

Entrega #1 - Diseño Arquitectónico y Principios de Software

Grupo: The devs

Integrantes:

-Julio César Astudillo Bustamante: UX/UI

-Kevin Eduardo Valverde mullo: frontend

-Omar Enrique Téllez Sanabria: backend

-Cristopher Alexis Granja Astudillo: Pruebas y CI/CD

Introducción

En esta primera entrega hemos avanzado con todo lo requerido, tenemos una buena idea de lo que queremos hacer y como lo vamos a implementar.

Nuestro objetivo con esta entrega es mostrar el inicio de lo que será este gran proyecto, las bases, ideas, su estructura, etc.

Contenido Técnico

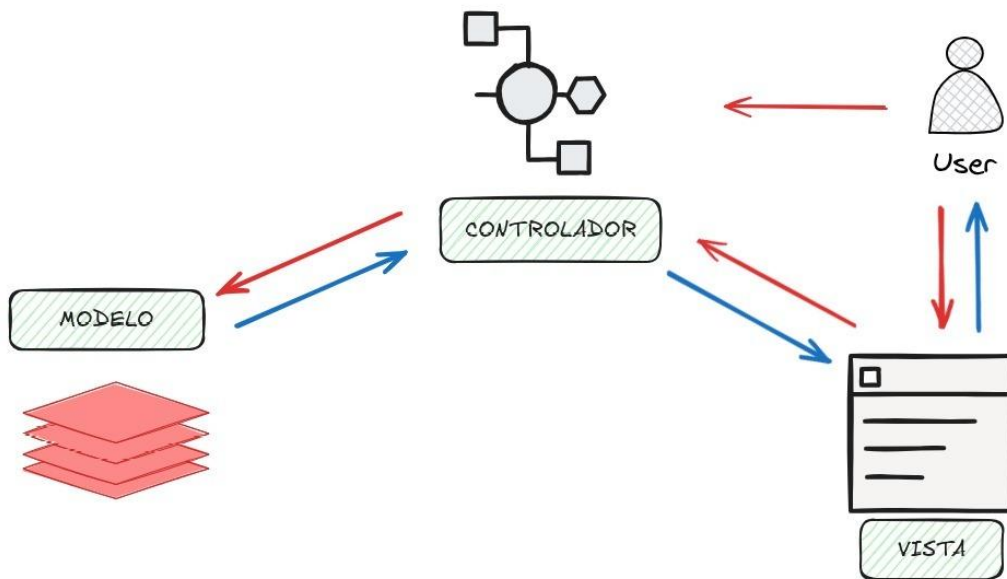
Historias de usuario:

1. Como Administrado, quiero crear, editar y eliminar usuarios para mantener actualizado el acceso del personal.
2. Como Administrador, quiero registrar y configurar tipos de usuario (roles) para controlar permisos y vistas.
3. Como usuario empleado, quiero registrar mi asistencia desde la página web para que quede guardada mi hora de entrada y salida.
4. Como Empleado, quiero ver mi historial de asistencias y reportes en PDF para revisar mis registros.
5. Como Administrador, quiero poder ver el historial de entradas y salidas de todos los usuarios para revisión administrativa.
6. Como administrador, quiero generar reportes en PDF de asistencia para tener respaldo físico o digital de los registros en caso de auditorías.

Arquitectura propuesta (Modelo-Vista-Controlador)

La arquitectura MVC (Modelo–Vista–Controlador) es un patrón que organiza el sistema de asistencia separando los datos, la lógica y la interfaz.

- **Modelo (M):** administra la base de datos de tu proyecto en a la cual contiene empleados, usuarios, horarios, entradas, salidas, reportes. Aquí se guardan y consultan todos los registros de asistencia.
- **Vista (V):** son las pantallas que los usuarios ven el formulario de login, botón de marcar entrada/salida, reportes de asistencia, panel de administrador.
- **Controlador (C):** maneja la lógica de recibe la acción de marcar entrada o salida, valida si es correcta (ej. usuario existe, no ha marcado antes), envía los datos al modelo y devuelve un resultado a la vista.



Aplicación de principios SOLID (Explicar al menos 3)

- **Responsabilidad Única (SRP):** los archivos de conexión (Conexion.php) solo manejan la conexión a la base de datos, aislando esa responsabilidad.
- **Inversión de Dependencia (DIP):** la lógica de negocio no depende directamente de la vista, sino que pasa por controladores (Ajax) que interactúan con los modelos.
- **Abierto/Cerrado (OCP):** el sistema permite extenderse (ej. generar nuevos reportes, nuevas vistas) sin modificar el núcleo, ya que cada módulo está separado.

Patrones de diseños usados

- **Singleton:** Se aplica en la conexión a la base de datos, garantizando que exista una sola instancia reutilizable en toda la aplicación.
- **DAO (Data Access Object) / Facade :** El acceso a la base de datos (consultas de usuarios, roles y asistencias) suele estar aislado en clases/métodos esto funciona como una fachada que simplifica el acceso al subsistema de datos.
- **Template Method :** En la generación de reportes PDF, se pueden definir pasos comunes (obtener datos, dar formato, exportar) con posibilidad de variar el formato o los filtros.

Consideraciones de seguridad y escalabilidad

Seguridad:

- Uso de inicio de sesión con credenciales.
- Separación de roles (Administrador vs Usuario).
- Archivos subidos se almacenan en carpeta files/ controlada.
- Configuración de conexión en archivo centralizado (Conexion.php).

Escalabilidad:

- Su arquitectura MVC facilita añadir nuevos componentes y módulos
- Uso de librerías externas como FPDF para extender funcionalidades.
- Posible integración con más bases de datos o servicios web.
- Estructura inicial del proyecto en el repositorio Requisitos de código

Diagramas UML

Diagrama de estado



Diagrama de actividades

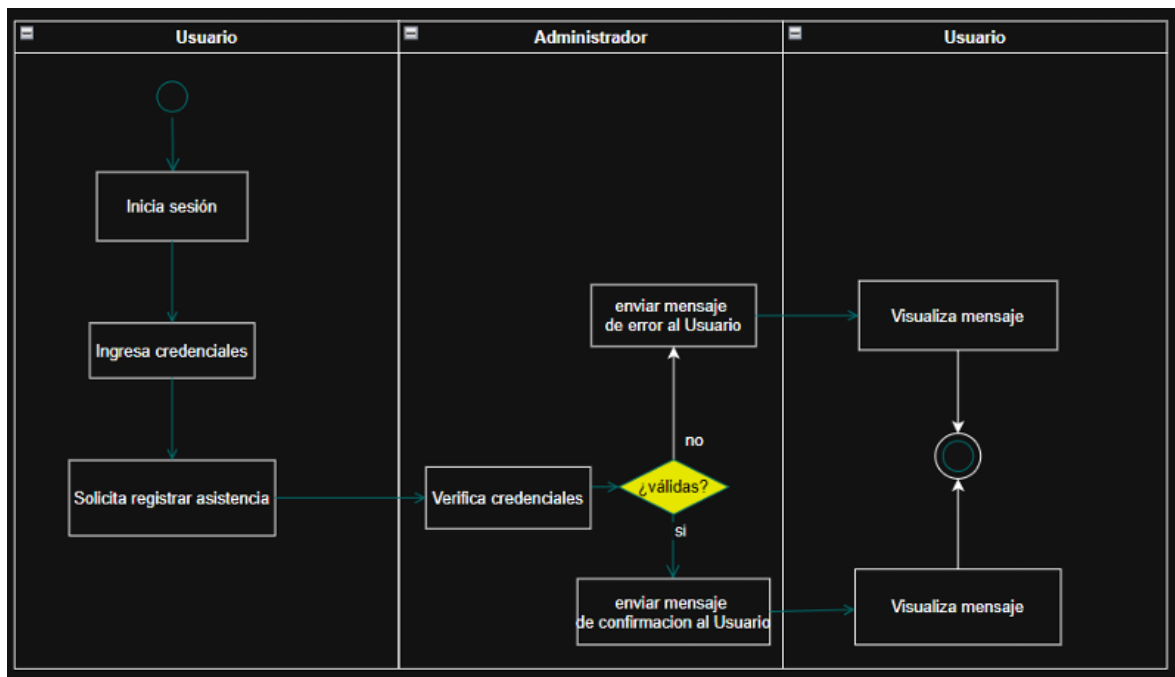


Diagrama caso de uso

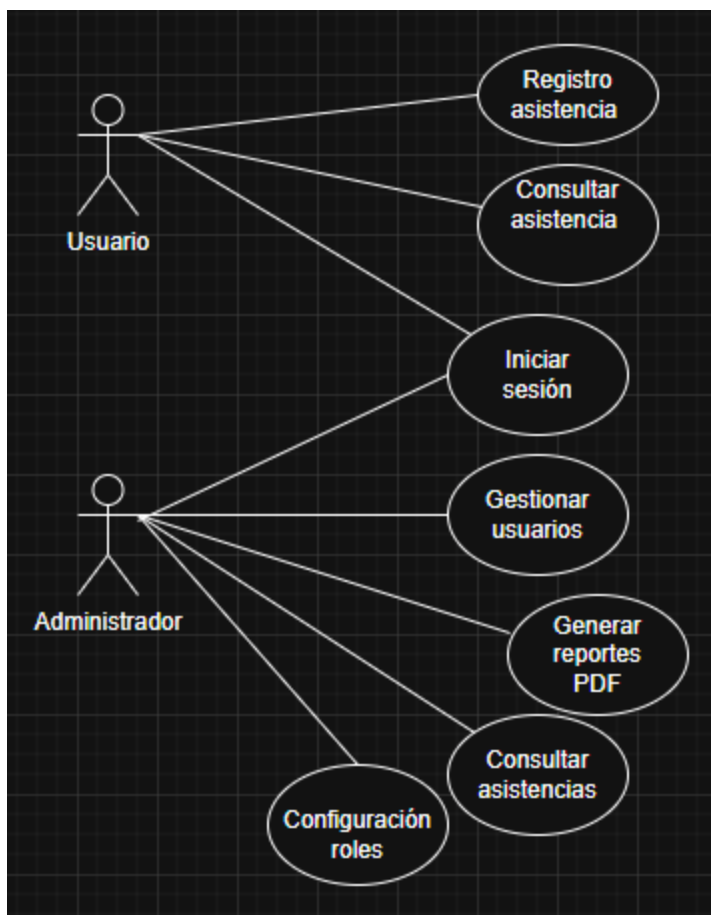


Diagrama de secuencia

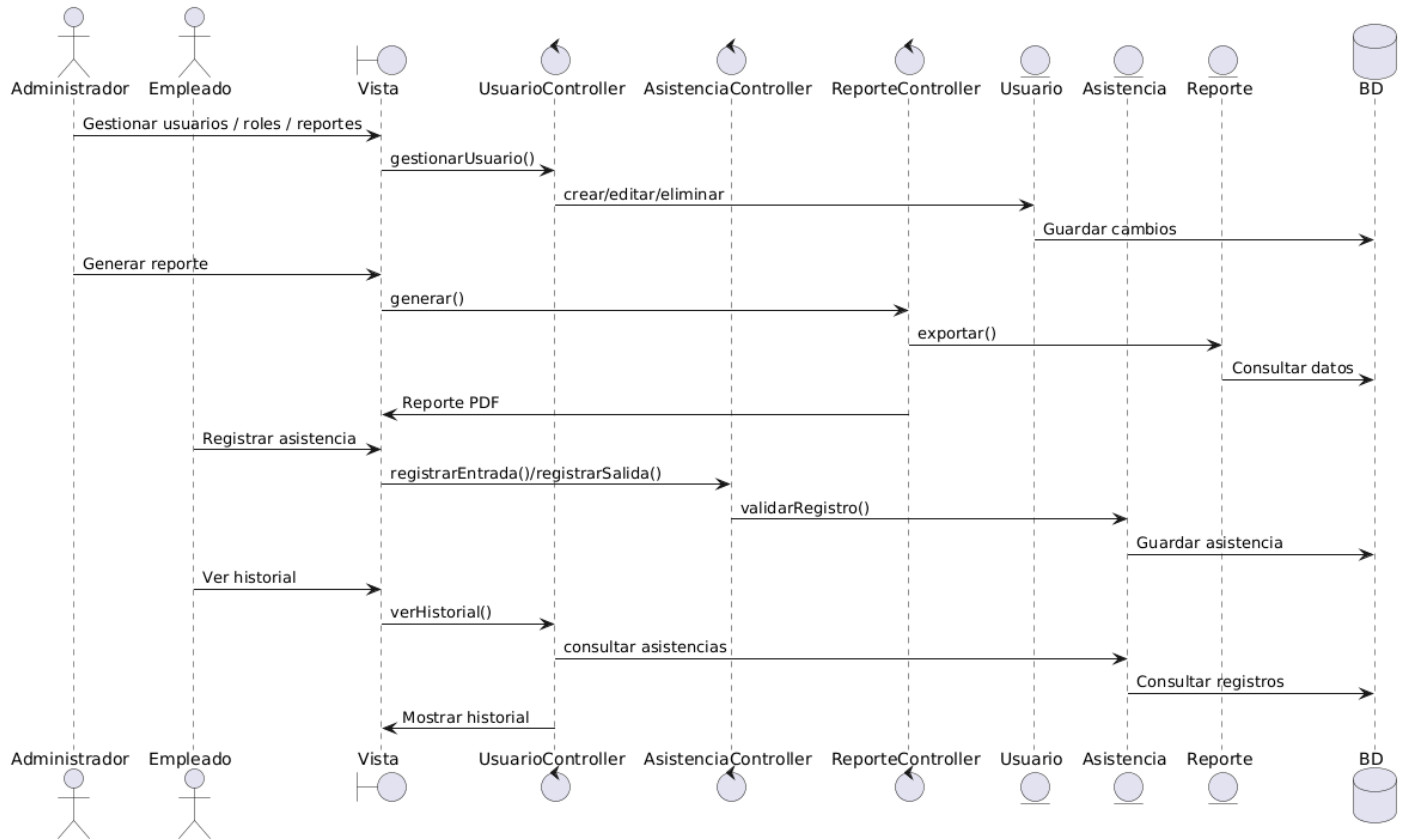
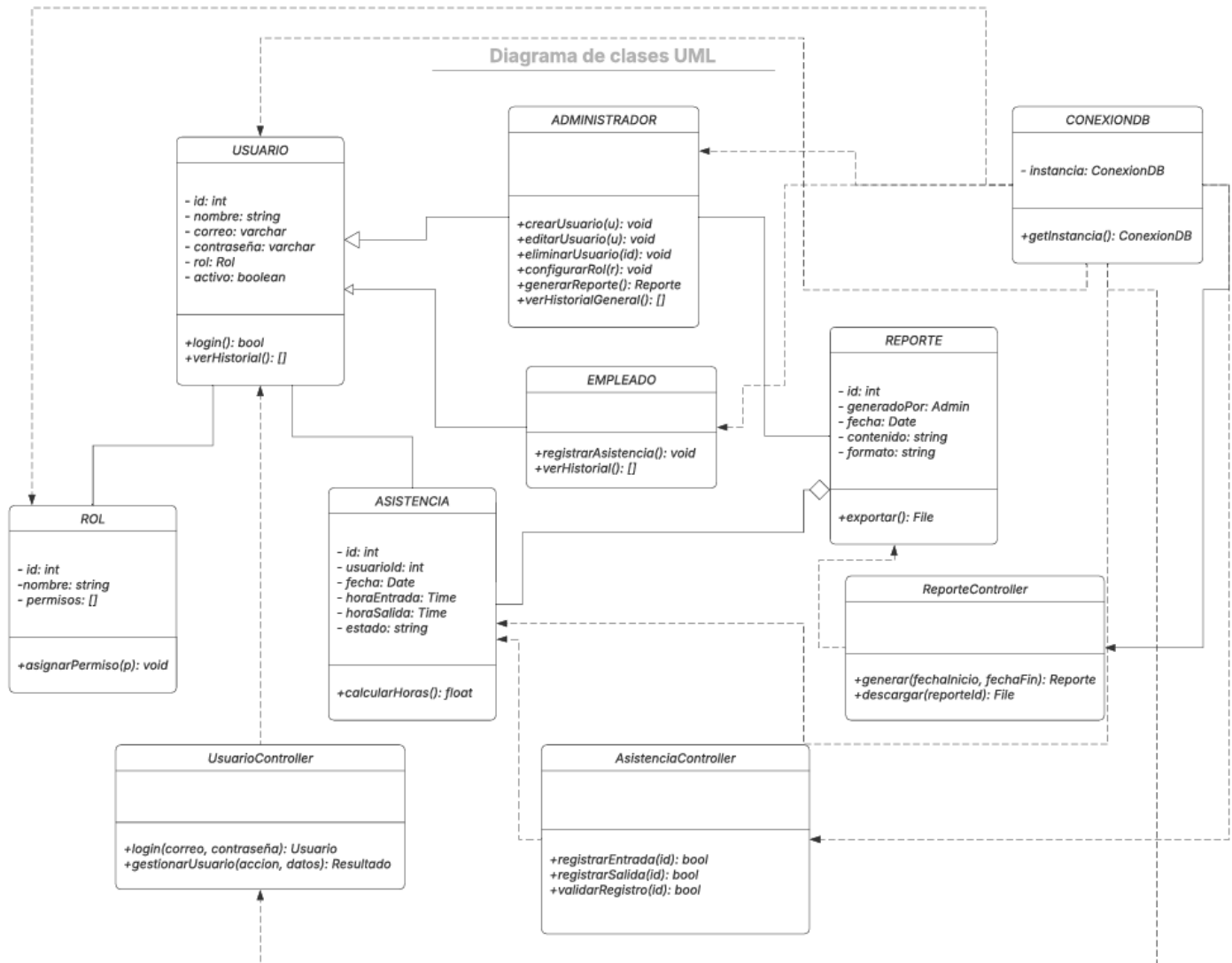


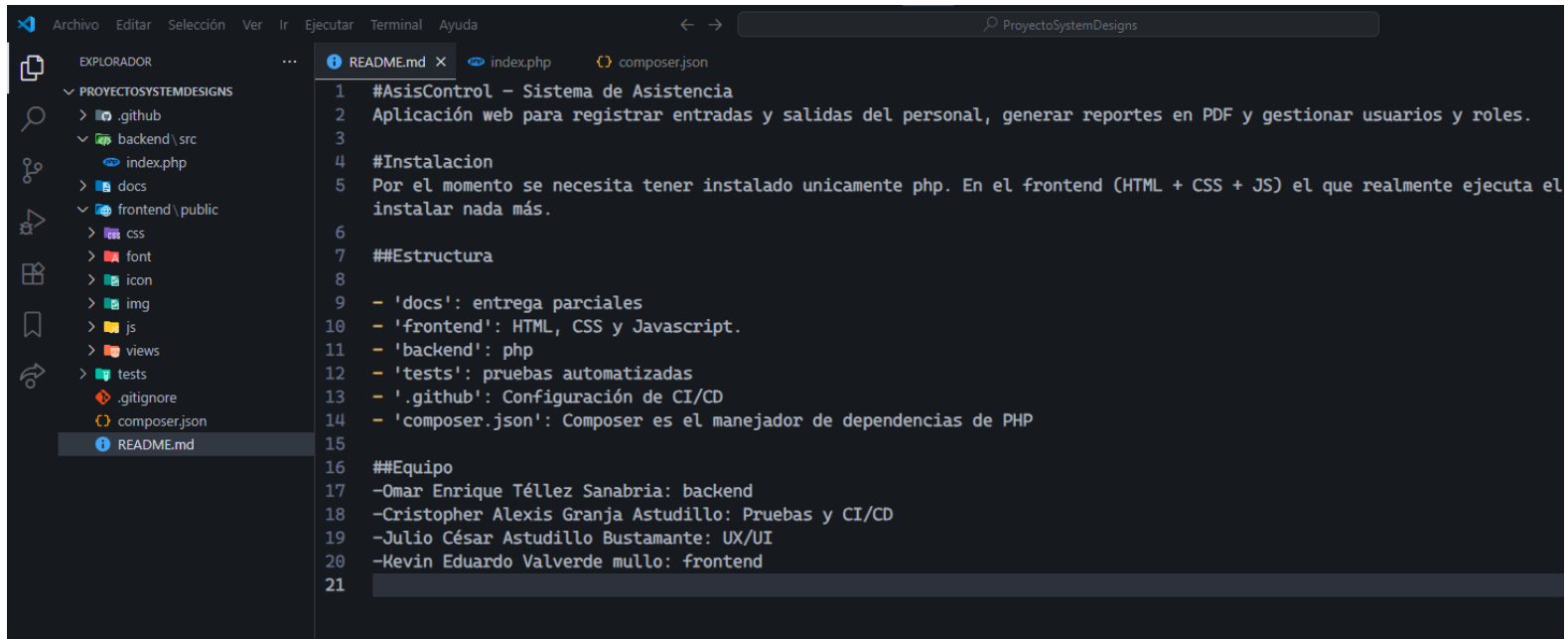
Diagrama de clases



Repositorio de GitHub:

<https://github.com/OmarTellez1/ProyectoSystemDesigns.git>

Editor de código



The image shows a code editor interface with a dark theme. On the left is a sidebar with a file explorer showing the project structure: 'PROYECTO SYSTEMDESIGNS' with subfolders like '.github', 'backend', 'docs', 'frontend', and 'public'. The main editor area displays the 'README.md' file. The content of the README is as follows:

```
1 #AsisControl - Sistema de Asistencia
2 Aplicación web para registrar entradas y salidas del personal, generar reportes en PDF y gestionar usuarios y roles.
3
4 #Instalacion
5 Por el momento se necesita tener instalado unicamente php. En el frontend (HTML + CSS + JS) el que realmente ejecuta el
6 instalar nada más.
7
8 ##Estructura
9 - 'docs': entrega parciales
10 - 'frontend': HTML, CSS y Javascript.
11 - 'backend': php
12 - 'tests': pruebas automatizadas
13 - '.github': Configuración de CI/CD
14 - 'composer.json': Composer es el manejador de dependencias de PHP
15
16 ##Equipo
17 -Omar Enrique Téllez Sanabria: backend
18 -Cristopher Alexis Granja Astudillo: Pruebas y CI/CD
19 -Julio César Astudillo Bustamante: UX/UI
20 -Kevin Eduardo Valverde mullo: frontend
21
```

Conclusiones

En esta primera entrega hemos aprendido lo importante que es planificar nuestro proyecto antes de programar. Tener las ideas, bases, diagramas, estructura del proyecto claras facilitara las cosas al momento de programar.