NEW - TEAM GESTIÓN ENFOCADA A BBDD Y JAVAFX

Trabajo realizado por:
Víctor Marín Escribano
Aitor Aróstegui Navarro
Cristian Ortega González
Nicolás Ortega Fernández

ÍNDICE

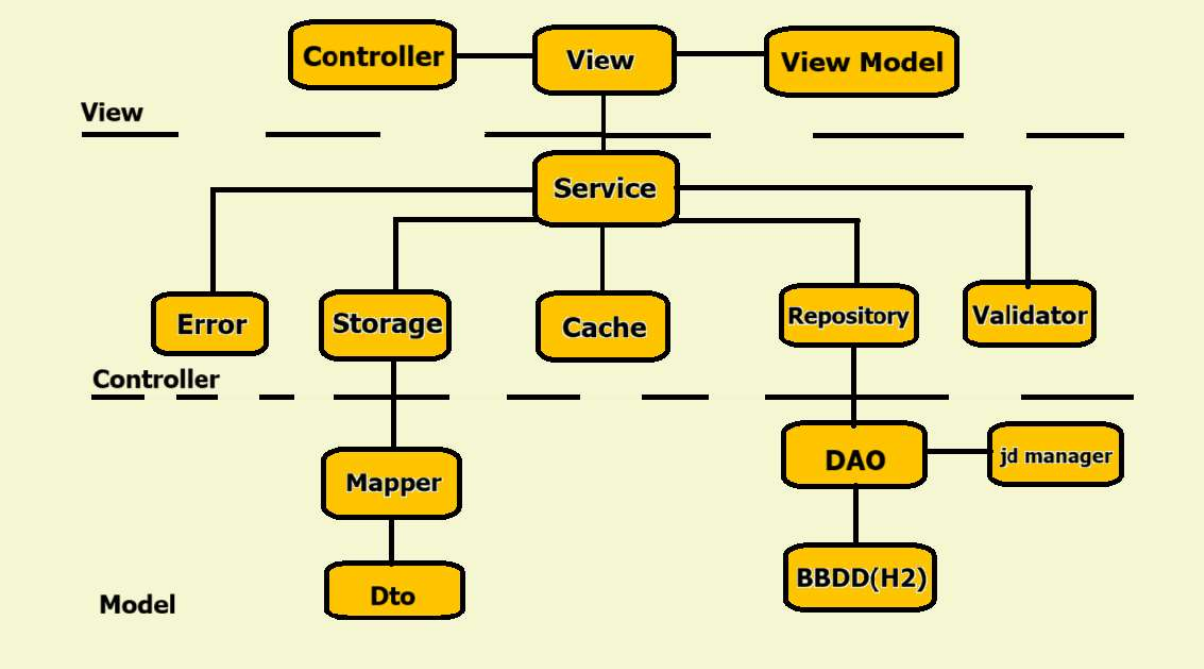
PRESENTACIÓN DEL PROYECTO

Los integrantes del grupo somos Aitor Aróstegui, Nicolás Ortega, Cristian Ortega y Víctor Marín, siendo este último el jefe de proyecto. De lo que trata este ejercicio es de gestionar una lista de personas. El proyecto esta enfocado al deporte, en este caso a la gestión del equipo de fútbol New-Team. Los jugadores de este podrán ser importados y exportados mediantes estas extensiones:

- CSV
- JSON
- XML
- BIN
- ZIP

Gracias a JavaFx hemos logrado que se vea bonito y no en terminal como en la versión anterior. En cuanto al almacenamiento, en esta 2a versión hemos implementado la BBDD H2. También hemos testeado todo lo posible.

DIAGRAMA DE ARQUITECTURA DE SOFTWARE



PATRONES UTILIZADOS

Los patrones que hemos utilizado son:

- **Patrón CRUD ->** En nuestro caso es un sistema de almacenamiento que tiene Crear, Leer, Actualizar, y Borrar, es decir, create, read, update y delete.
    - Crear Miembro
    - Actualizar Miembro / Actualizar Entrenador / Actualizar Persona / Actualizar Jugador
    - Eliminar Miembro
    - Cargar datos a fichero
- **Patrón LRU ->** Es una cache en la que se elimina el elemento que más tarda en usarse para sustituirlo con elementos más recientes.

Tenemos la parte de CacheImplementationLru en la cual empezamos con una data class llamada cache entrada que encapsula el valor almacenado y el timestamp del último acceso. Dentro de CacheImplementationLru, implementa una cache con capacidad máxima, usa un ConcurrentHasMap para el almacenamiento concurrente y gestiona automáticamente la eliminación del elemento menos usado cuando se alcanza la capacidad máxima.

ELEMENTOS DE LA ARQUITECTURA

Arquitectura MVC:

- **Model**: representa los datos de la aplicación, por ejemplo los modelos de las personas.
- **View**: interfaz con la que el usuario interactúa. En nuestro caso hemos usado JavaFx para esto.
- Controlador: lleva a cabo la interacción entre modelo vista y servicio.

Además de esto tenemos:

- Servicio: lógica de negocio. Valida y filtra los datos entre otras cosas.
- Almacenamiento: Storage y Repository son los encargados de guardar y cargar datos desde CSV, JSON, BIN, XML y ZIP.

Para todo el proyecto también se ha hecho uso de las siguientes librerías:

```
//libreria de drivers de base de datos H2
implementation("com.h2database:h2:2.3.232")
//jdbi libraries
implementation("org.jetbrains.kotlin:kotlin-reflect")
implementation("org.jdbi:jdbi3-core:3.48.0")
```

```
implementation("org.jdbi:jdbi3-sqlobject:3.48.0")
implementation("org.jdbi:jdbi3-kotlin:3.48.0")
implementation("org.jdbi:jdbi3-kotlin-sqlobject:3.48.0")
implementation("org.jetbrains.kotlin:kotlin-stdlib:1.9.23")
// librerías para el logger
implementation("org.lighthousegames:logging:1.3.0")
implementation('ch.qos.logback:logback-classic:1.5.13')
//implementation("io.github.pdvrieze.xmlutil:serialization-
jvm:0.84.3")
implementation("io.github.pdvrieze.xmlutil:serialization-
jvm:0.90.3")
//Json serializable
implementation("org.jetbrains.kotlinx:kotlinx-serialization-
core:1.6.2")
implementation("org.jetbrains.kotlinx:kotlinx-serialization-
json:1.6.2")

// mockito
testImplementation("org.mockito:mockito-junit-jupiter:5.12.0")
testImplementation("org.mockito.kotlin:mockito-kotlin:5.3.1")
// test normales
testImplementation("org.junit.jupiter:junit-jupiter-
api:${junitVersion}")
testRuntimeOnly("org.junit.jupiter:junit-jupiter-
engine:${junitVersion}")
//caffeine cache
implementation("com.github.ben-manes.caffeine:caffeine:3.2.0")
//Result exeptions
implementation("com.michael-bull.kotlin-result:kotlin-result:2.0.0")
//BCrypt
implementation("org.mindrot:jbcrypt:0.4")
// Open Brwoser
implementation("com.vaadin:open:8.5.0.4")
//koin
constraints {
    implementation("io.insert-koin:koin-core:3.5.6")
}
implementation("io.insert-koin:koin-core")
testImplementation "org.jetbrains.kotlin:kotlin-test:2.1.0"
// Jacoco
implementation("org.jacoco:org.jacoco.core:0.8.12")
testImplementation "org.jetbrains.kotlin:kotlin-test:1.9.23"
```

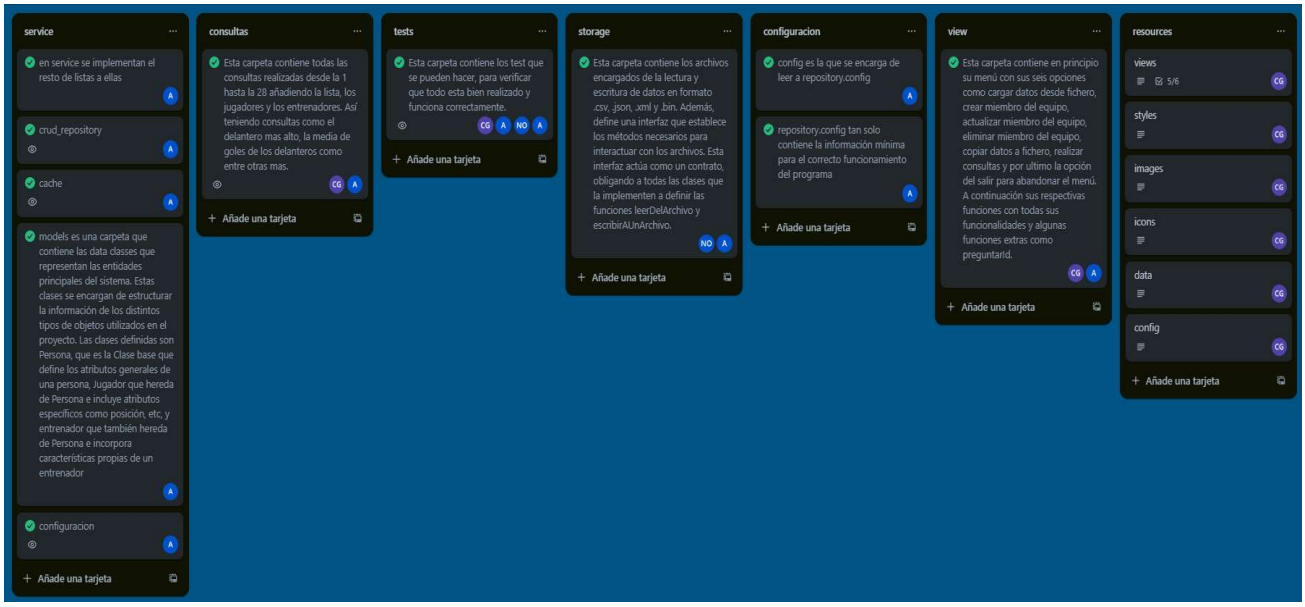
EXPLICACIÓN DE LAS TECNOLOGÍAS

A continuación, un listado de todas las tecnologías de las que se ha hecho uso a lo largo de la práctica además del porque han sido usadas:

- **IntelliJ**: entorno de desarrollo principal usado para escribir y ejecutar el código. Se ha hecho uso de este entorno de desarrollo debido a que es el que se ha usado a lo largo de todo el curso, en parte por su facilidad a la hora de usar proyectos colaborativos y porque garantiza muy bien la calidad del código.
- **ScreenBuilder**: software usado para diseñar la interfaz gráfica (GUI) de la aplicación. No solo es el software usado a lo largo del curso, sino que además es muy intuitivo y útil para perfiles con menos experiencia.
- **Visual Studio Code**: entorno de desarrollo secundario para revisar el código de los compañeros. Es el entorno más rápido si lo que se busca es simplemente visualizar código.
- **Git**: control de versiones para la gestión de cambios en el código. Se ha usado, además de porque es el estándar, ya que permite guardar un historial de cambios muy útil.
- **GitHub**: ha servido para alojar el repositorio del proyecto y poder gestionar los cambios desde ahí. Se ha usado ya que ofrece una integración directa con Git.
- **GitFlow**: para el flujo de trabajo, representado con ramas. Usado ya que muestra claramente la estructura en el desarrollo, lo que ha sido muy útil.
- **Windows Terminal:** para ejecutar los comandos y gestionar el proyecto desde terminal. Usada debido a que es la predeterminada y cubre todas las necesidades del proyecto.
- **Trello**: para el reparto de tareas. Se exigió en la anterior práctica y es muy cómodo en cuanto a gestionar que parte hace cada miembro del equipo
- **Canva**: para el diseño de la portada. Usado por su facilidad para hacer portadas en parte gracias a las plantillas.
- **Word**: para poder redactar la Documentación del proyecto. Usado ya que es el estándar en cuanto a procesadores de texto.

PLANIFICACIÓN. TRELLO

Para el reparto de tareas y poder planificar el proyecto eficientemente se ha hecho uso de Trello, un software de administración muy intuitivo que ha sido de gran ayuda. A continuación, el Trello del proyecto:



REQUISITOS FUNCIONALES, NO FUNCIONALES Y DE INFORMACIÓN.

Requisitos Funcionales:

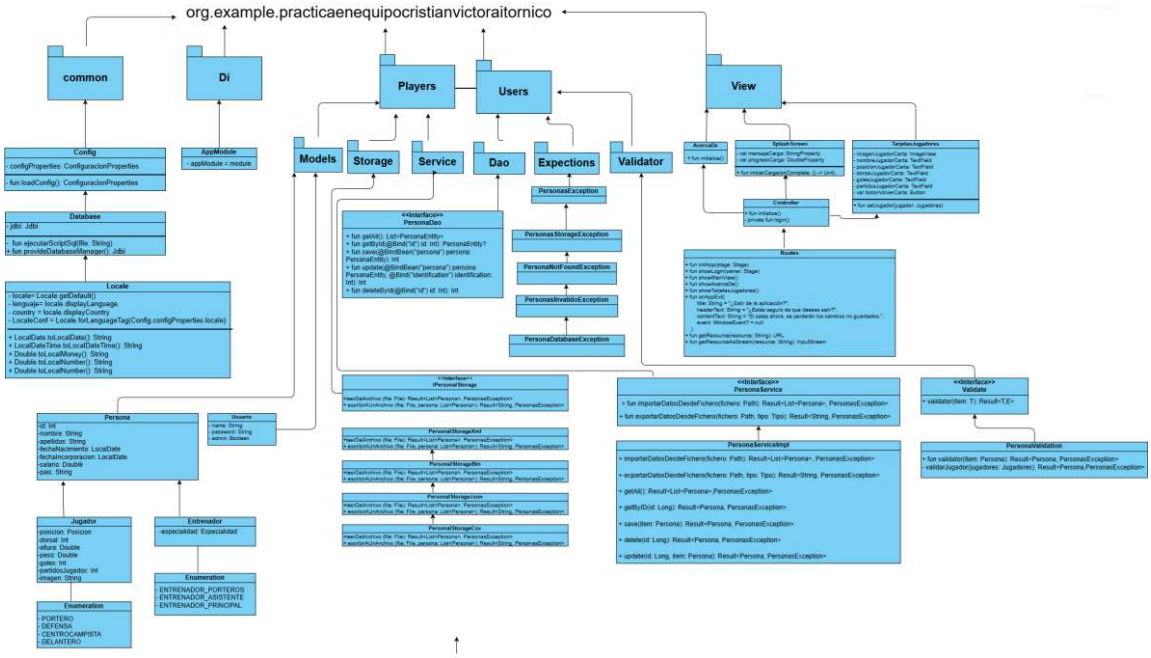
- **RF1:** se realizan operaciones CRUD (create, read, update y delete) con el objetivo de gestionar los miembros del equipo.
- **RF2:** se asigna un ID único a cada miembro del equipo guardado.
- **RF3:** se almacenan nombre, apellidos, fecha de nacimiento, fecha de incorporación, salario, imagen y pais.
- **RF9:** se implementa una caché con filosofía LRU con un tamaño y caducidad de datos parametrizables a través de un fichero de configuración.
- **RF12:** inicio de sesión con una base de datos de usuarios en la que se guarda el usuario y su contraseña utilizando la función de hash criptográfica de BCrypt.
- **RF10:** se implementa la interfaz de usuario con un logotipo personalizados y únicos para la aplicación.

Requisitos No Funcionales:

- **RNF1**: caché para la mejora de velocidad.
- **RNF2**: 100% de test.
- **RNF3**: ejecución mediante Jar y Exe.
- **RNF4**: trabajo en remoto mediante GitHub.

- **RNF5**: datos totalmente validados

DIAGRAMA DE CLASES



En el proyecto esta subida la imagen para verla con mejor resolución y poder ampliarla

PRINCIPIOS SOLID

- **1. Principio de responsabilidad única:** cada clase debe tener una única responsabilidad como por ejemplo la clase jugadores que solo es exclusivamente para los jugadores o también como la clase entrenadores que como indica el nombre es para los entrenadores, es decir, que cada clase cumple su función.
- **2 -> Principio de Abierto/Cerrado:** el principio dice que una clase debe estar abierta para añadir nuevas funcionalidades, sin modificar su código. En este caso, por ejemplo, en personaServiceImplementation tenemos el ejemplo de a la hora de meter un nuevo formato de archivo, estamos extendiendo, pero no modificando.
- **3 -> Principio de sustitución de Liskov:** tenemos subclases para usarse en lugar de su clase, como por ejemplo tenemos la clase persona y de ellas las subclases que heredan de ella que son entrenador y jugador

- **4 -> Principio de Segregación de Interfaces:** es mejor tener interfaces específicas que no una gran interfaz como en este caso por ejemplo la interfaz del CrudPersonas que es exclusivamente el crud para Persona o la interfaz RepositoryCrud, que es el crud del repositorio.
- **5 -> Principio de Inversión de Dependencias:** nuestro código depende de interfaces y de abstract class, como abstract class persona o las interfaces RepositorioCrud.

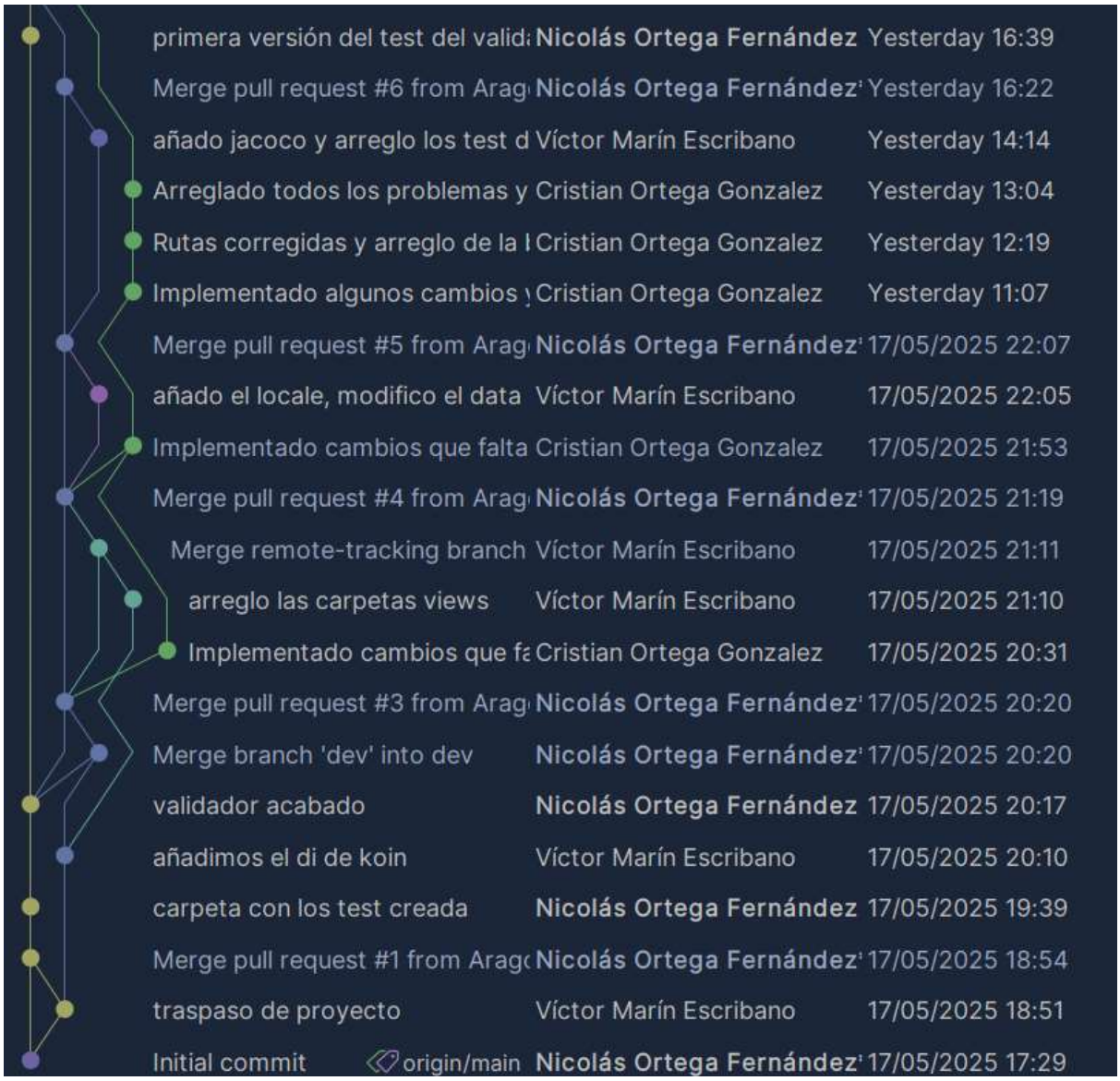
DISEÑO DE TEST

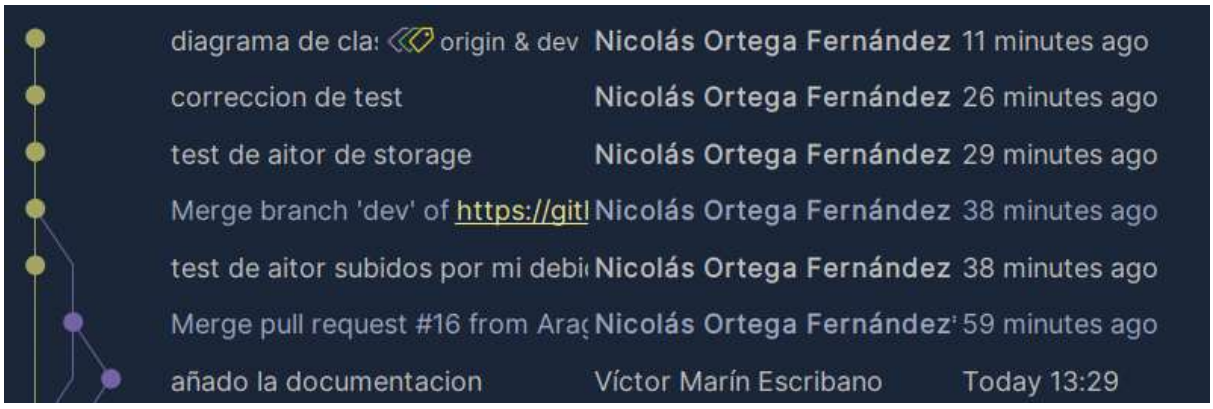
Para llevar a cabo los test se ha hecho uso de los propios test de Kotlin y de la librearía de mockito, para testear cosas como el servicio:

```
// mockito
testImplementation("org.mockito:mockito-junit-jupiter:5.12.0")
testImplementation("org.mockito.kotlin:mockito-kotlin:5.3.1")
```

FLUJO DE TRABAJO DE GITFLOW

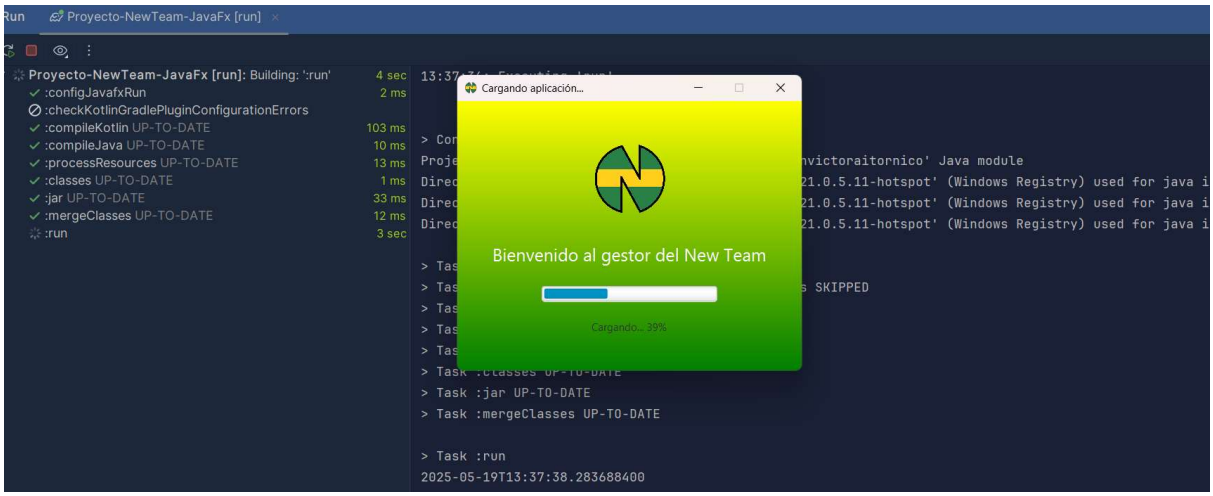
A continuación, el flujo de trabajo:


primera versión del test del valid Nicolás Ortega Fernández Yesterday 16:39
Merge pull request #6 from Arag Nicolás Ortega Fernández Yesterday 16:22
añado jacoco y arreglo los test d Víctor Marín Escribano Yesterday 14:14
Arreglado todos los problemas y Cristian Ortega Gonzalez Yesterday 13:04
Rutas corregidas y arreglo de la I Cristian Ortega Gonzalez Yesterday 12:19
Implementado algunos cambios y Cristian Ortega Gonzalez Yesterday 11:07
Merge pull request #5 from Arag Nicolás Ortega Fernández 17/05/2025 22:07
añado el locale, modifico el data Víctor Marín Escribano 17/05/2025 22:05
Implementado cambios que falta Cristian Ortega Gonzalez 17/05/2025 21:53
Merge pull request #4 from Arag Nicolás Ortega Fernández 17/05/2025 21:19
Merge remote-tracking branch Víctor Marín Escribano 17/05/2025 21:11
arreglo las carpetas views Víctor Marín Escribano 17/05/2025 21:10
Implementado cambios que fa Cristian Ortega Gonzalez 17/05/2025 20:31
Merge pull request #3 from Arag Nicolás Ortega Fernández 17/05/2025 20:20
Merge branch 'dev' into dev Nicolás Ortega Fernández 17/05/2025 20:20
validador acabado Nicolás Ortega Fernández 17/05/2025 20:17
añadimos el di de koin Víctor Marín Escribano 17/05/2025 20:10
carpeta con los test creada Nicolás Ortega Fernández 17/05/2025 19:39
Merge pull request #1 from Arag Nicolás Ortega Fernández 17/05/2025 18:54
traspaso de proyecto Víctor Marín Escribano 17/05/2025 18:51
Initial commit origin/main Nicolás Ortega Fernández 17/05/2025 17:29

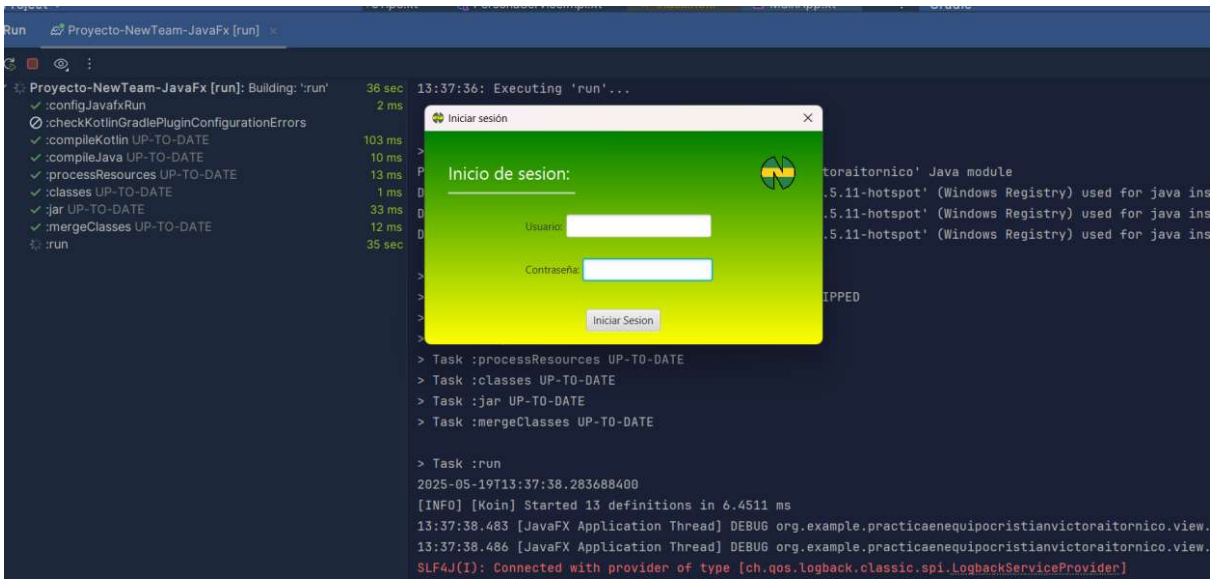


EJEMPLOS DE EJECUCIÓN

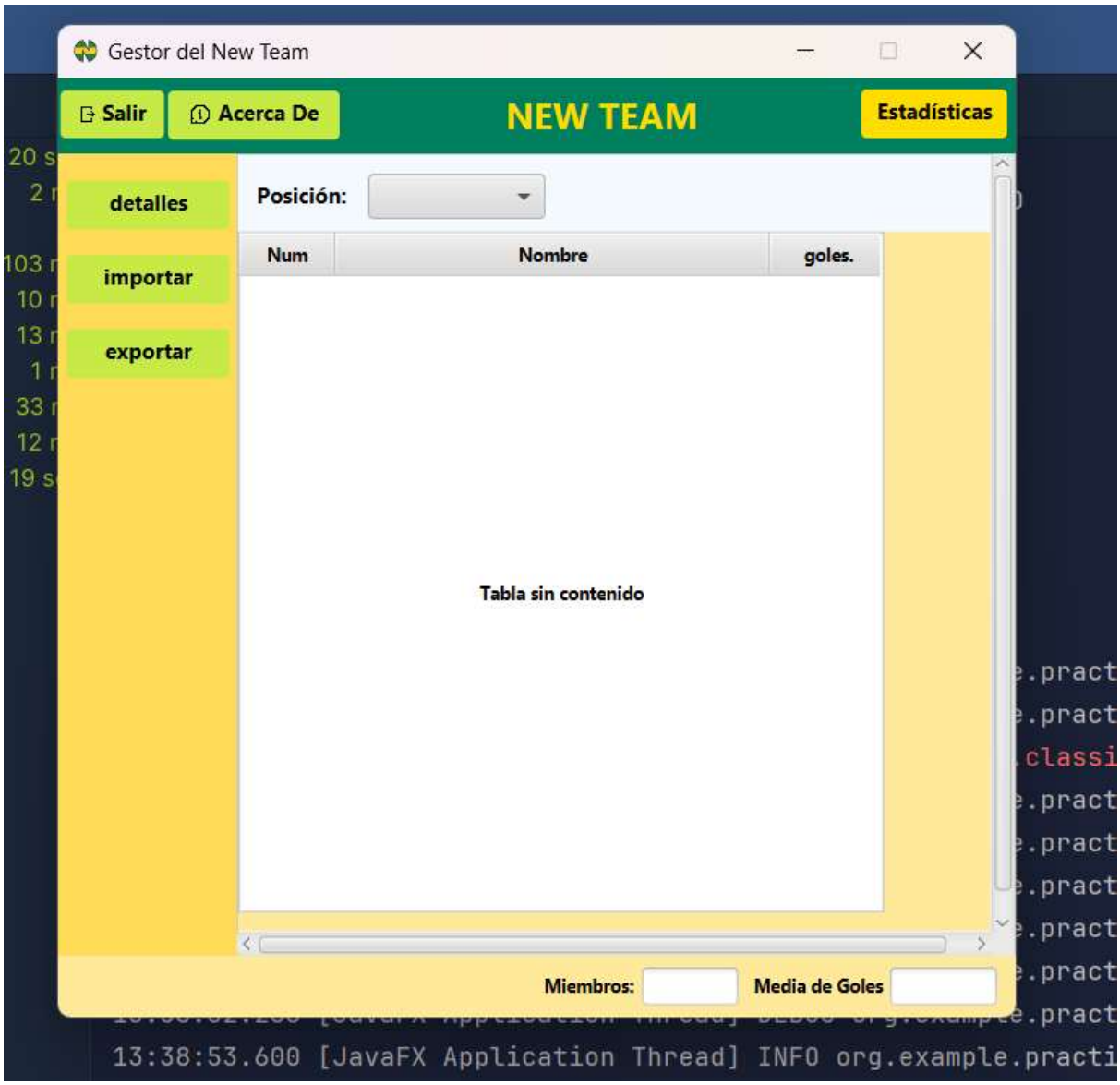
Splash Screen:



Inicio de sesión:



Gestor:



Código en terminal desde la ejecucción:
[INFO] [Koin] Started 13 definitions in 6.4511 ms

13:37:38.483 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.view.routes.RoutesManagerKt -- Inicializando RoutesManager

13:37:38.486 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.view.routes.RoutesManagerKt -- Iniciando aplicación con pantalla Splash

SLF4J(I): Connected with provider of type [ch.qos.logback.classic.spi.LogbackServiceProvider]

13:37:43.920 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.view.routes.RoutesManagerKt -- Mostrando pantalla de login

13:38:52.259 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.common.database.DatabaseManagerKt -- Obteniendo config properties...

13:38:52.262 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.common.config.Config -- creando config.properties

13:38:52.262 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.common.config.Config -- cargando propiedades

13:38:52.266 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.common.database.DatabaseManagerKt -- Config.url=jdbc:h2:./jugadores, initData=true, initTables=true

13:38:52.268 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.common.database.DatabaseManager -- inicializando jdbi

13:38:53.600 [JavaFX Application Thread] INFO org.example.practicaenequipocristianvictoraitornico.users.dao.UsersDaoKt -- obteniendo dao de usuarios

13:38:54.533 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.users.service.UsersServiceImplKt -- contraseña valida

13:38:54.537 [JavaFX Application Thread] INFO org.example.practicaenequipocristianvictoraitornico.view.controller.UsersController -- Usuario admin ha iniciado sesión correctamente

13:38:54.537 [JavaFX Application Thread] DEBUG org.example.practicaenequipocristianvictoraitornico.view.routes.RoutesManagerKt -- Mostrando vista principal

ESTIMACIÓN ECONÓMICA

Para llevar a cabo el cálculo de lo que se estima que podría costar este proyecto hay que tener los siguientes factores en cuenta:

- **Duración del proyecto**: 2-3 semanas.
- **Integrantes**: en este proyecto se han visto involucradas 4 personas.
- **Entrega**: además de la aplicación funcional, para la entrega era necesaria esta documentación, un vídeo explicando el funcionamiento de la aplicación y una presentación presencial.
- **Horas de trabajo**: entre todos los integrantes, en la totalidad del proyecto se estiman alrededor de 300h de trabajo.

En base a esto, y poniendo un salario de **12€/h:**

300h x 12€/h = **3.600€**

A estos 3.600€ se le podrían haber sumado extras como:

- Sofware de pago. No añadido ya que el centro nos lo proporciona.
- Costes de edición y producción del vídeo. No añadido ya que se han usado herramientas gratuitas como Discord para la reunión y OBS para la grabación.
- Transporte. No añadido ya que en el caso de este grupo no se ha llevado a cabo.

VÍDEO

https://youtu.be/ZLoX7L4xVSw