PFS003 - SIGEA

Developement Team

Indice

Estado

[Aceptado]

I. Propuesta de Estructura de Archivos del Proyecto

1.1. Estructura General del Proyecto

Debido al cambio de arquitectura, se debe modificar tambien la estructura propuesta.

Para esta nueva estructura, se considera la arquitectura actual del proyecto (ver `ADR002`), Capas + MVC + Clean Architectura, se propone usar una carpeta `modules/` donde cada integrante pueda desarrollar un conjunto de requisitos de forma individual reduciendo los conflictos que puedan llegar a darse.

El flujo de cambios y versiones se hara a traves de `git flow` en un unico repositorio, pero, a diferencia de la anterior estructura, la actual puede generar mas conflictos, hasta el punto de ocasionar problemas graves si es que la comunicacion entre los integrantes no es la adecuada,

por lo que es necesario contar con una persona que maneje y/o gestione los `pull requests`, `merges` con `develop` y `main`, ademas del control de versiones del sistema. Al igual que en el anterior documento (PFS001), cada integrante dispondra de una rama especifica para su modulo y solo tocara lo que le corresponde con constante comunicacion con el encargado del control de versiones.

Si se desea interactuar con otro modulo se tiene que realizar una fusion (`merge`) hacia la rama develop de la rama donde se encuentra desarrollando el modulo solicitante con la rama del modulo objetivo.

Para coordinar los componentes fuera del dominio (`src/main/` o `src/main/java/com/zentry/sysgec`) se realizaran reuniones generales donde se discutira y observara el avance de cada integrante, con el objetivo de poder realizar la orquestacion y configuracion necesarias de acuerdo a la configuracion inidividual de cada modulo desarrollado por cada integrante.

Siguiendo el siguiente esquema:

```
[ UI (Capa de presentación) ]
    Controlador (C) - Recibe requests del usuario
    Modelo (M) - Datos que se muestran (DTOs, ViewModels)

[ Capa de aplicación / casos de uso ]
    Orquesta la lógica de negocio
    Llama a servicios del dominio

[ Capa de dominio ]
    Entidades, lógica de negocio pura
    Interfaces (repositorios, servicios)

[ Capa de infraestructura ]
    Implementaciones reales: bases de datos, APIs externas, etc.
```

Se planteo la siguiente estructura general para el proyecto:

```
app/
   src/
       main/
           java/
               com/zentry/sigea/
                   config/              # Configuración de Spring, Beans, Seguridad, CORS, etc
                   utils/               # Utilidades genéricas (fechas, strings, etc.)
                   presentation/          # Capa de presentación (UI + Controladores)
                       api/            # @RestController comunicacion con frontend
```

```
                    models/          # DTOs y ViewModels que la vista necesita

                services/             # Capa de aplicación (Casos de uso)
                    usecases/         # Casos de uso concretos
                    interfaces/       # Interfaces para los servicios del dominio

                core/                 # Capa de dominio
                    entities/         # Entidades de dominio
                    value-objects/    # Objetos de valor (si aplica)
                    repositories/     # Interfaces de los repositorios
                    services/         # Servicios de dominio puros

                infrastructure/       # Capa de infraestructura
                    repositories/     # Implementaciones de los repositorios
                    database/         # Configuración y acceso a base de datos
                        entities/         # Entidades JPA (@Entity)
                        mappers/          # Mappers entre dominio  infraestructura
                    external-apis/    # Clientes de APIs externas
                    config/           # Configuración de infraestructura (env, logger, etc
        resources/
            application.properties       # Configuración principal (por defecto)
            application.yml              # O alternativa en formato YAML
            i18n/                        # Archivos de internacionalización (mensajes por
                messages_en.properties
                messages_es.properties
            db/
                migration/               # Scripts SQL para migraciones (ej: Flyway, Liquib
                    V1__init.sql
                seed/                    # Datos de prueba para poblar la DB
            logback-spring.xml           # Configuración del sistema de logs (Logback)
            banner.txt                   # Banner que se muestra al arrancar la app
            META-INF/                    # Archivos especiales como manifest, spring.facto
    test/
        java/
            com/miempresa/myapplication/
                presentation/
                services/
                core/
                infrastructure/
                ...
    docs/
```

1.2. Explicacion Detallada de Cada Capa

1.2.1. Capa de Presentacion

```
presentation/
   api/          # Controladores REST (MVC - Controller)
   models/       # DTOs y ViewModels (MVC - Model)
```

- *Contenido*:
  - Controladores (@RestController): Reciben peticiones HTTP del frontend, validan datos, y delegan la lógica a la capa de aplicación.
  - Modelos de presentación (DTOs / ViewModels): Estructuras de datos para entrada y salida (request y response).
- *Anotaciones típicas*
  - @RestController
  - @RequestMapping
  - @GetMapping, @PostMapping, etc.
  - @Validated, @RequestBody, @PathVariable
  - @ResponseStatus

1.2.2. Capa de Aplicacion

```
services/
   usecases/       # Casos de uso concretos
   interfaces/     # Interfaces que exponen servicios de aplicación
```

- *Contenido*:
  - Casos de uso: Representan acciones del sistema (“registrar usuario”, “crear reserva”, etc.).
  - Orquestan la interacción entre controladores, servicios del dominio y repositorios.
  - No contienen lógica de negocio compleja, solo coordinación.
  - Usan interfaces del dominio (p. ej. UserRepository) para acceder a la persistencia.
- *Anotaciones típicas*:
  - @Service
  - (a veces) @Transactional para mantener la consistencia de operaciones.

1.2.3. Capa de Dominio

```
core/
   entities/         # Entidades del dominio (con lógica de negocio, puras sin @Entity)
   value-objects/    # Objetos de valor (inmutables, sin identidad)
   repositories/     # Interfaces de repositorios
   services/         # Servicios de dominio puros (reglas de negocio)
```

- *Contenido*:
  - Entidades: Clases con identidad y reglas de negocio propias.
  - Objetos de valor: Clases inmutables sin identidad (ej. Email, Money).
  - Servicios de dominio: Lógica de negocio que involucra múltiples entidades.
  - Interfaces de repositorio: Contratos que la infraestructura debe implementar.
- *Anotaciones típicas*:
  - Generalmente no usan anotaciones de Spring.
  - Si se usan, que sean neutrales (@Entity si usas JPA, pero se debe evitar con mappers en infraestructura).

1.2.4. Capa de Infraestructura

```
infrastructure/
   repositories/     # Implementaciones concretas de los repositorios del dominio
   database/         # Configuración y entidades JPA, mappers, etc.
       entities/          # Entidades JPA (@Entity)
       mappers/           # Mappers entre dominio   infraestructura
   external-apis/    # Clientes HTTP para APIs externas
   config/           # Configuración técnica (env, logger, beans, etc.)
```

- *Contenido*:
  - Implementaciones concretas de las interfaces definidas en el dominio.
  - Adaptadores a tecnologías: ORM, REST clients, mensajería, etc.
  - Mappers entre entidades JPA entidades del dominio.
  - Configuración de beans, propiedades, seguridad, etc.
- *Anotaciones típicas*:
  - @Repository
  - @Component
  - @Configuration

– @Bean
– @EnableJpaRepositories, @Entity, etc.