**Herramientas y Configuración del Entorno**

* **Enfoque de Desarrollo:**

Para este proyecto, se decidió implementar una metodología ágil que combina elementos de Scrum y Kanban. Esto permite gestionar tareas de manera visual y dinámica, a la vez que se promueve un trabajo interactivo y colaborativo. Esta aproximación asegura una capacidad de respuesta rápida ante cambios en los requisitos y facilita la entrega de versiones funcionales en ciclos cortos.

Además, el diseño del sistema sigue el modelo 4+1 de vistas arquitectónicas, que permite representar la arquitectura desde diferentes perspectivas complementarias: lógica, desarrollo, procesos, despliegue y casos de uso.

* **Tecnologías y Herramientas Utilizadas:**

A continuación, se detalla el conjunto de herramientas tecnológicas que se utilizaron durante el desarrollo del sistema:

| **Categoría** | **Herramienta / Tecnología** |
| --- | --- |
| Desarrollo Frontend | TypeScript |
| Diseño de Interfaz | Tailwind CSS |
| Desarrollo Backend | Node.js con Express |
| Gestión de Datos | MySQL |
| Control de Código | Git integrado con GitHub |
| Organización del Trabajo | Jira utilizando tableros Kanban |
| Editor de Código | Visual Studio Code |
| Automatización de Procesos | GitHub Actions |
| Implementación y Hosting | Railway |

En el contexto del modelo 4+1, herramientas como Draw.io fueron fundamentales para la creación de diagramas que representan las vistas arquitectónicas. Estas visitas permitieron al equipo de desarrollo comunicar claramente las decisiones de diseño a los distintos interesados.

**Modelo de Arquitectura:**

El sistema ha sido diseñado siguiendo el modelo Cliente-Servidor, donde las operaciones principales se dividen entre los clientes, que manejan la interfaz de usuario, y el servidor, que se encarga del procesamiento de solicitudes y la gestión de datos. Este enfoque garantiza una clara separación de responsabilidades y facilita la integración de múltiples dispositivos.

En complemento, el modelo 4+1 proporciona una estructura modular para la arquitectura, abarcando:

* Vista lógica: Define las relaciones entre entidades clave del sistema.(Modelo entidad - relación)
* Vista de implementación: Describe la distribución de los componentes de software sobre la infraestructura de hardware, considerando su despliegue real y la forma en que interactúan físicamente.(Diagrama de componentes)
* Vista de procesos: Describe cómo se ejecutan y sincronizan los flujos de actividades principales.(Diagramas de actividades)
* Vista de escenarios: describen secuencias de interacciones entre objetos, y entre procesos. Se utilizan para identificar y validar el diseño de arquitectura.(Casos de uso)

**Ventajas del Modelo:**

1. Flexibilidad: Posibilita incorporar nuevas funcionalidades o clientes sin afectar significativamente la infraestructura del servidor.
2. Eficiencia: Optimiza la interacción entre usuarios y el backend, asegurando tiempos de respuesta adecuados.
3. Modularidad: Divide las funciones del sistema en componentes fácilmente actualizables y escalables.
4. Clara Documentación: A través del modelo 4+1, se crean diagramas claros y detallados que facilitan la comprensión y mantenimiento del sistema.

**Componentes Clave de la Arquitectura:**

* API Gateway: Administra y redirige las solicitudes entre los clientes y los servicios del backend.
* Microservicios Especializados: Módulos independientes para manejar usuarios, cursos, pagos y reportes.
* Bases de Datos Dedicadas: Cada módulo gestiona su propia base de datos para mejorar el rendimiento y la seguridad.
* Conexiones Externas: Integración con servicios de terceros para procesamiento de pagos, asegurando transacciones confiables.

Con esta arquitectura y el soporte del modelo 4+1, el sistema está preparado para operar de manera eficiente y escalable, manteniendo la calidad en la experiencia del usuario.

**Patrón de diseño de software / Arquitectura de diseño**

El modelo arquitectónico principal es el de **Microservicios:**

* El proyecto está dividido en varios módulos independientes: api\_user, api\_course, api\_payment, api\_report, y gateway. Cada uno tiene su propio pom.xml, estructura de carpetas, controladores, servicios y entidades.
* Cada módulo es una aplicación Spring Boot independiente, con su propia clase principal (@SpringBootApplication), lo que permite ejecutarlos como servicios separados.
* Los servicios se comunican entre sí mediante HTTP (por ejemplo, el uso de RestTemplate y WebClient para consumir APIs de otros módulos).
* No existe un único punto de entrada ni una sola base de datos compartida, sino que cada microservicio puede gestionar su propio dominio y persistencia.
* Utilización de un API Gateway.

### Otros patrones presentes:

* **Patrón por capas:** Dentro de cada microservicio, el código está organizado en capas (controladores, servicios, repositorios, entidades).
* **MVC:** Los controladores gestionan las peticiones, los servicios la lógica de negocio y las entidades representan los datos, siguiendo el patrón Modelo-Vista-Controlador.

### Modelos descartados:

* Monolítico: No es monolítico porque el sistema está claramente dividido en servicios independientes.
* Cliente-Servidor: Aunque hay comunicación HTTP, la arquitectura va más allá de un simple cliente-servidor.
* Maestro-Esclavo: No hay un componente maestro que controle esclavos.