Pontifica Universidad Javeriana

Fundamentos de Ingeniería de Software

Laura Sofia Aponte Sánchez

Juan Esteban Bello Durango

Santiago Galindo Cubillos

Juan Felipe Gutierrez

Luis Gabriel Moreno Sandoval

Bogotá D.C

Abril 8 de 2025

Segunda entrega proyecto fundamentos “ROPA”

1.Metodologias agiles

A nivel de metodologías ágiles hemos implementado sub-issues con commits en cada uno organizando por persona en cada sub-issue para mejorar la participación del equipo y la continuidad y organización del proyecto.

1.1.Sprints

Tenemos un total de 7 sprints hasta esta semana en los cuales siguen estas respectivas actividades, hay que tener en cuenta que al inicio no se tenía tan clara la idea de historias de usuario pero a medida que transcurre el proyecto se arregla ese problema creando sub issues y definiendo de forma correcta las historias de usuario con los milestones:

Sprint 1

Requerimientos funcionales

Configuracion de repositorio boiler plate

Sprint 2

Creación de las tablas mysql

Requisitos no funcionales

Sprint 3

Se recolecta la información necesaria de la base de datos

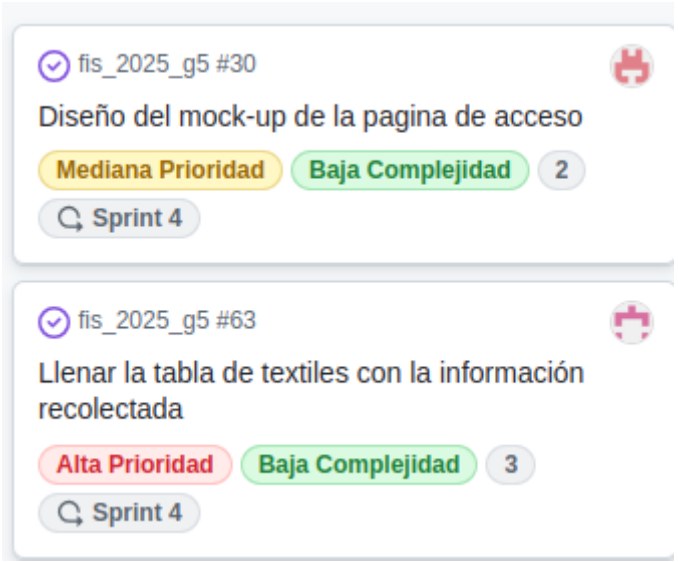
Se crea la conexión con la base de datos

Se diseña el diagrama relacional de la base de datos

Sprint 4

Se diseña el mockup de la pagina de acceso

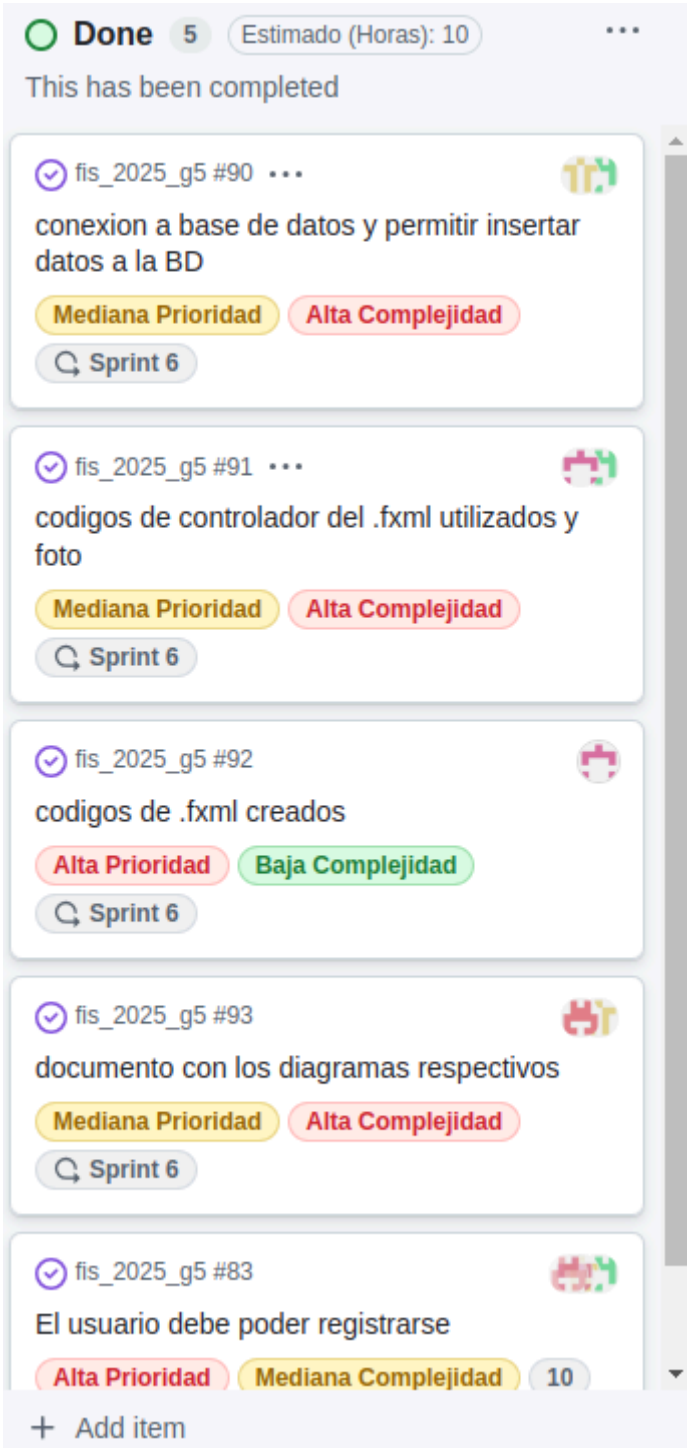
Se llena las tablas con la información recolectada



Sprint 5

Desde aqui en adelante se hace la corrección del error de issues debido a que anteriormente no estaban bien definidos desde aqui ya se tienen su estructura respectiva con descripción y de cada sub issue y la historia de usuario tambien se define de forma respectiva

Sprint 6



En el sprint 6 ya se tienen los commits de cada sub-issue y se mantiene una mayor organización de la continuidad del proyecto.

Sprint 7

Del sprint actual también se tienen sus respectivos sub-issues realizados con commits incluidos en cada uno.

Para esta entrega se ha llegado al sprint 7

2.DevOps

Para la entrega 2 se hicieron 2 pipelines para automatizar test del repositorio por medio de una serie de pasos.

El primero es para:

1. Hace Checkout del código

```
- name: Checkout del código
  uses: actions/checkout@v3
```

- Descarga el código fuente del repositorio en el runner para poder trabajar con él. Es el primer paso estándar en casi todos los pipelines.

2. Configura el JDK 21

```
- name: Configurar JDK 21
uses: actions/setup-java@v3
with:
distribution: 'temurin'
java-version: '21'
```

- Instala y configura el **Java Development Kit (JDK)**
- Esto es necesario para compilar y ejecutar código Java (el proyecto es Java con Maven).

3. Construye del proyecto

```
- name: Construcción del proyecto
run: mvn clean package
```

- Usa Maven para limpiar el proyecto (`mvn clean`) y empacarlo (`mvn package`), lo que normalmente genera un `.jar` dentro de la carpeta `target/`.

- Esta es la compilación del código y generación del artefacto ejecutable.

4. Ejecuta las pruebas

```
- name: Ejecutar pruebas
run: mvn test
```

- Ejecuta los tests del proyecto definidos en Maven.

- Esto asegura que el código compilado pasa todas las pruebas automáticas.

5. Sube los artefacto

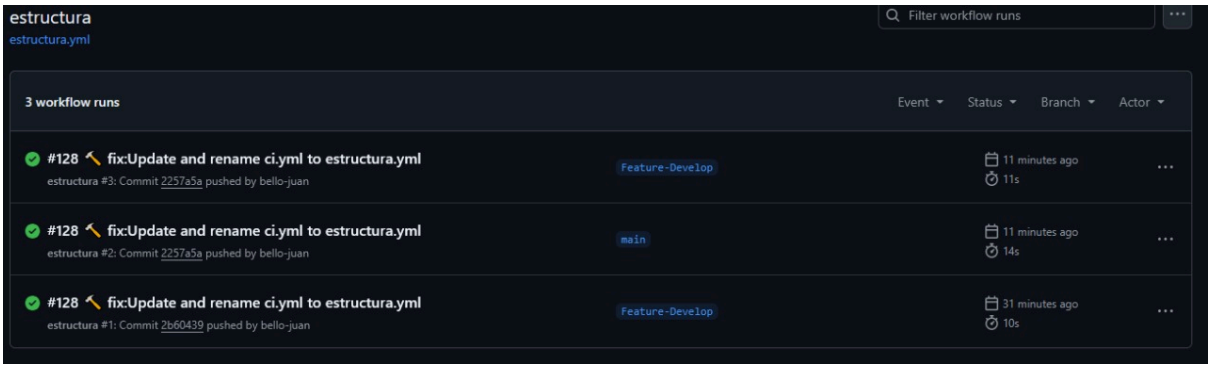
```
- name: Subir artefacto
uses: actions/upload-artifact@v4
```

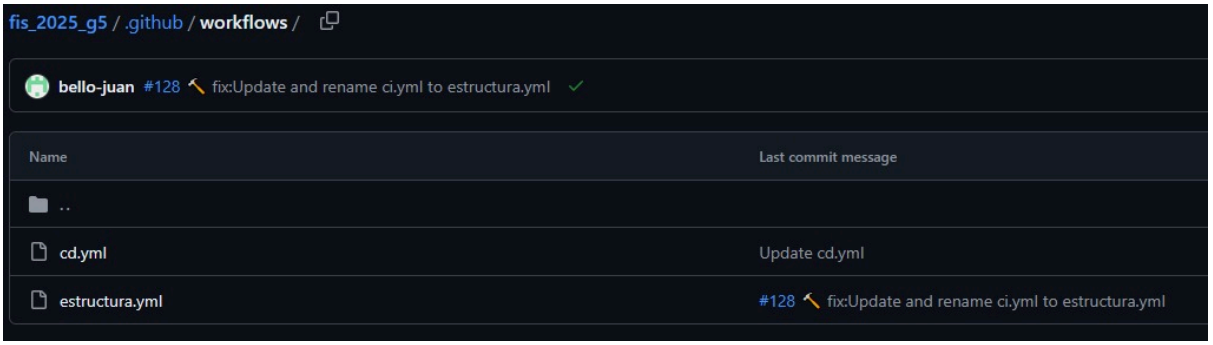
```
with:
name: app-jar
path: target/*.jar
```

- Toma el `.jar` generado en la carpeta `target/` y lo **sube como artefacto** del pipeline.

- Esto permite que esté disponible para otros jobs o para ser descargado manualmente desde la interfaz de GitHub Actions.

Las dos imágenes muestran el funcionamiento del pipeline cd.yml y la organización de los workflows del proyecto. En el historial de ejecuciones se evidencia que el pipeline se activa correctamente con cada push o pull request en ramas clave como main, Feature-Develop y workflow-patch, ejecutando exitosamente procesos de compilación, pruebas y subida de artefactos. También se observa una ejecución fallida relacionada con una reestructuración de directorios, lo que demuestra que el pipeline ayuda a detectar errores de forma temprana. Finalmente, en la estructura del repositorio se ve que existen dos workflows (cd.yml y estructura.yml), y que se realizó un ajuste reciente para mejorar la claridad al renombrar el archivo ci.yml a estructura.yml.





El segundo es para:

Crear la Estructura General

```
name: estructura
```

- Este es el nombre del workflow: `estructura`.

Se crean eventos que lo activen

```
on:
push:
branches: [
"main","Feature-Develop","workflow-patch"]
pull_request:
branches: [
"main","Feature-Develop","workflow-patch"]
```

- Se activa cuando hay:
    - Un **push** a las ramas principales (`main`, `Feature-Develop`, `workflow-patch`).
    - Un **pull request** hacia esas mismas ramas.

Pernitiendo ejecutar este pipeline en los flujos de trabajo regulares, por ejemplo, al hacer cambios en la estructura del proyecto o en scripts.

Job principal: `process-and-report`

```
jobs:
process-and-report:
runs-on: ubuntu-latest
```

- Define un único job que se llama `process-and-report`.

Pasos del job

1. Revision del código

```
- name: Checkout del código
  uses: actions/checkout@v3
```

- Clona el repositorio para que el pipeline tenga acceso al código fuente.

2. Configura el JDK

```
- name: Configurar JDK
uses: actions/setup-java@v3
with:
java-version: '21'
distribution: 'temurin'
```

- Instala y configura **Java 21** (igual que en el otro pipeline).

3. Revisa la estructura del proyecto

```
- name: Ver estructura del proyecto
run: ls -R
```

- Lista todo el contenido del directorio del proyecto (`ls -R`).
- Sirve para visualizar o validar la organización del código, carpetas, y archivos.

4. Ejecuta el procesamiento de datos

```
- name: Ejecutar procesamiento de datos
  run: |
    echo "Procesando normativa y formularios..."
    echo "Generando informe..."
    mkdir -p reportes
    echo "Informe generado el $(date)" > reportes/informe.txt
    ls -l reportes
```

- Simula un procesamiento de información normativa o formularios.
- Crea una carpeta de reportes.
- Genera un archivo llamado `informe.txt` que contiene la fecha y hora actual.
- Muestra los archivos dentro de `reportes`.

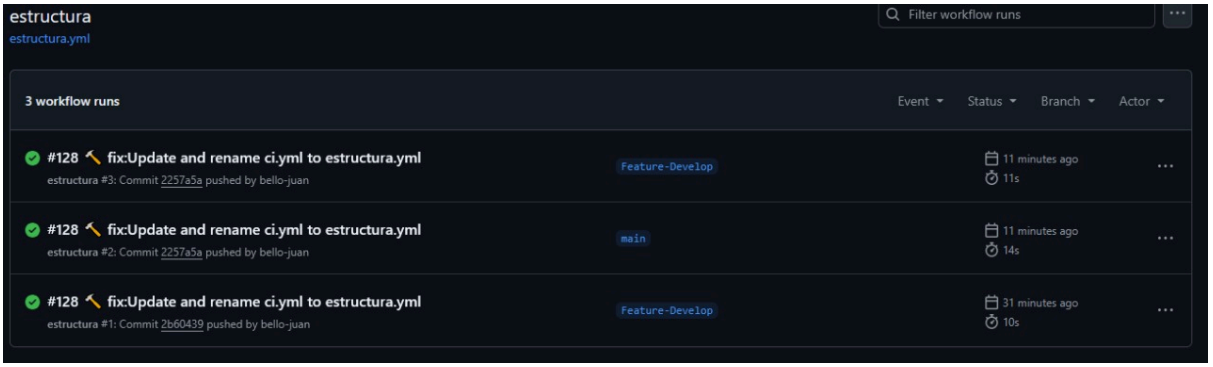
Este paso es completamente personalizable: podría contener procesamiento real si agregas comandos más complejos (por ejemplo, scripts en Java, Python o bash).

5. Sube los artefactos del informe

```
- name: Subir artefactos del informe
  uses: actions/upload-artifact@v4
  with:
    name: informe-final
    path: reportes/informe.txt
```

- Toma el informe generado (`informe.txt`) y lo sube como un artefacto de GitHub Actions, que luego puede descargarse manualmente desde la interfaz de GitHub.

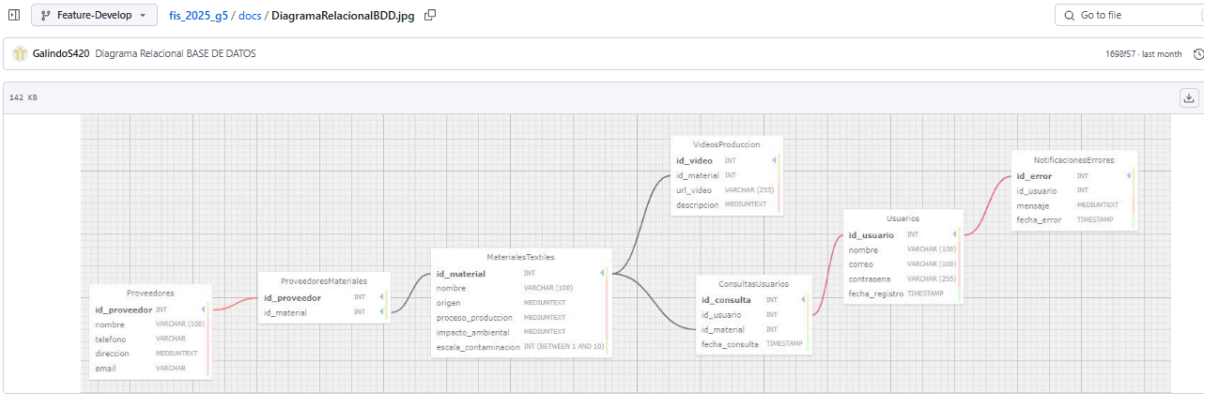
En la siguiente imagen muestra la ejecución del pipeline `estructura.yml`, el cual fue recientemente renombrado desde `ci.yml`. Se evidencia que ha sido ejecutado tres veces de forma exitosa en ramas como `main` y `Feature-Develop`, en todos los casos con el mismo commit relacionado con el cambio de nombre y actualización del archivo. Esto confirma que el workflow está correctamente configurado para activarse con `push` y `pull request` en las ramas principales, y que su funcionamiento básico —como verificar la estructura del proyecto, generar un informe e incluirlo como artefacto— se realiza sin errores.



3.Diseño y arquitectura

El proyecto posee todos los diagramas solicitados: diagrama de Base de Datos, Clases, Componentes y Despliegue. Son acordes a la lógica de negocio y alos requerimientos solicitados.

Diagrama de base de datos



Historia de Usuario: El usuario, proveedor y emprendedor deben poder acceder a sus cuentas

Diagrama de clases:

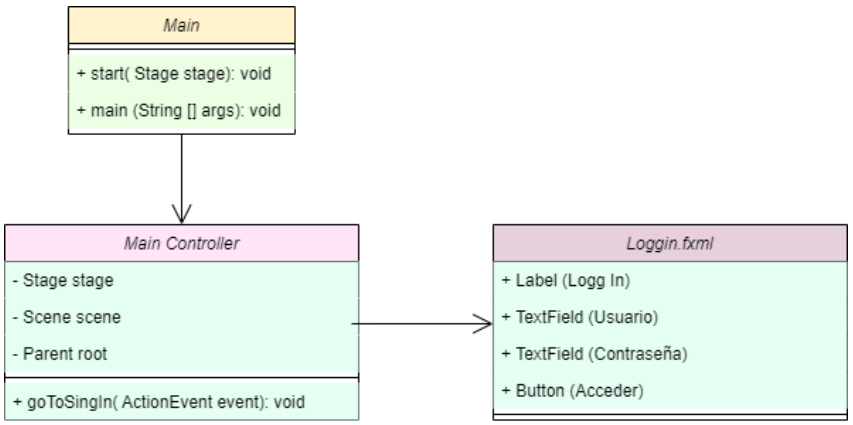


Diagrama de componentes:

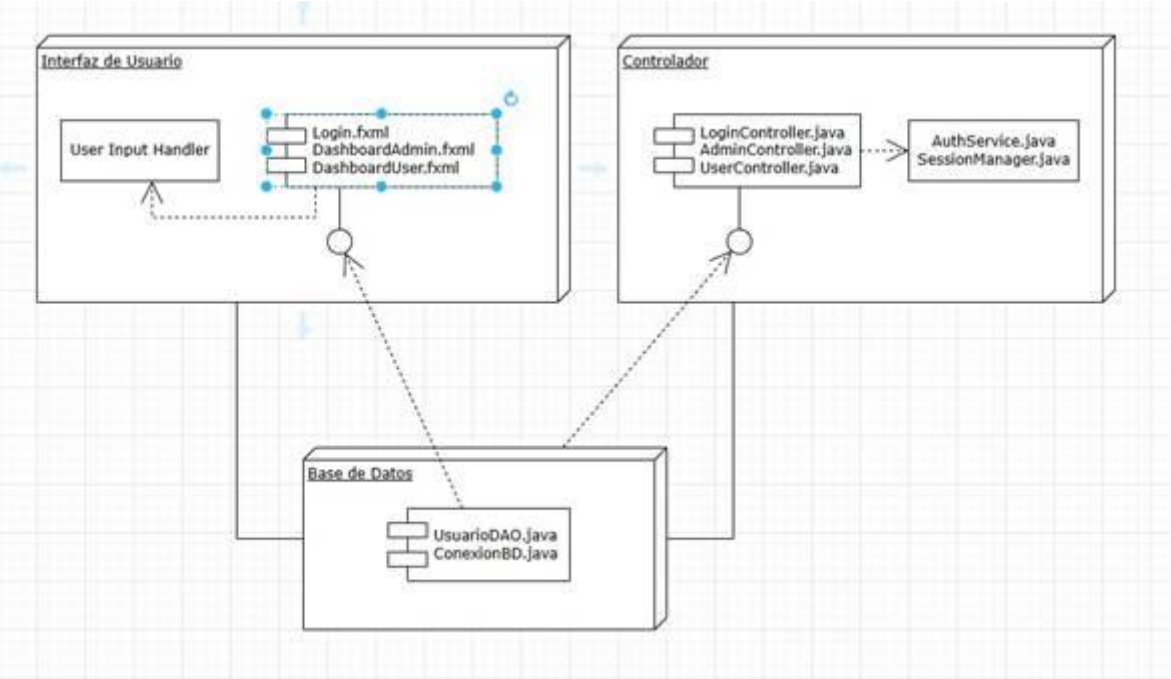


Diagrama de paquetes:

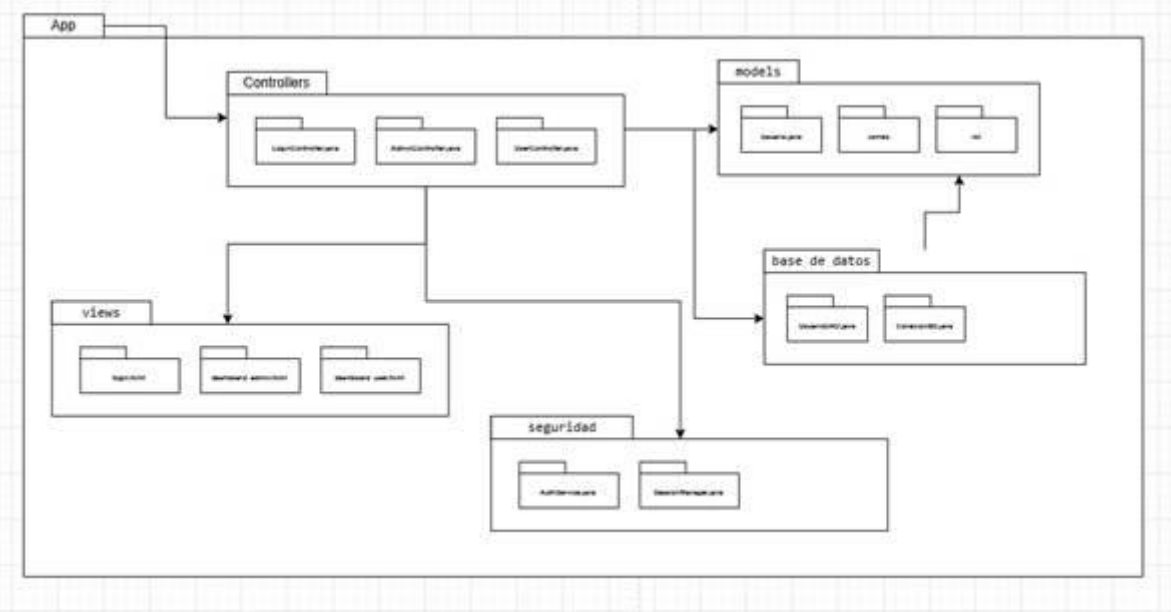
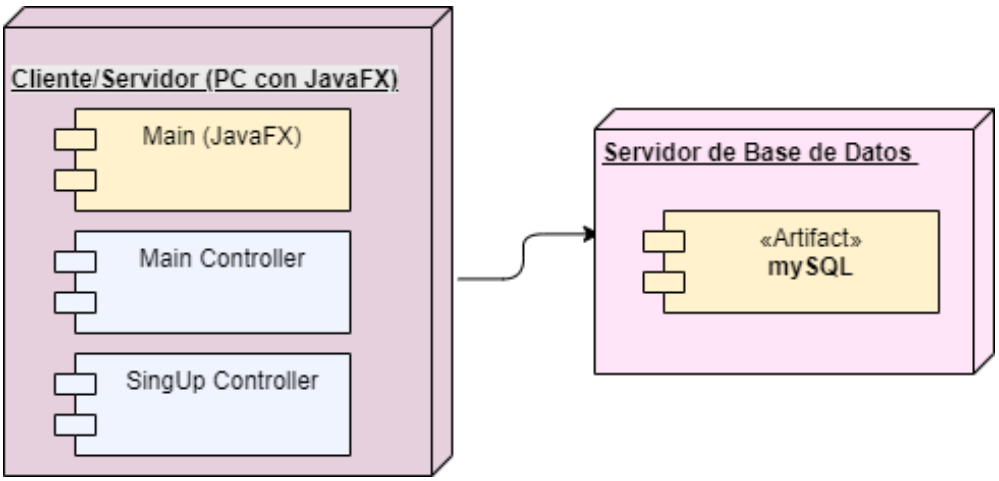


Diagrama de despliegue:



Historia de Usuario: El usuario debe poder registrarse.

Diagrama de clases:

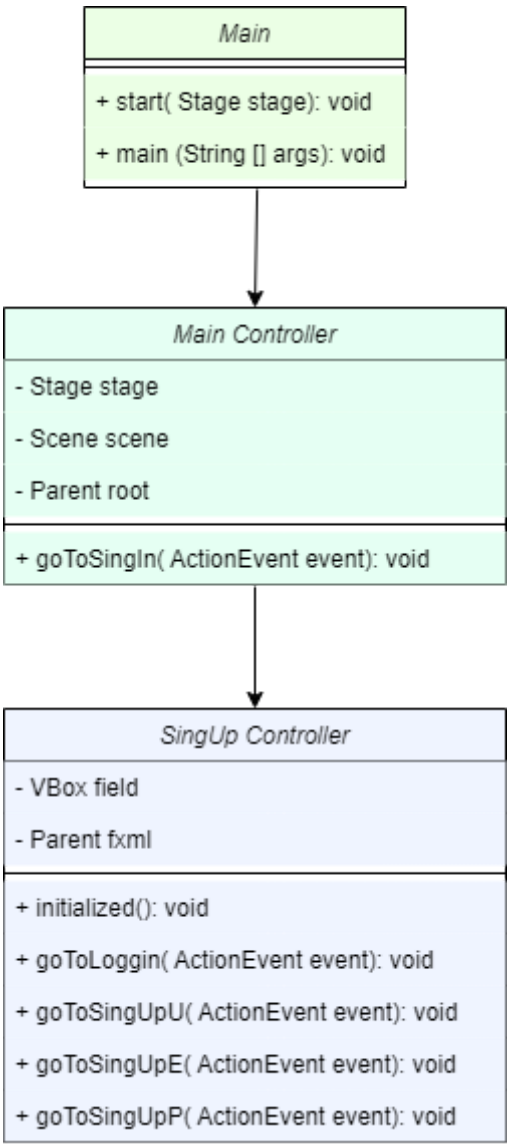
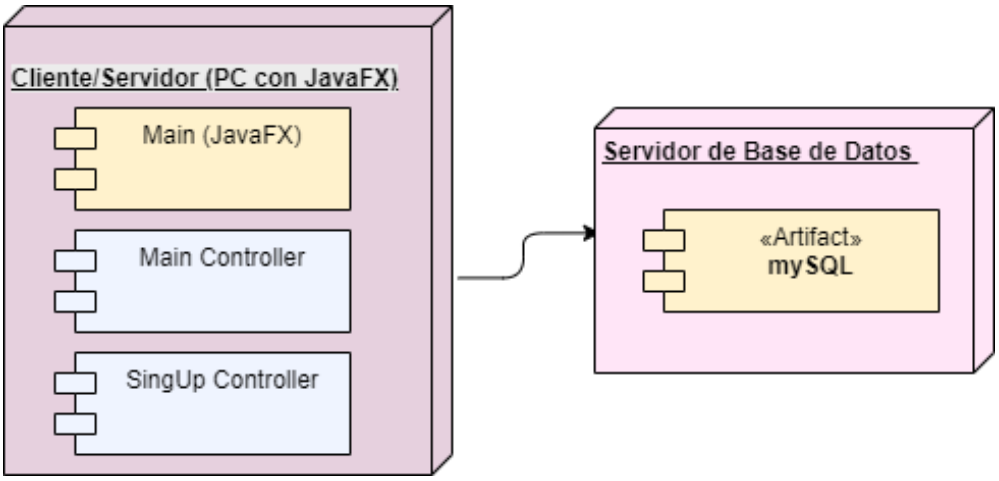
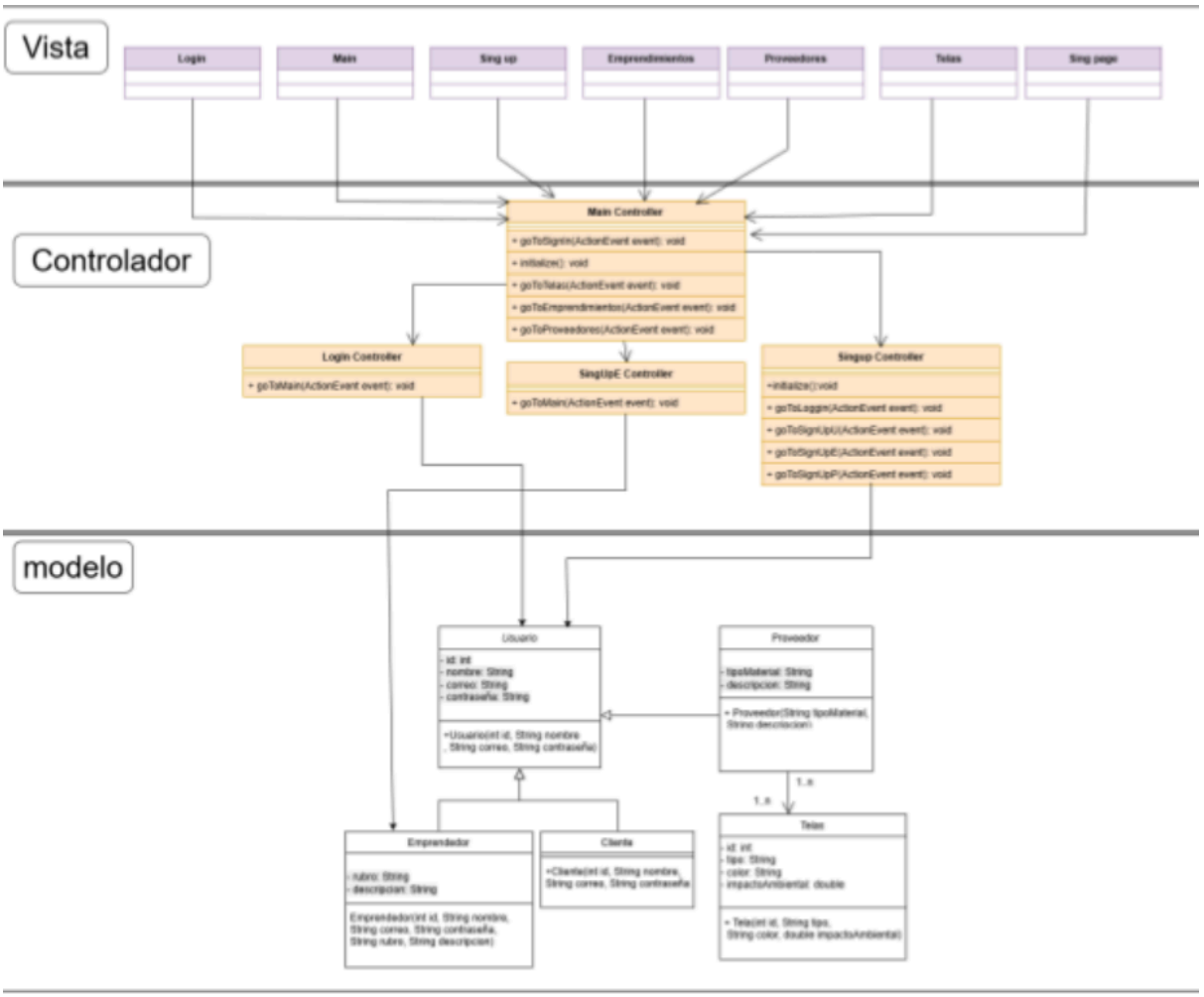


Diagrama de despliegue:



Modelo MVC

Para el desarrollo del proyecto se sigue el patrón de arquitectura MVC (Modelo-Vista-Controlador), el cual permite una clara separación de responsabilidades, mejorando la organización, mantenimiento y escalabilidad del código.



Modelo

Contiene la lógica de negocio y los datos del sistema. En este proyecto, las clases modelo son:

· Usuario: Clase base con atributos comunes (id, nombre, correo, contraseña).

o Cliente y Emprendedor heredan de Usuario. El emprendedor tiene campos adicionales como rubro y descripción.

· Proveedor: Representa a los proveedores de telas, con atributos como tipo de material y descripción.

· Tela: Clase asociada a los proveedores. Incluye información sobre el tipo de tela, color e impacto ambiental.

Estas clases encapsulan los datos que el sistema maneja y pueden incluir métodos adicionales como validaciones o cálculos futuros.

Controlador

Es el puente entre la vista y el modelo. Se encarga de recibir eventos de la interfaz de usuario, procesarlos y comunicarse con el modelo o cambiar de vista.

· MainController: Coordina la navegación principal entre pantallas como inicio de sesión, sign up, telas, emprendimientos y proveedores.

· LoginController: Controla la pantalla de login y redirige al menú principal tras una autenticación exitosa.

· SignupController: Administra las acciones en la pantalla de registro y navegación hacia otras secciones.

· SignupEController: Controla el registro específico de emprendedores.

Cada controlador responde a eventos de la interfaz, como clics de botones, y permite transiciones entre vistas.

Vista

Corresponde a las pantallas del sistema, implementadas en JavaFX. Incluye interfaces como:

· Login, Main, Sign up, Emprendimientos, Proveedores, Telas, Sign page.

Cada vista está vinculada a su controlador correspondiente y permite la interacción del usuario con el sistema

4.Patrones usados

El patrón GRASP se ve asignando responsabilidades de forma lógica dentro del diseño. En el proyecto, el patrón controlador se aplica al definir clases encargadas de recibir las solicitudes del usuario y coordinar las respuestas, como los controladores de usuarios, productos o proveedores. En creator está presente cuando una clase, como un servicio, crea instancias de otras clases relacionadas, lo cual mejora la cohesión. Además, se puede evidenciar un diseño con bajo acoplamiento y alta cohesión al separar claramente la lógica de negocio, la interfaz y el acceso a datos, permitiendo que las clases están enfocadas en una sola tarea y dependan mínimamente unas de otras.

El patrón GoF se pueden inferir patrones donde una clase se encarga de crear objetos como usuarios o productos sin exponer la lógica de creación al cliente, facilitando la extensibilidad del sistema. El patrón Singleton es aplicable para clases que manejan conexiones a la base de datos, asegurando que solo haya una instancia en todo el sistema.

El principio de Responsabilidad Única (SRP) se aplica en el proyecto al separar la lógica en capas, como controladores, servicios y repositorios, donde cada clase cumple una función específica.

5.Referencias

Se uso https://openai.com/index/chatgpt/ para la explicacion detallada de la seccion pipeline