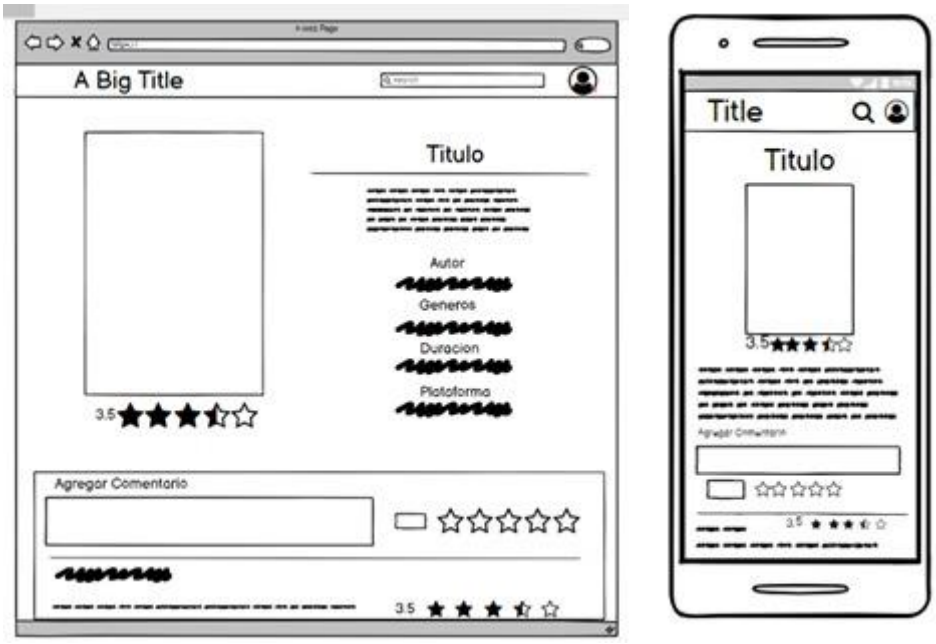
Interfaz de usuario para la edición y creación de un libro



Nota. Mock ups de referencia para el diseño de vista de escritorio y vista móvil de la interfaz de usuario de edición y creación de libros. Fuente: elaboración propia.

Figura 8

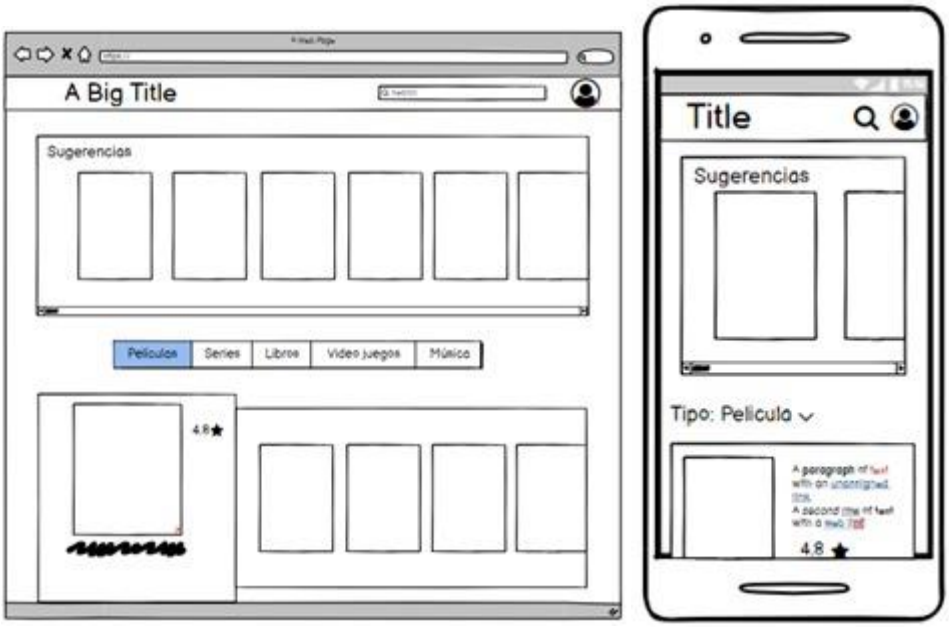
Interfaz de usuario consultar información de un libro.



Nota. Mock ups de referencia para el diseño de vista de escritorio y vista móvil de la interfaz consulta de información de un libro. Fuente: elaboración propia.

Figura 9

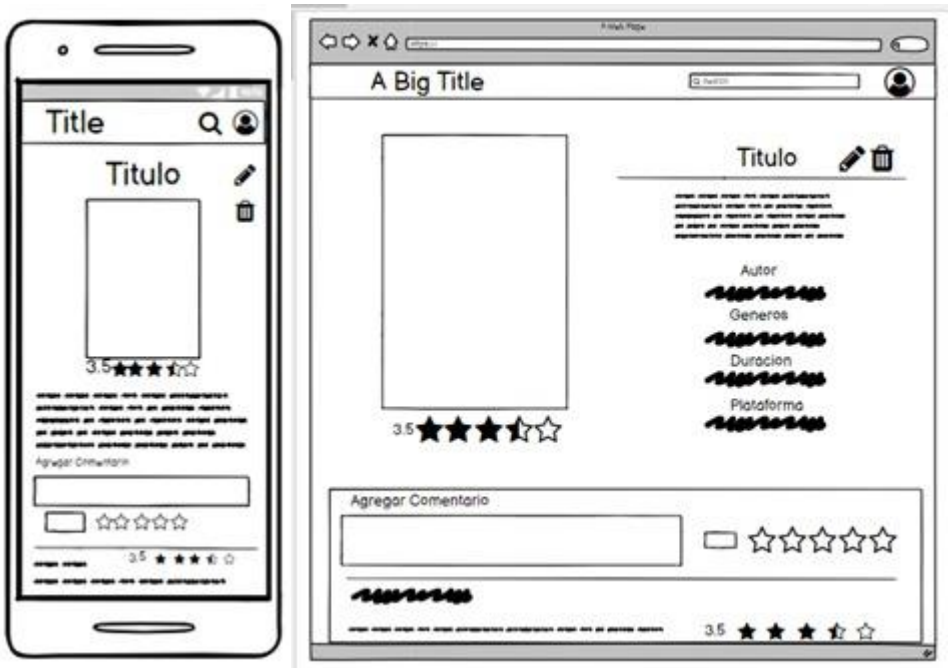
Interfaz de usuario consultar todos los libros.



Nota. Mock ups de referencia para el diseño de vista de escritorio y vista móvil de la interfaz consultar todos los libros. Fuente: elaboración propia.

Figura 10

Interfaz de usuario para eliminación de un libro



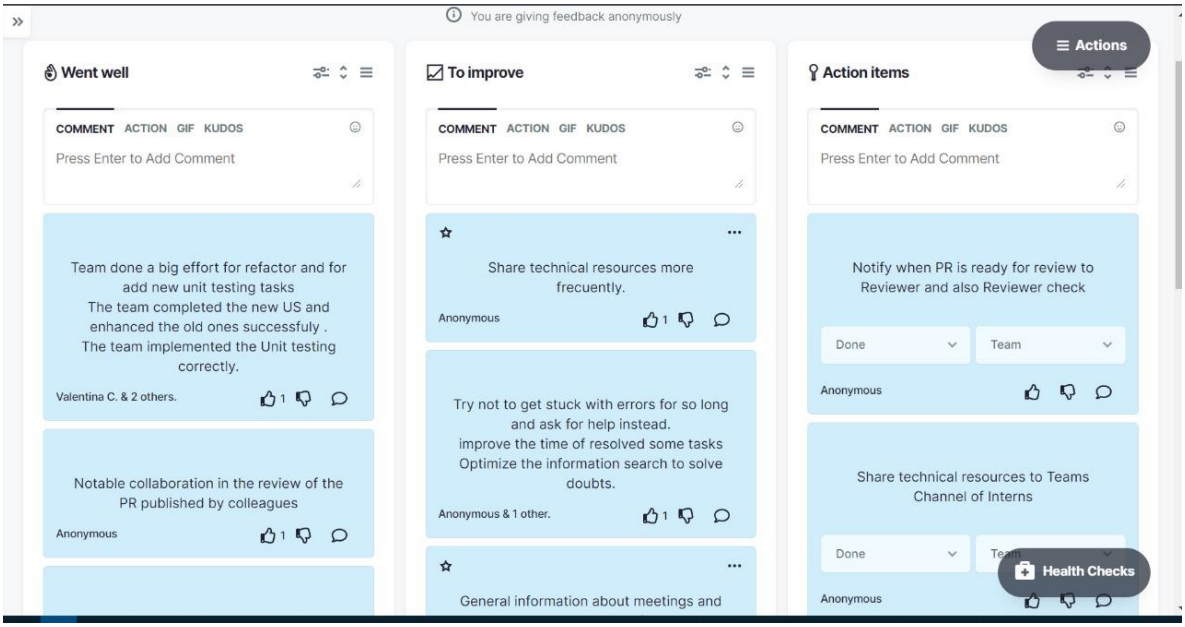
Nota. Mock ups de referencia para el diseño de vista de escritorio y vista móvil de la interfaz de usuario de eliminación de libros. Fuente: elaboración propia.

ACTIVIDADES SCRUM

- **Sprints**: Tienen una duración de 2 semanas, con una carga de aproximadamente 12 puntos de historia por miembro del equipo.
- **Daily Scrum**: Se llevan a cabo todos los días a las 9:45 AM Tiene una duración de 15 min, cada miembro del equipo se toma un minuto para responder 3 preguntas, ¿Qué hizo el último día?, ¿Qué tiene planeado hacer hoy?,¿Existe algún bloqueo que no le permita avanzar con las tareas? El tiempo sobrante se puede tomar para discutir dudas o inquietudes.
- **Sprint planning**: Se llevan a cabo al final de cada sprint, tienen una duración aproximada de 1 hora, el equipo discute sobre cuales tareas deberían ser incluidas en el siguiente sprint, adicionalmente se les asigna puntos de historia haciendo uso ocasional de scrum poker.
- **Sprint Retrospective**: Se llevan a en el mismo espacio que los sprint planning, haciendo uso de la herramienta web reetro.io cada miembro aporta aspectos positivos a resaltar sobre el rendimiento del equipo en el sprint pasado, además se aportan aspectos a mejorar junto con acciones que ayuden a resolver esas situaciones.

Figura 11

Tablero de retrospectiva usando la herramienta retro.io



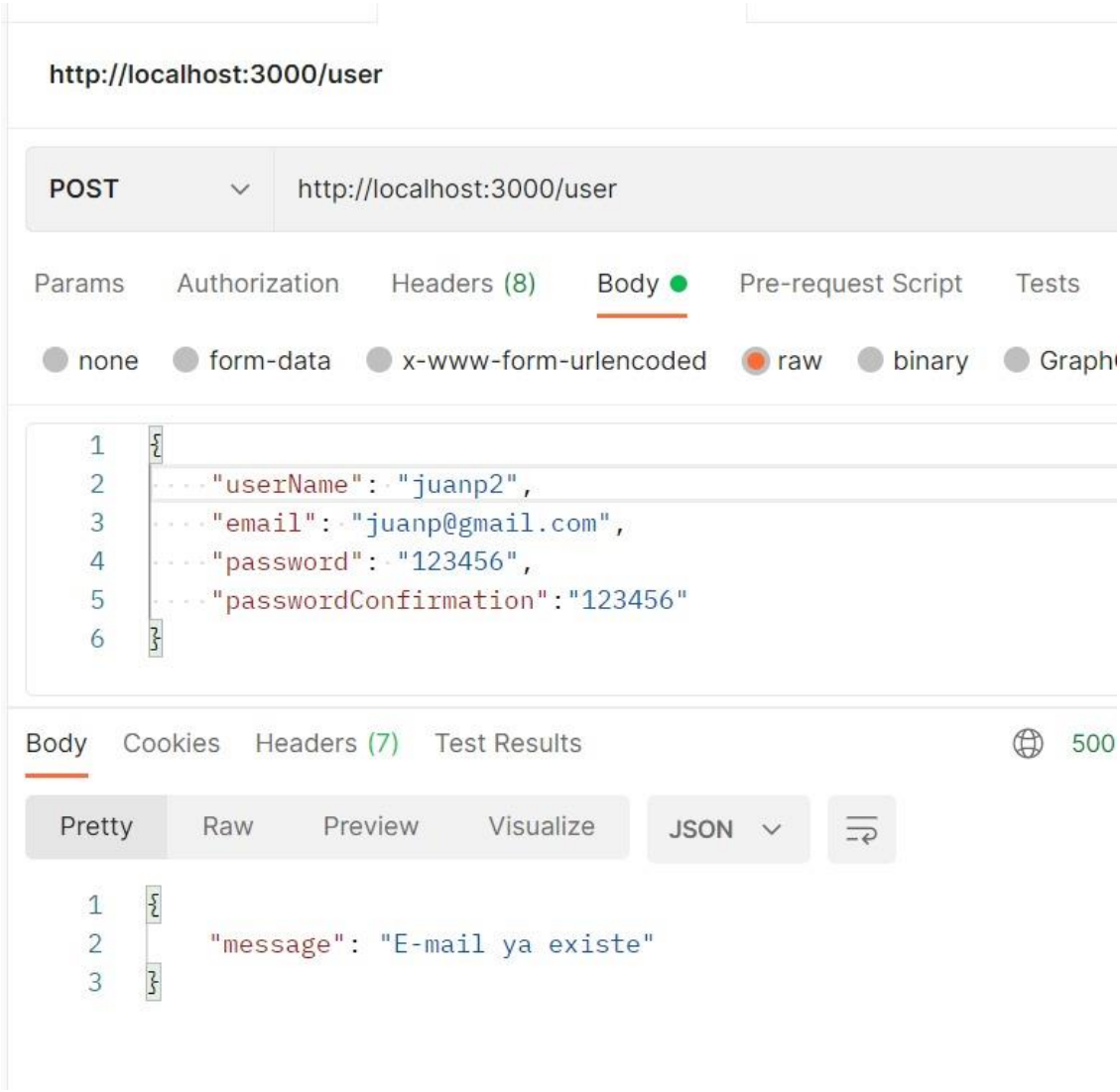
Nota. Tablero en el cual cada miembro del equipo aporta aspectos positivos sobre el trabajo del sprint anterior y aspectos a mejorar, adicionalmente los miembros se comprometen con alguna acción con el fin de mejorar el trabajo del equipo.

DESARROLLO

Cada miembro del equipo debe implementar la solución de las historias de usuario asignadas, crear una rama destinada para ello en el repositorio, crear un Pull request con el código finalizado y posteriormente incluir evidencias del correcto funcionamiento de la característica que se trabajó. A continuación, se muestran algunas de ellas

Figura 12

Evidencia presentada de la historia de usuario "Endpoint Registro de usuario" referenciada en la figura 6



http://localhost:3000/user

POST http://localhost:3000/user

Params Authorization Headers (8) Body Pre-request Script Te

none form-data x-www-form-urlencoded raw binary

```
{
    "userName": "juanp",
    "email": "juanp2@gmail.com",
    "password": "123456",
    "passwordConfirmation":"123456"
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Nombre de usuario ya existe"
}
```

http://localhost:3000/user

POST http://localhost:3000/user

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary GraphQ

```
{
    "userName": "juanp",
    "email": "juanp@gmail.com",
    "password": "123456",
    "passwordConfirmation":"123456"
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Usuario Registrado correctamente"
}
```

http://localhost:3000/user

POST http://localhost:3000/user

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSO

```
{
    "userName": "juanp2",
    "email": "juanp2@gmail.com",
    "password": "1234567",
    "passwordConfirmation":"123456"
}
```

Body Cookies Headers (7) Test Results 400

Pretty Raw Preview Visualize JSON

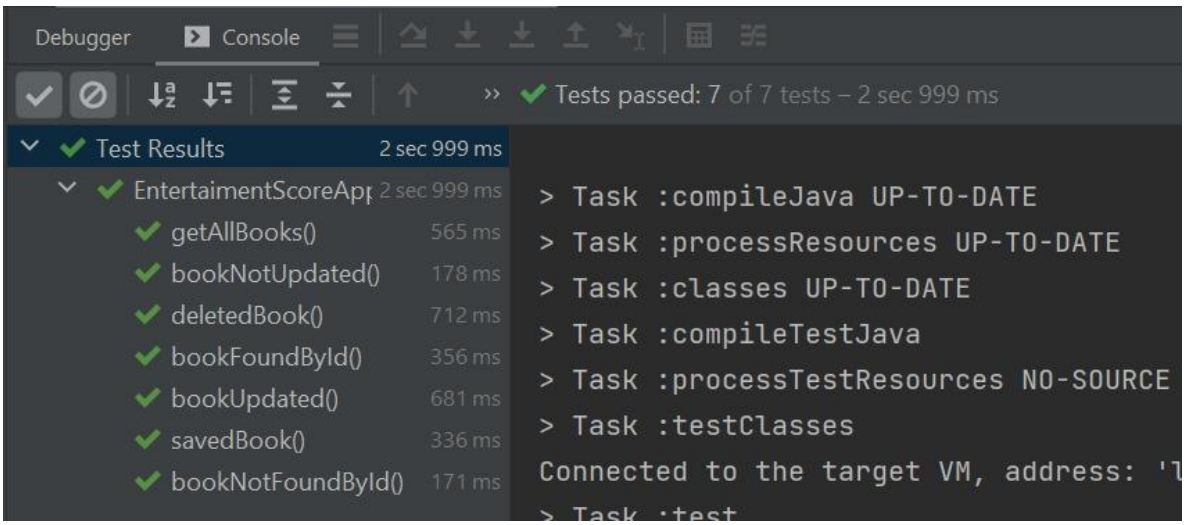
```
{
    "logId": "Las contraseñas no coinciden"
}
```

Nota. Las evidencias muestran las posibles respuestas correspondientes de la API en función de la información enviada.

Al momento de implementar los endpoints que componen la API de la aplicación, cada miembro del equipo debe añadir evidencias respecto al proceso de unit testing de cada uno de los métodos que componen la API.

Figura 13

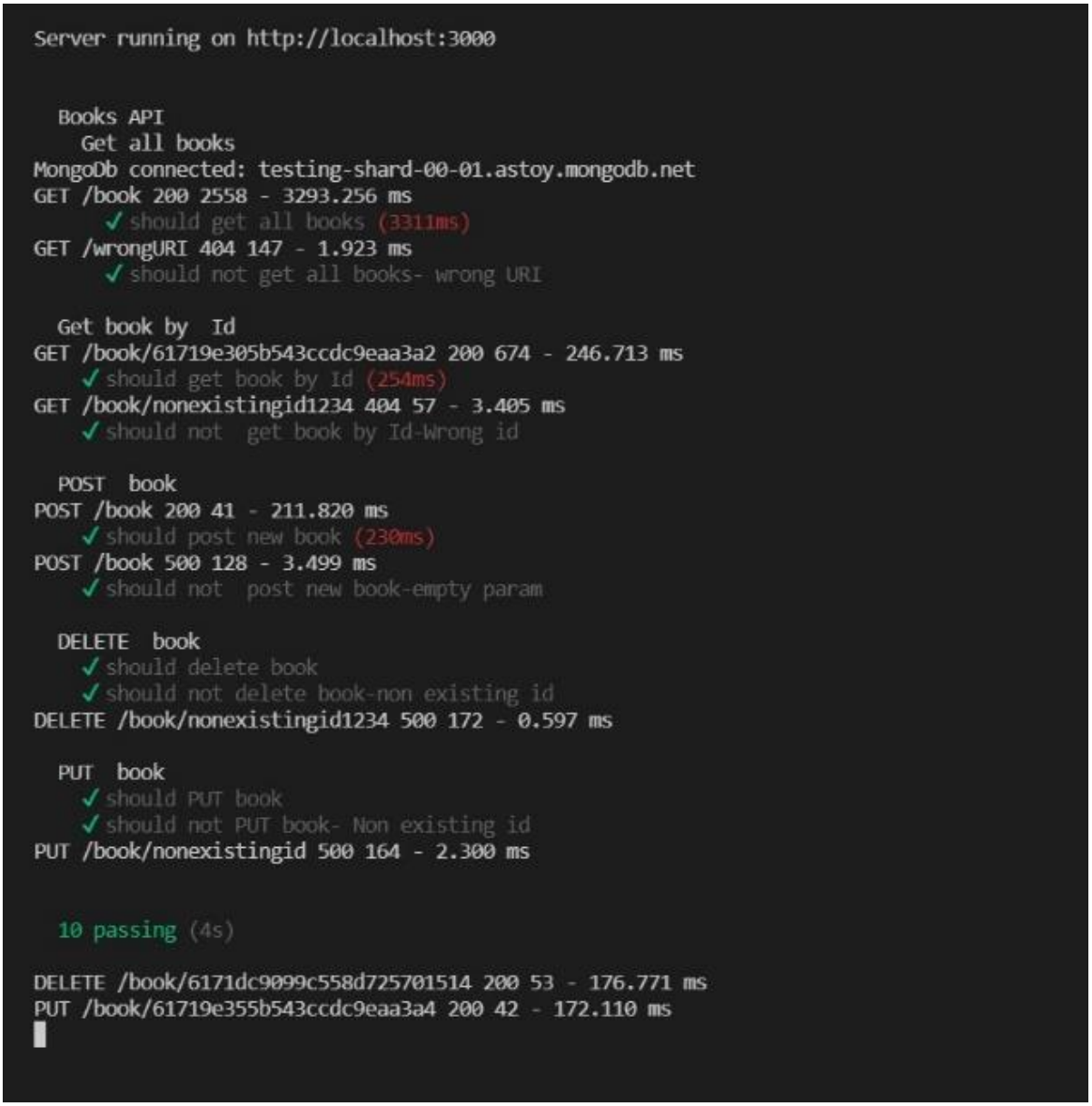
Evidencia generada referente a el unit testing realizado en la API codificada usando java-Springboot y la librería Junit .



Nota. En la imagen se observan los métodos usados en la clase de testing junto a su estado de aprobado

Figura 14

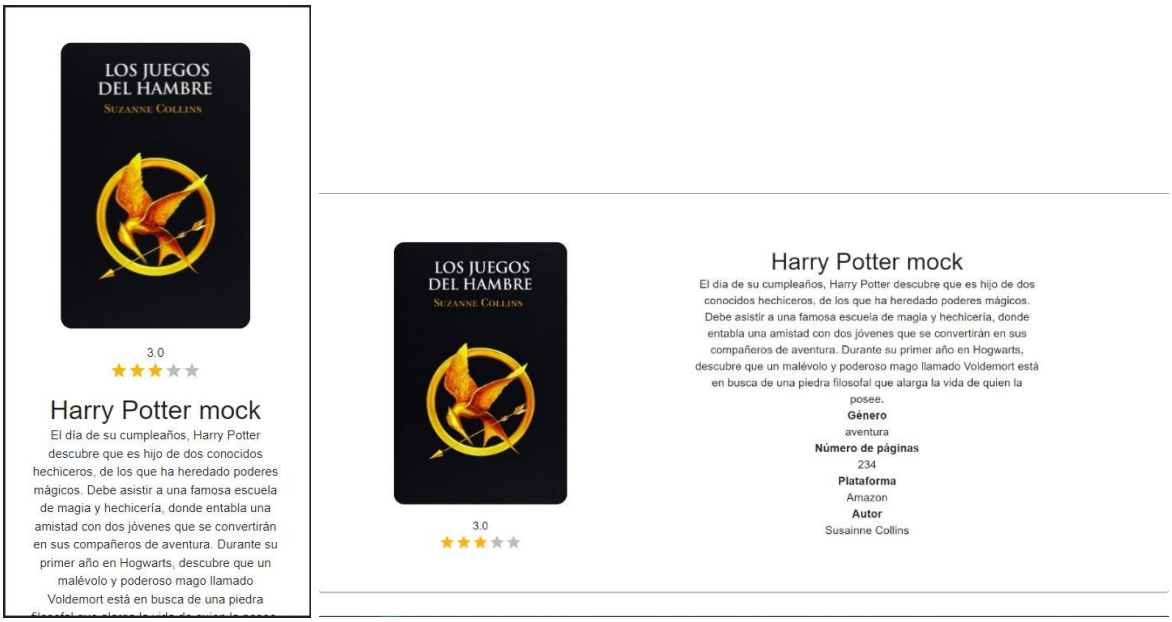
Evidencia generada referente a el unit testing realizado en la API codificada usando node-express y la librería chai.js.



Nota. En la imagen se observan los métodos usados en la clase de testing junto a su estado de aprobado

Figura 15

Evidencia vista de escritorio y vista móvil tarea consultar libro.



Fuente: Elaboración propia.

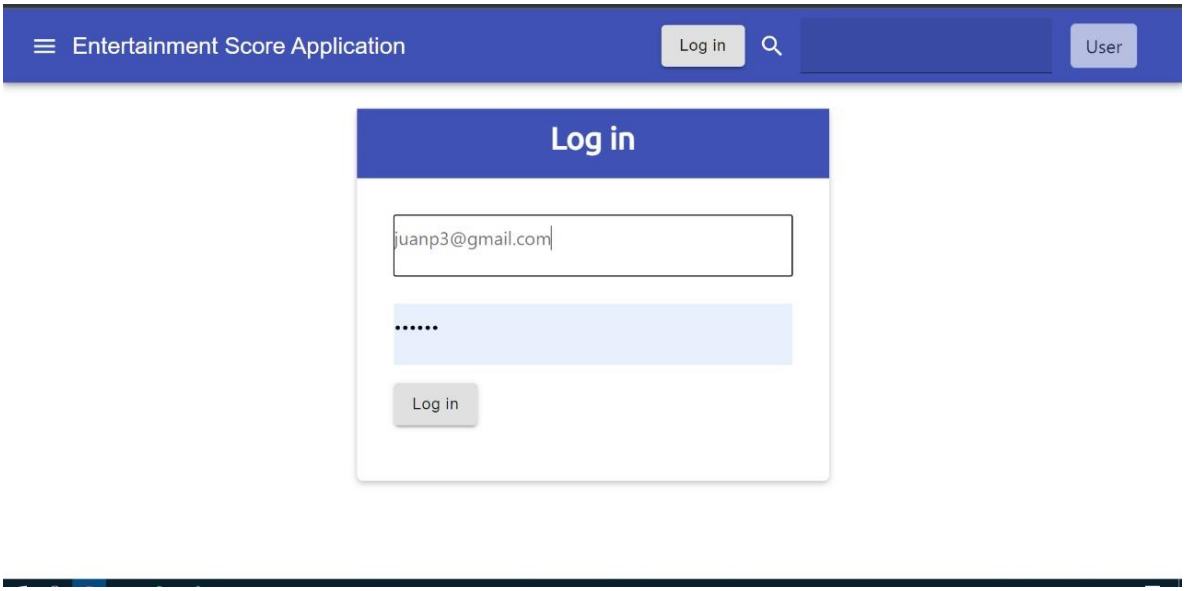
Figura 16

Evidencia vista escritorio y vista móvil editar libro.

Fuente: Elaboración propia.

Figura 17

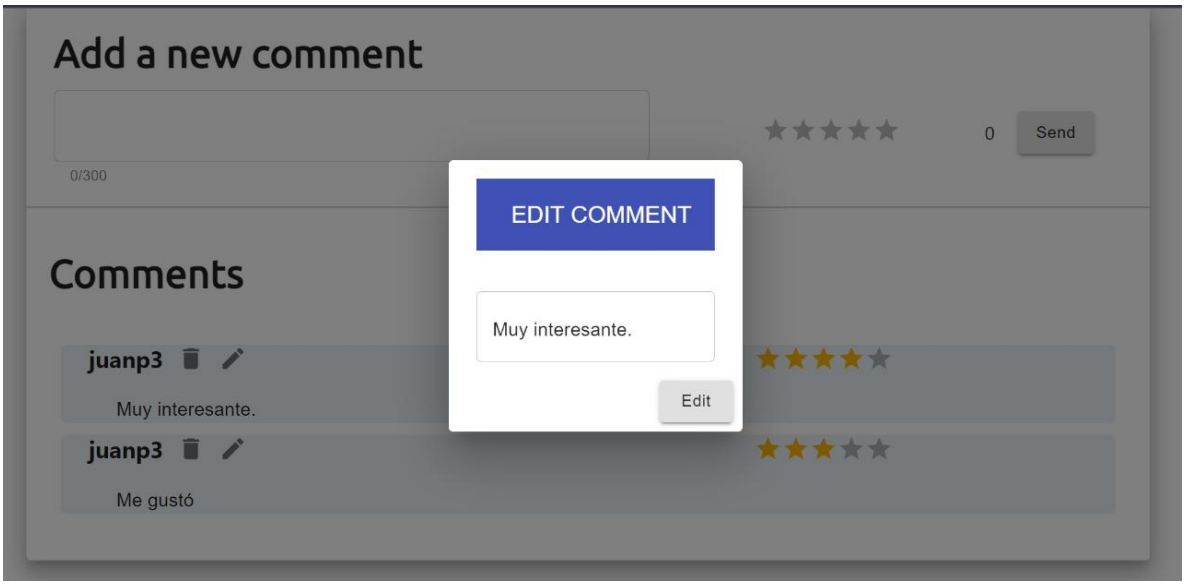
Evidencia vista Login.



Fuente: Elaboración Propia

Figura 18

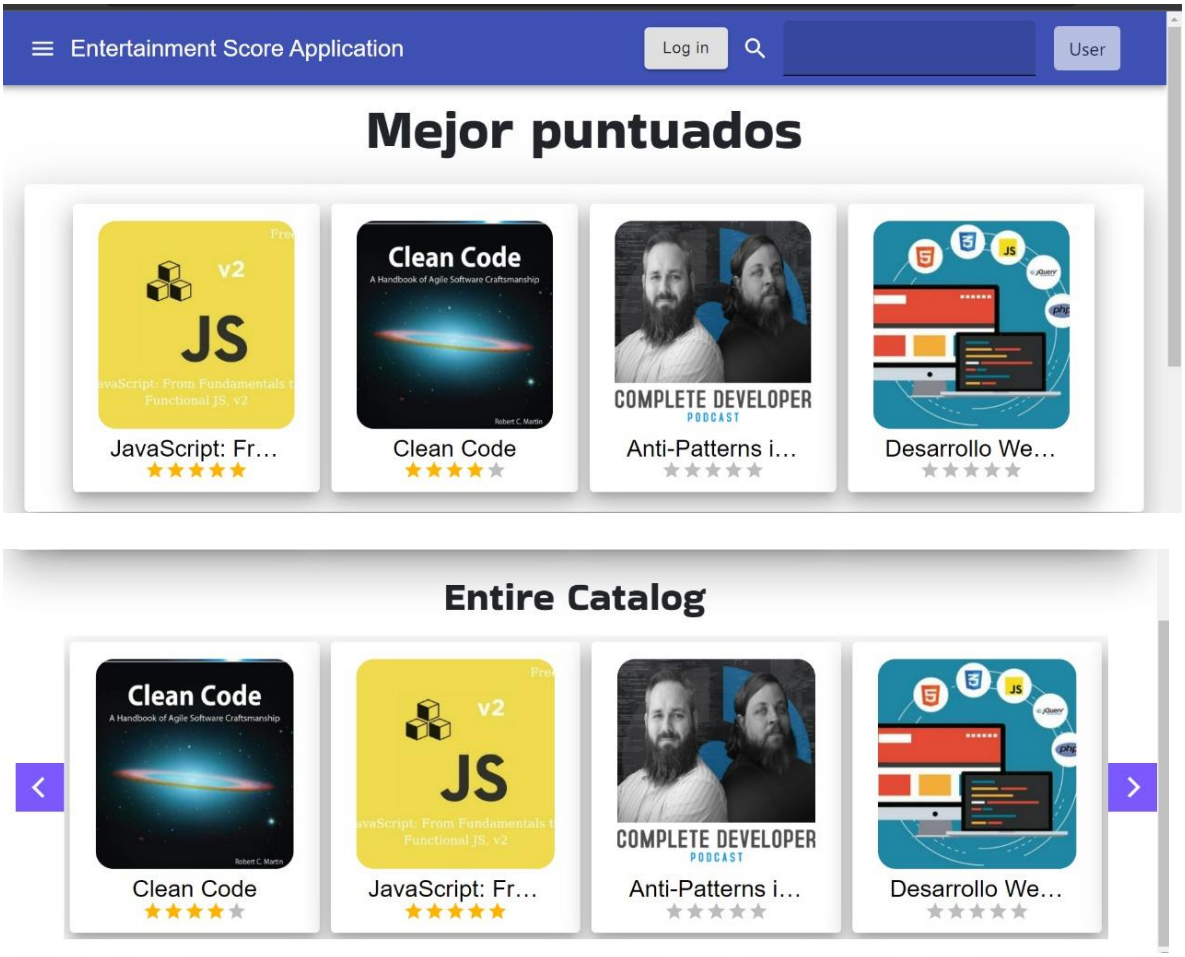
Evidencia vista editar comentario



Fuente: Elaboración propia

Figura 19

Evidencia vista consultar todos los libros



Nota: Vista principal del aplicativo, en la cual se muestra una vista previa de los 5 libros con mejor puntuación promedio. Posteriormente se muestra todo el catálogo

Fuente: Elaboración propia

ANÁLISIS DE RESULTADOS

Los resultados presentados están compuestos por 3 APIs correspondientes a libros, usuarios y comentarios implementadas en nodejs,express junto con la librería mongoose para el manejo de la base de datos mongo y java-Springboot, usando la base de datos no relacional mongoDb.

Figura 17

Atributos y estructura Json de los 3 tipos de objetos manejados.

Fuente: Elaboración propia

Como se referencia en la sección de resultados, cada uno de los métodos que componen las APIs tuvieron un proceso de unit testing, el cual se encarga de verificar que dichos métodos devuelvan el mensaje de éxito o de error correctos y además retorne los objetos requeridos en sus formatos correspondientes.

Como se mencionó anteriormente la capa del front-end se implementó usando la librería de javascript React.js, la cual proporciona una serie de funciones útiles a la hora de crear interfaces de usuario interactivas, como el manejo de actualizaciones de componentes cuando los datos cambian, adicionalmente se utilizó la librería materialUi la cual permite importar una serie de componentes visuales para crear interfaces más fácilmente.

Se utilizó Git como herramienta para el manejo de versiones del software, en la cual creó una rama con el nombre bookDev en la cual se manejarían las versiones principales de la aplicación, las cuales contenían características terminadas, por otro lado se crearon ramas separadas para cada una de las características desarrolladas, para ser mezcladas con la rama principal se debía crear un pull request el cual debía ser aprobado por el jefe del proyecto, una vez aprobado, la rama característica se mezcla con la rama principal y se elimina.

7. CONCLUSIONES:

A lo largo del desarrollo se trabajó en diversos aspectos que componen una aplicación web en los cuales se adquirieron una serie de habilidades que aportan mucho valor a el conjunto de habilidades de un ingeniero de sistemas.

Al tener manejo de herramientas de trabajo colaborativo(Git) es posible hacer parte de un equipo de desarrollo de una manera organizada y eficaz, indispensable para cualquier tipo de desarrollo que cuente con un grupo de desarrollo.

Al adquirir conocimiento de diseño e implementación de base de datos es posible aportar gran valor a prácticamente cualquier proyecto de desarrollo de software en el que se trabaje, ya que la gran mayoría de software debe realizar algún tipo de manejo de datos, adicionalmente tener la capacidad de permitir que los usuarios manipulen dichos datos a través de un servicio front-end que sea responsivo y que se adapte a el tamaño de pantalla del dispositivo donde se ejecute la aplicación es de suma importancia y un componente indispensable en cualquier desarrollo de software.

Finalmente, aprender conceptos relacionados con el unit testing permite que un ingeniero sea capaz de comprobar que el código que implemente funcione correctamente, facilitando aspectos del desarrollo.

El desarrollo de el aplicativo resultó ser muy fructífero para el conjunto de habilidades de los ingenieros participantes, aportando una serie de conocimientos supremamente útiles y esenciales a la hora de hacer parte de un desarrollo de software.

8. TRABAJO FUTURO:

El desarrollo trabajado contiene las características necesarias para el correcto funcionamiento del software. sin embargo, existen adiciones que aportarían gran valor al producto.

- Permitir que usuarios puedan iniciar sesión a través de una integración con cuentas de correo de Microsoft y Gmail.
- Mostrar reportes de puntuación de libros de otras fuentes.
- Soporte para otros modelos de datos, permitiendo incluir otros entretenimientos como películas, series o música (Videos, múltiples imágenes, audio)
- Soporte para imágenes de perfil de usuario.
- Una sección que permita ver al usuario un historial de comentarios publicados por él.
- Permitir que los usuarios tengan la posibilidad de ver estadísticas de uso de otros usuarios (comentarios publicados, favoritos)

Las adiciones mencionadas anteriormente explotarían el potencial de la idea inicial, permitiendo que se expanda tanto el alcance actual del software como la experiencia de usuario.

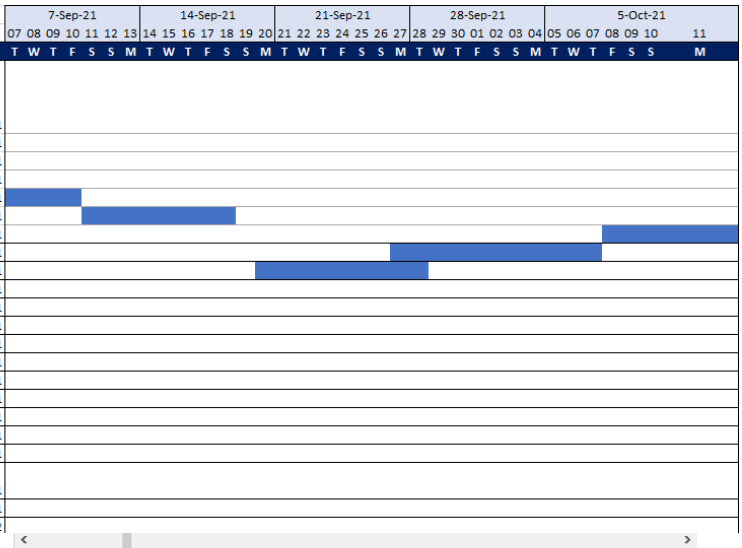
9. CRONOGRAMA

RoadMap.xlsx

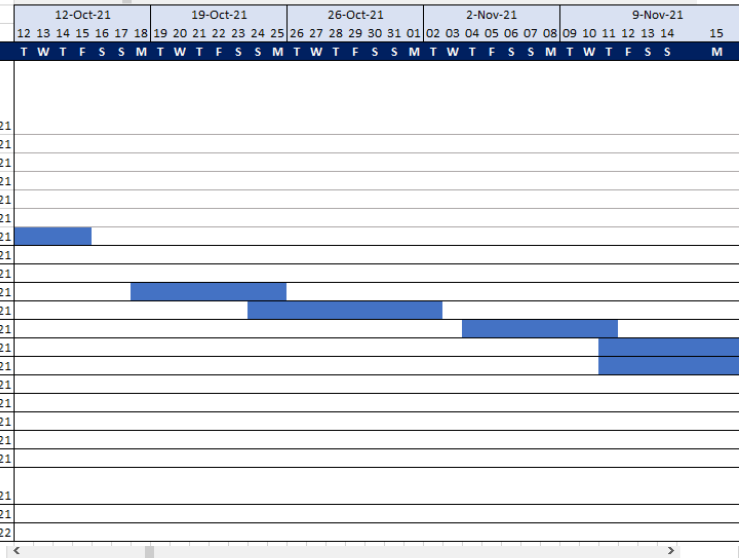
				6-Jul-21							13-Jul-21							20-Jul-21							27-Jul-21							3-Aug-21							
FECHA DE INICIO	7/6/2021																																					
SEMANA	0			06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	01	02	03	04	05	06	07	08	09
Descripción	**Inicio**	**Duración**	**Fin**	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021																																			
Endpoint crear libro	8/5/2021	9	8/14/2021																																			
Ednpoint eliminar libro	8/16/2021	9	8/25/2021																																			
Endpoint consultar libro por id	8/24/2021	9	9/2/2021																																			
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021																																			
Endpoint editar libro	9/11/2021	8	9/19/2021																																			
Deuda técnica	10/8/2021	8	10/16/2021																																			
Capacitación React	9/27/2021	11	10/8/2021																																			
capacitación node-express	9/20/2021	9	9/29/2021																																			
CRUD funcional express	10/18/2021	8	10/26/2021																																			
Unit Testing backend	10/24/2021	10	11/3/2021																																			
interfaz crear libro	11/4/2021	8	11/12/2021																																			
Interfaz editar libro	11/11/2021	8	11/19/2021																																			
Interfaz consultar libro por id	11/11/2021	8	11/19/2021																																			
Enpoint registro usuario	11/18/2021	8	11/26/2021																																			
Interfaz eliminar libro	11/19/2021	8	11/27/2021																																			
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021																																			
integración frontend y backend	12/1/2021	8	12/9/2021																																			
Endpoint editar comentarios	12/9/2021	7	12/16/2021																																			
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021																																			
Interfaz editar comentario	12/27/2021	3	12/30/2021																																			
Refactor de código	1/3/2022	3	1/6/2022																																			

				3-Aug-21							10-Aug-21							17-Aug-21							24-Aug-21							31-Aug-21							
FECHA DE INICIO	7/6/2021																																					
SEMANA	4			03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	01	02	03	04	05	06
Descripción	**Inicio**	**Duración**	**Fin**	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021																																			
Endpoint crear libro	8/5/2021	9	8/14/2021																																			
Ednpoint eliminar libro	8/16/2021	9	8/25/2021																																			
Endpoint consultar libro por id	8/24/2021	9	9/2/2021																																			
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021																																			
Endpoint editar libro	9/11/2021	8	9/19/2021																																			
Deuda técnica	10/8/2021	8	10/16/2021																																			
Capacitación React	9/27/2021	11	10/8/2021																																			
capacitación node-express	9/20/2021	9	9/29/2021																																			
CRUD funcional express	10/18/2021	8	10/26/2021																																			
Unit Testing backend	10/24/2021	10	11/3/2021																																			
interfaz crear libro	11/4/2021	8	11/12/2021																																			
Interfaz editar libro	11/11/2021	8	11/19/2021																																			
Interfaz consultar libro por id	11/11/2021	8	11/19/2021																																			
Enpoint registro usuario	11/18/2021	8	11/26/2021																																			
Interfaz eliminar libro	11/19/2021	8	11/27/2021																																			
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021																																			
integración frontend y backend	12/1/2021	8	12/9/2021																																			
Endpoint editar comentarios	12/9/2021	7	12/16/2021																																			
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021																																			
Interfaz editar comentario	12/27/2021	3	12/30/2021																																			
Refactor de código	1/3/2022	3	1/6/2022																																			

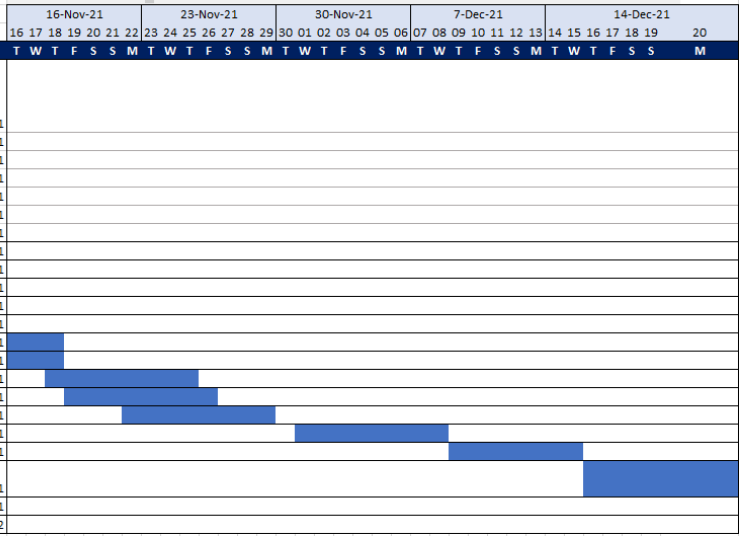
FECHA DE INICIO	7/6/2021		
SEMANA	9		
Descripción	**Inicio**	**Duración**	**Fin**
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021
Endpoint crear libro	8/5/2021	9	8/14/2021
Ednpoint eliminar libro	8/16/2021	9	8/25/2021
Endpoint consultar libro por id	8/24/2021	9	9/2/2021
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021
Endpoint editar libro	9/11/2021	8	9/19/2021
Deuda técnica	10/8/2021	8	10/16/2021
Capacitación React	9/27/2021	11	10/8/2021
capacitación node-express	9/20/2021	9	9/29/2021
CRUD funcional express	10/18/2021	8	10/26/2021
Unit Testing backend	10/24/2021	10	11/3/2021
interfaz crear libro	11/4/2021	8	11/12/2021
Interfaz editar libro	11/11/2021	8	11/19/2021
Interfaz consultar libro por id	11/11/2021	8	11/19/2021
Enpoint registro usuario	11/18/2021	8	11/26/2021
Interfaz eliminar libro	11/19/2021	8	11/27/2021
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021
integración frontend y backend	12/1/2021	8	12/9/2021
Endpoint editar comentarios	12/9/2021	7	12/16/2021
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021
Interfaz editar comentario	12/27/2021	3	12/30/2021
Refactor de código	1/3/2022	3	1/6/2022



FECHA DE INICIO	7/6/2021		
SEMANA	14		
Descripción	**Inicio**	**Duración**	**Fin**
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021
Endpoint crear libro	8/5/2021	9	8/14/2021
Ednpoint eliminar libro	8/16/2021	9	8/25/2021
Endpoint consultar libro por id	8/24/2021	9	9/2/2021
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021
Endpoint editar libro	9/11/2021	8	9/19/2021
Deuda técnica	10/8/2021	8	10/16/2021
Capacitación React	9/27/2021	11	10/8/2021
capacitación node-express	9/20/2021	9	9/29/2021
CRUD funcional express	10/18/2021	8	10/26/2021
Unit Testing backend	10/24/2021	10	11/3/2021
interfaz crear libro	11/4/2021	8	11/12/2021
Interfaz editar libro	11/11/2021	8	11/19/2021
Interfaz consultar libro por id	11/11/2021	8	11/19/2021
Enpoint registro usuario	11/18/2021	8	11/26/2021
Interfaz eliminar libro	11/19/2021	8	11/27/2021
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021
integración frontend y backend	12/1/2021	8	12/9/2021
Endpoint editar comentarios	12/9/2021	7	12/16/2021
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021
Interfaz editar comentario	12/27/2021	3	12/30/2021
Refactor de código	1/3/2022	3	1/6/2022

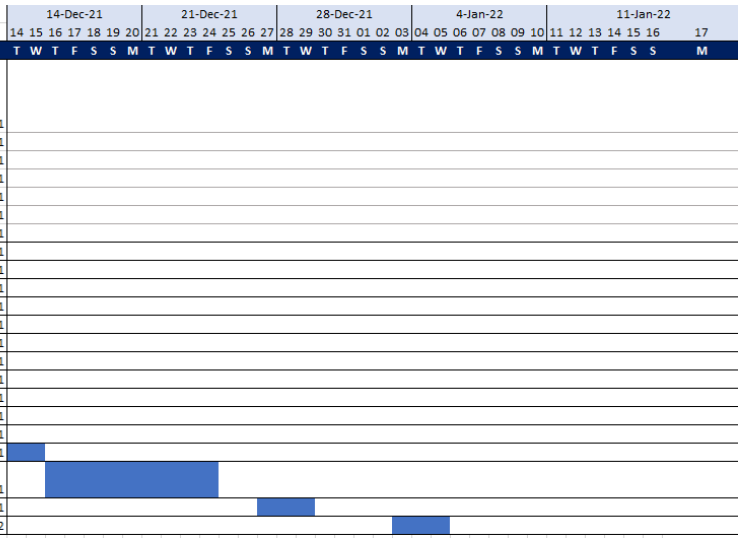


FECHA DE INICIO	7/6/2021		
SEMANA	19		
Descripción	**Inicio**	**Duración**	**Fin**
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021
Endpoint crear libro	8/5/2021	9	8/14/2021
Ednpoint eliminar libro	8/16/2021	9	8/25/2021
Endpoint consultar libro por id	8/24/2021	9	9/2/2021
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021
Endpoint editar libro	9/11/2021	8	9/19/2021
Deuda técnica	10/8/2021	8	10/16/2021
Capacitación React	9/27/2021	11	10/8/2021
capacitación node-express	9/20/2021	9	9/29/2021
CRUD funcional express	10/18/2021	8	10/26/2021
Unit Testing backend	10/24/2021	10	11/3/2021
interfaz crear libro	11/4/2021	8	11/12/2021
Interfaz editar libro	11/11/2021	8	11/19/2021
Interfaz consultar libro por id	11/11/2021	8	11/19/2021
Enpoint registro usuario	11/18/2021	8	11/26/2021
Interfaz eliminar libro	11/19/2021	8	11/27/2021
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021
integración frontend y backend	12/1/2021	8	12/9/2021
Endpoint editar comentarios	12/9/2021	7	12/16/2021
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021
Interfaz editar comentario	12/27/2021	3	12/30/2021
Refactor de código	1/3/2022	3	1/6/2022



FECHA DE INICIO	7/6/2021		
SEMANA	23		

Descripción	Inicio	Duración	Fin
Capacitación técnica java y Spring framework(patrones de diseño, principios solid, programación usando hilos)	7/6/2021	30	8/5/2021
Endpoint crear libro	8/5/2021	9	8/14/2021
Ednpoint eliminar libro	8/16/2021	9	8/25/2021
Endpoint consultar libro por id	8/24/2021	9	9/2/2021
Endpoint consultar todos los libros	9/1/2021	10	9/11/2021
Endpoint editar libro	9/11/2021	8	9/19/2021
Deuda técnica	10/8/2021	8	10/16/2021
Capacitación React	9/27/2021	11	10/8/2021
capacitación node-express	9/20/2021	9	9/29/2021
CRUD funcional express	10/18/2021	8	10/26/2021
Unit Testing backend	10/24/2021	10	11/3/2021
interfaz crear libro	11/4/2021	8	11/12/2021
Interfaz editar libro	11/11/2021	8	11/19/2021
Interfaz consultar libro por id	11/11/2021	8	11/19/2021
Enpoint registro usuario	11/18/2021	8	11/26/2021
Interfaz eliminar libro	11/19/2021	8	11/27/2021
Interfaz consultar todos los libros	11/22/2021	8	11/30/2021
Integración frontend y backend	12/1/2021	8	12/9/2021
Endpoint editar comentarios	12/9/2021	7	12/16/2021
Endpoint eliminar comentarios por libro id	12/16/2021	9	12/25/2021
Interfaz editar comentario	12/27/2021	3	12/30/2021
Refactor de código	1/3/2022	3	1/6/2022



10. BIBLIOGRAFÍA

1. *What is Scrum* (2021.). scrum.org. https://www.scrum.org/resources/what-is-scrum
2. *Spring Framework* (2021). Spring.io. https://spring.io/projects/spring-framework
3. *¿Qué es mongoDB?* (2021). mongodb. https://www.mongodb.com/es/what-is-mongodb
4. *React* (2021). reactjs. https://es.reactjs.org/
5. *Web application(Web app)*(AGO/2019) searchsoftwarequality. https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app
6. *What is a web Application?* (31/May/2016) Robert Gibb https://blog.stackpath.com/web-application/
7. *Desing Patterns and frameworks*(s.f).oracle, https://docs.oracle.com/cd/E19929-01/816-4337/03_design_issues.html

8. *About Intertec* (2021.). intertecintl. https://www.intertecintl.com/about
9. *API REST*(06/04/2021).ibm. https://www.ibm.com/co-es/cloud/learn/rest-apis
10. *REST APIs*(27/Jul/2020). Juviler Jamie https://blog.hubspot.com/website/what-is-rest-api
11. *SOLID:the first 5 principles of Object Oriented Design* (17/Sep/2020). Samuel Oloruntoba *https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design*
12. *PATRONES DE DISEÑO (2021).* refactoring.guru https://refactoring.guru/es/design-patterns/what-is-pattern
13. Shvets Alexander *(2019) Sumérgete en los patrones de diseño. https://refactoring.guru/es/design-patterns/singleton*
14. *A solid Guide to SOLID* (18/05/2021) baeldung. https://www.baeldung.com/solid-principles
15. *Nosql-databases* (06/Ago/2019) IBM. https://www.ibm.com/cloud/learn/nosql-databases#toc-what-is-a--BX7Fquvs
16. *What is a non-Relational Database?* (2021) mongodb. https://www.mongodb.com/es/nosql-explained
17. *About* (s.f) git https://git-scm.com/about
18. *The 2020 Scrum Guide* (2020) scrumguides https://scrumguides.org/scrum-guide.html
19. *Scrum events* (31/OCT/2017) saraclip https://www.saraclip.com/eventos-en-scrum/
20. *What is unit Testing* (2021) smartbear https://smartbear.com/learn/automated-testing/what-is-unit-testing/

21. *Unit Testing Techniques and Best Practices* (28/OCT/2021) Gill Singh Navdeep https://www.xenonstack.com/insights/what-is-unit-testing

22. *La herramienta de desarrollo de software líder de los equipos ágiles (2021) Atlassian https://www.atlassian.com/es/software/jira*

23. *A brief history of Node.js* (2020) Boström, O., Borins, M., Laru, O., Miller, A., Awais, A., J, K., Chowdhury, M., Horky, K., Harrington, B. and Cunliffe, M. https://nodejs.dev/learn/a-brief-history-of-nodejs