Proyecto New Team

Pablo Zuil Armenteros
Pablo De la Fuente De los Santos
Lucía Fuertes Cruz

Índice

Introducción al proyecto

METODOLOGIAS USADAS

- Lenguaje de programación: Kotlin (elegido por su implementación en el aula).
- Arquitectura y diseño: basado en la eficiencia y modularidad.
- Pruebas y validaciones: garantizan la cobertura del programa.

FUNCIONALIDADES

- Gestión de personal: guarda los datos específicos de cada empleado.
- Operaciones CRUD: permite realizar operaciones para modificar, crear, o eliminar entre otras al personal.
- Optimización de consultas: uso de una cache LRU para optimizar el uso de funciones.
- Importación/Exportación: Importa y exporta datos en formatos CSV y JSON
- Consultas: ofrece operaciones de filtrado y buscado en la devolución de datos.

Estructura del programa

TABLA DE ROLES

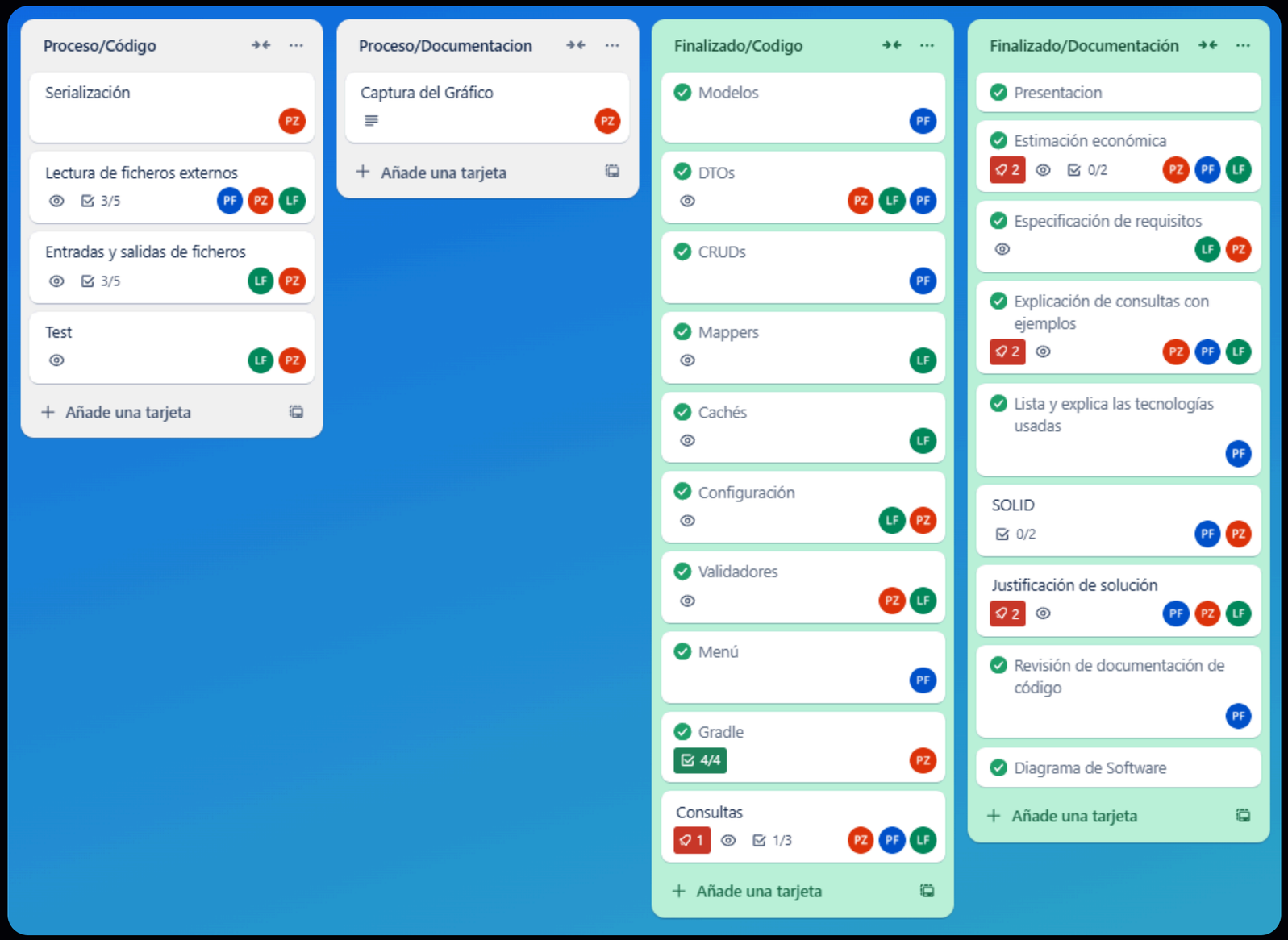
- Pablo Zuil: jefe del proyecto
- Lucía Fuertes: empleada
- Pablo de la Fuente: empleado

TAREAS PRINCIPALES

1. Diseño e Implementación del Sistema
2. Desarrollo del Servicio con Caché LRU
3. Validación y Gestión de Datos
4. Menú de Funcionalidades
5. Implementación de Consultas Específicas
6. Requisitos Técnicos y Metodología
7. Documentación del Proyecto

PRINCIPALES RIESGOS

- Fallo al exportar o importar datos
- Conflictos con Git y sus comandos
- Sobrecarga de trabajo en tiempo limitado
- Cobertura esperada no alcanzada por fallos en los test


Proceso/Código
Serialización
PZ
Lectura de ficheros externos
3/5
PF PZ LF
Entradas y salidas de ficheros
3/5
LF PZ
Test
LF PZ
Añade una tarjeta
Proceso/Documentacion
Captura del Gráfico
PZ
Añade una tarjeta
Finalizado/Codigo
Modelos
PF
DTOs
PZ LF PF
CRUDs
PF
Mappers
LF
Cachés
LF
Configuración
LF PZ
Validadores
PZ LF
Menú
PF
Gradle
4/4
PZ
Consultas
1
1/3
PZ PF LF
Añade una tarjeta
Finalizado/Documentación
Presentacion
Estimación económica
2
0/2
PZ PF LF
Especificación de requisitos
LF PZ
Explicación de consultas con ejemplos
2
PZ PF LF
Lista y explica las tecnologías usadas
PF
SOLID
0/2
PF PZ
Justificación de solución
2
PF PZ LF
Revisión de documentación de código
PF
Diagrama de Software
Añade una tarjeta

Arquitectura y tecnologías usadas

ARQUITECTURA DEL PROGRAMA

- Capa de Presentación: interfaz de usuario para ver y gestionar datos desde el menú
- Capa de Lógica de Negocio: Procesa la información, valida datos y ejecuta consultas.
- Capa de Acceso a Datos: Gestiona la interacción con archivos de almacenamiento.

LIBRERÍAS CLAVE

- Sistema de logs: org.lighthousegames:logging
- Implementación de Logger: ch.qos.logback:logback-classic
- Generación de documentación: org.jetbrains.dokka:dokka-gradle-plugin
- Manejo de datos en JSON: org.jetbrains.kotlinx:kotlinx-serialization-json
- Manejo de datos en XML: io.github.pdvrieze.xmlutil:serialization-jvm

TECNOLOGÍAS UTILIZADAS

- Lenguaje de programación: Kotlin
- Entorno de desarrollo: IntelliJ IDEA
- Sistema de control de versiones: Git
- Gestor de dependencias: Gradle

Requisitos del programa

REQUISITOS FUNCIONALES

1. Proceso de datos
2. Almacenamiento de datos
3. Lectura y escritura de ficheros
4. Configuración que gestiona el programa
5. Consulta de datos

REQUISITOS NO FUNCIONALES

1. Que tenga un archivo jar ejecutable
2. Correcto funcionamiento del programa mediante el uso de test
3. Adaptación del almacenamiento según las fuentes
4. Correcto filtrado de información según condiciones

Servicio Cache LRU

EXPLICACION

Para mejorar la gestión del personal del club, se implementó un sistema de caché LRU (Least Recently Used). Esta caché ayuda a acelerar el acceso a los datos más utilizados, reduciendo el tiempo de respuesta y optimizando el rendimiento del programa.

CREACION DE ARCHIVOS

- Cache
- CacheImpl
- CacheImplTest

Importacion y Exportacion de datos

EXPLICACION

El programa permite importar y exportar datos en formatos CSV y JSON, ya que no se logró implementar completamente la compatibilidad con XML y Binario. Para ello, se utilizaron librerías para manejar la serialización y deserialización de datos en JSON.

CREACION DE ARCHIVOS

- PersonalStorage
- PersonalStorageImpl
- PersonalStorageFile
- PersonalStorageCsv
- PersonalStorageJson
- PersonalStorageImplTest
- PersonalStorageCsvTest
- PersonalStorageJsonTest

Consultas

DECISIONES A TENER EN CUENTA

En el proyecto, el menú incluye una opción que permite seleccionar qué consultas ejecutar. Para ello, se ha implementado una función específica para cada consulta, facilitando su uso tanto para el cliente como para el desarrollador.

EJEMPLOS DE CONSULTAS

- Estimación total del coste de la plantilla:
  personalRepository.getAll().sumOf { it.salario!!.toInt() }

- Los entrenadores que se han incorporado al club en los últimos 5 años:
  personalRepository.getAll().filterIsInstance<Entrenador>
  ().filter { it.fechaIncorporacion >= 2015.toString() }

Pruebas y validación

COBERTURA

- es una métrica que indica qué tan bien tu código está siendo probado.
- Media total del coverage: 44%
  - 62% (Class)
  - 50% (Method)
  - 39% (Line)
  - 25% (Branch)

EXPLICACIÓN

- Las pruebas de validación se centrarán en verificar que el programa cumple con los requisitos funcionales y no funcionales definidos.

TIPO DE PRUEBAS UTILIZADAS

- Uso de Junit para realizar pruebas unitarias
- Uso de MockK

Presupuestos y tiempos

SUPUESTOS DEL PROYECTO:

- Duración del proyecto: 2 semanas
- Equipo de desarrollo: 3 personas (1 Líder de Proyecto, 2 Desarrolladores)
- Tecnología: Kotlin, caché LRU, soporte para CSV, XML, JSON y Binario
- Requerimiento: Producto funcional, documentación completa y presentación final

COSTOS DEL PERSONAL

Rol	Coste mensual	Duracion	Coste total
Lider de proyecto	2500	2 semanas	1250
Desarrollador	2000	2 semanas	1000
Desarrollador	2000	2 semanas	1000

Presupuestos y tiempos

TIEMPOS DEL PROYECTO

Fase del Proyecto	Horas Estimadas	Horas Reales
Planificación	3	10
Diseño	15	15
Implementación	60	71
Pruebas	20	30
Documentación	15	24
Total	**120**	**150**