INGENIERÍA DE SOFTWARE (3385)

PROYECTO 2B

2025

INTEGRANTES:

Adorno Gabriela Soledad

Cibils Mateo

Delfino Juan Manuel

Estanguet Juan Ignacio

Ludueña Augusto

DESARROLLO

El patrón que fue identificado en el sistema es el **Singleton**, patrón Creacional que es utilizado en el código del programa para encargarse de inicializar nuestro servidor **“DBConfigSingleton”** y de que este sea único para la base de datos que manejamos. Este patrón lo encontramos dentro del archivo DBConfigSingleton.java con la dirección : “is1_2025_eti\src\main\java\com\is1\proyecto\config “. Y fue aplicado de la siguiente forma dentro del código:

- Donde podemos observar la instancia privada y estática del objeto singleton, lo que nos asegura que solo habrá una instancia en el programa.

```
private static DBConfigSingleton instance;
```

- Al tener un constructor privado permite que solamente lo instancie su misma clase, por lo que las demás clases públicas no van a poder generar instancias de “**DBConfigSingleton**”.

```
// Constructor privado para evitar instanciación directa
private DBConfigSingleton() {
    // Configuraciones para SQLite
    this.driver = "org.sqlite.JDBC"; // Driver JDBC para SQLite
    this.dbUrl = System.getProperty(key: "db.url", def: "jdbc:sqlite:./db/dev.db");
    this.user = ""; // SQLite no usa usuario
    this.pass = ""; // SQLite no usa contraseña
}
```

- Por último tenemos el **getInstance()**, que es declarado de forma pública lo que nos asegura que haya un acceso global por parte de los objetos que vayan a utilizarlo, en nuestro caso **App.java** (nuestra clase principal) hace uso de esta propiedad: `DBConfigSingleton dbConfig = BConfigSingleton.getInstance();.`

```
public static synchronized DBConfigSingleton getInstance() {
    if (instance == null) {
        instance = new DBConfigSingleton();
    }
    return instance;
}
```

En el sistema se implementó la siguiente historia de usuario:

ID de HU	HU-001
Título	Alta de profesor al sistema
Declaración	Como administrador del sistema, quiero registrar un nuevo profesor ingresando su información personal, para poder asignarlo a las asignaturas correspondientes dentro de una carrera.
Descripción Detallada	
Criterios de Validación	● Flujo exitoso: Al completar todos los campos obligatorios (nombre, apellido, correo, DNI) con datos válidos y guardar, el El sistema muestra un mensaje de éxito.. ● Validaciones de Datos: El sistema debe impedir el registro si: ○ Faltan campos obligatorios. ○ Si el formato del correo electrónico no es válido. ○ El correo electrónico o el DNI ya existen en la base de datos. ● Manejo de Errores: Si alguna validación falla, el sistema debe mostrar un mensaje de error claro, sin permitir que se guarde el formulario. ● Acción de Cancelar: El formulario debe incluir un botón "Cancelar" que elimine todos los datos ingresados y devuelva al usuario a la pantalla anterior.
Tareas Asociadas a la Implementación	

Por medio de los siguientes archivos y modificaciones:

- Teacher.java que se encuentra dentro de **models**.
- Dentro de squeme.sql se agregó la tabla propia para cargar a los profesores.
- En Apps.java se encuentra el GET y POST correspondiente a la obtención del formulario para la carga de datos del profesor y el posterior envío del formulario a la base de datos.
- También fue creado el template teachers_formulario.mustach para el manejo del estilo y estructura del form HTML.