Documento de diseño del sistema

Sentido del documento

Este documento describe el diseño del sistema de software para este proyecto. Se propondrán soluciones para los problemas de diseño y se describirán las decisiones de diseño tomadas. El documento de diseño del sistema (SDD) es un documento de referencia para el diseño del sistema de software y la arquitectura del software. Es un documento vinculante para el desarrollo del software y la implementación de los subsistemas.

El contenido de este documento sigue y se ajusta a los requisitos especificados en el IEEE 1016–1998[1]

Índice

1. Introducción

1.1. Overview

1.2. Definitions, acronyms, and abbreviations

Referencias

[1] «IEEE Guide for Software Requirements Specifications», Institute of Electrical y Electronics Engineers, IEEE Std. 1016-1998, 1998. DOI: 10.1109/IEEESTD.2009.5167255.

2. Design Goals

3. Composición del sistema, arquitectura y capas

La aplicación sigue una arquitectura por capas, añadiendo el patrón «Façade» para la comunicación entre capas. El objetivo de dicha arquitectura es facilitar el desarrollo, mantenimiento y escalabilidad de la aplicación.

3.1. Capas del sistema y sus responsabilidades

- **Capa de presentación:** Gestiona la interacción con el exterior (usuarios) y muestra información, centralizando mediante *FachadaAplicacion*. Las operaciones de esta capa son canalizadas a través de la *FachadaGUI*, que es la responsable de la interfaz de usuario.
- **Capa aplicación:** Es la capa de coordinación entre la capa de presentación y la capa de negocio. A ella la GUI le envía las peticiones de los usuarios y el estado de la aplicación. Delegando en la capa de negocio (Control) la lógica de negocio.
- **Capa de control:** Contiene la lógica de negocio y las reglas de negocio. Centralizada en *ControladorAplicacion*, controla el flujo de la aplicación y delega en los *Controladores* específicos de cada módulo. Esta capa es la responsable de la persistencia de los datos, delegando en el módulo *Capa de datos* la gestión de la base de datos.
- **Capa de datos:** Contiene la lógica de acceso a datos y la persistencia de los mismos. Esta capa es la responsable de la gestión de la base de datos y de la persistencia de los datos. Hace uso de *DAO* (Data Access Object) para

la gestión de la base de datos. Está centralizada en *FachadaBaseDatos*, que es la responsable de la gestión de la base de datos.

3.2. Diagrama de capas/arquitectura

La Figura 1 muestra la arquitectura de capas del sistema, mostrando las principales fachadas que conectan cada capa y las relaciones entre ellas. Estas fachadas encapsulan la funcionalidad de cada capa, promoviendo el desacoplamiento y la claridad en el flujo de datos.

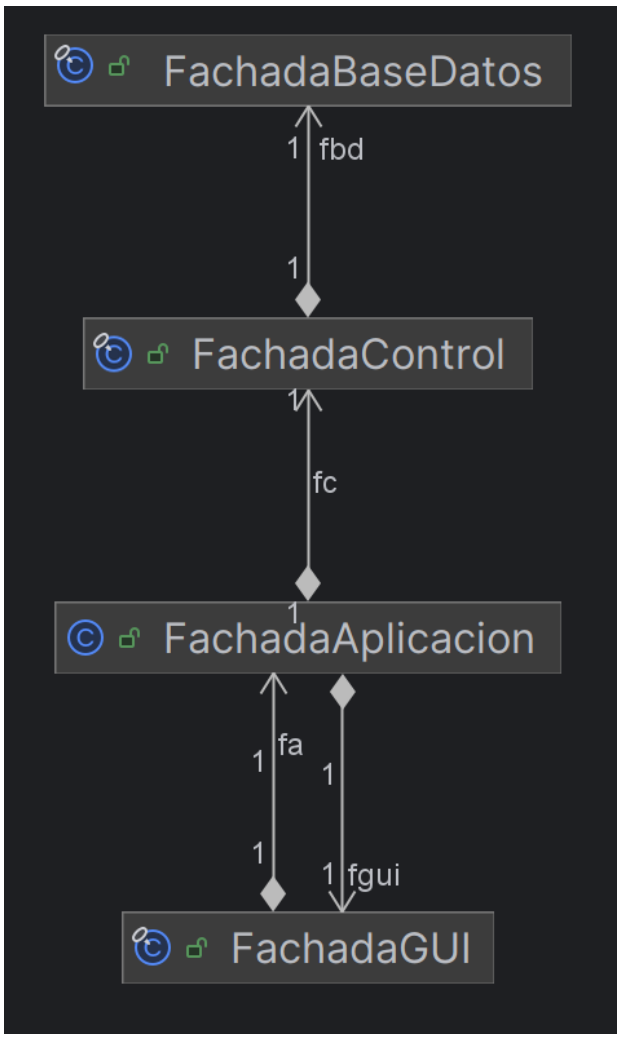

Figura 1: Diagrama de arquitectura por capas

3.3. Componentes identificados y sus responsabilidades

Después de hacer un análisis de los requisitos funcionales y no funcionales, hemos definido los siguientes componentes y sus responsabilidades:

3.3.1. Componentes identificados

- **Formularios de la GUI:** Son los responsables de la interacción con el usuario y de la presentación de la información.
- **Controladores (Servicios y lógica de negocio):** Gestionan la lógica de la aplicación e interceden entre los DAO y la interfaz gráfica.
- **DAO:** Se encargan de ≪hablar≫ con la base de datos. Permiten encapsular y abstraer la base de datos y a los datos mismos de la lógica de la aplicación.
- **Base de datos:** Gestiona los datos ≪permanentes≫.

3.3.2. Responsabilidades

Para cada componente, hemos identificado las siguientes responsabilidades funcionales:

GUI (Presentación)
- Mostrar la interfaz gráfica.
- Gestionar las diferentes vistas.

Controladores (Servicios)
- Gestionar usuarios.
- Autentificar.
- Cambiar datos.
- Gestionar alquileres.
- Gestionar la criptografía.

DAOs (Data Access Object)
- Hacer consultas a la base de datos.
- Modificar la base de datos.
- Eliminar entradas de la base de datos.

DAOs identificados: DAOUsuario, DAOEstacion, DAOBicicleta, DAOViaje, DAOFactura.

Base de datos (Persistencia)
- Guardar los datos de:
  - Usuarios
  - Bicicletas
  - Estaciones
  - Alquileres

3.4. Dependencias y flujos de control

El flujo habitual de control de la aplicación es el que se muestra en la Figura 2. En este flujo, la *GUI* envía una petición a la *FachadaAplicacion*, que

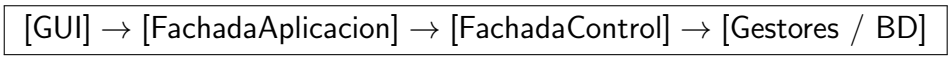

Figura 2: Flujo de control

delega en la *FachadaControl* para que gestione la petición y pueda solicitar a la *FachadaBaseDatos* el acceso/almacenamiento de datos en la base de datos.

Ciertamente, existen algunas dependencias cíclicas entre las clases de la capa de presentación, la fachada de presentación y la fachada de aplicación, ya que la *FachadaAplicacion* necesita conocer la *FachadaGUI* para poder enviarle mensajes. Puede parecer alarmante, pero no es un problema, ya que el ciclo ocurre principalmente dentro de la capa de presentación y no afecta a la lógica de negocio. Además, las llamadas desde la *FachadaAplicacion* a la *FachadaGUI* son para situaciones muy específicas, como cambiar el idioma o relanzar la interfaz.

3.5. Justificación de la arquitectura (Rationale)

La elección de la arquitectura por capas se ha tomado por su capacidad de desacoplar y diferencias las responsabilidades de las distintas capas. Gracias a esta arquitectura, este sistema gana mayor facilidad de mantenimiento, escalabilidad y modularidad.

Por otro lado, la elección del patrón *Façade* se ha tomado por su capacidad de simplificar la interfaz de las capas y facilitar la comunicación entre ellas. Este patrón permite ocultar la complejidad de las capas y proporcionar una interfaz más sencilla y fácil de usar. El otro patrón utilizado es el *DAO*, que centraliza y separa el acceso a datos en módulos específicos. Evitando malas prácticas como el acceso directo a la base de datos desde la capa de presentación o la capa de negocio.

4. Hardware/software mapping

5. Persistent data management

6. Access control and security

7. Global software control

8. Boundary conditions