

PFS002 - SIGEA

Ordoñez Silva, Yonel Jr.

Indice

Estado	1
I. Propuesta de Estructura de Archivos del Proyecto	1
1.1. Estructura General del Proyecto	1
1.2. Explicacion Detallada de Cada Capa	4
1.2.1. Capa de Presentacion	4
1.2.2. Capa de Aplicacion	5
1.2.3. Capa de Dominio	5
1.2.4. Capa de Infraestructura	6

Estado

[Propuesto]

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 Arquitectura, 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`, además del control de versiones del sistema. Al igual que en el anterior documento (PFS001), cada integrante dispondrá de una rama específica para su módulo y solo tocará lo que le corresponde con constante comunicación con el encargado del control de versiones.

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

Para coordinar los componentes fuera del dominio (`src/main/` o `src/main/java/com/zentry/sysgec`) se realizarán reuniones generales donde se discutirá y observará el avance de cada integrante, con el objetivo de poder realizar la orquestación y configuración necesarias de acuerdo a la configuración individual de cada módulo 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 plantea 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 comunicación 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, Liquibase)
            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.factories

test/
    java/
        com/miempresa/myapplication/
            presentation/
            services/
            core/
            infrastructure/
            ...

docs/

```

Una posible mejora que se podría implementar, con el fin de mejorar la organización y modularización, es añadir en una carpeta `modules/` que contenga cada módulo de requisitos del proyecto; luego dentro de cada módulo estructurar las capas.

```
app/
  src/
    main/
      java/
        com/zentry/sigea/
          # Podemos usar módulos para estructurar cada módulo de requisitos
          modules/          # Agrupación por módulos funcionales
            ParticipantsRegGest/  # Módulos de Registro y Gestión de Participantes
              presentation/      # Capa de presentación (UI + Controladores)
                api/             # @RestController comunicación 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
                external-apis/   # Clientes de APIs externas
                config/          # Configuración de infraestructura (env, logs, etc)
              Certification/
              ...
```

1.2. Explicación Detallada de Cada Capa

1.2.1. Capa de Presentación

```
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

```

infraestructure/
  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.