# Informe Final

En este proyecto se ha trabajado en el desarrollo de una plataforma educativa que busca ser segura eficiente y escalable para permitir la gestión de cursos estudiantes y profesores de manera óptima. La idea principal ha sido crear un sistema que pueda crecer con el tiempo sin que eso implique rehacer gran parte del código ni comprometer su seguridad. Para ello se ha elegido la Programación Orientada a Objetos como el paradigma base ya que permite organizar el código de forma estructurada modular y reutilizable lo que facilita tanto el mantenimiento como la futura ampliación del proyecto.

Uno de los aspectos fundamentales en los que se ha trabajado es la seguridad ya que en una plataforma educativa los datos personales de los estudiantes y profesores deben estar bien protegidos contra posibles ataques o accesos no autorizados. Para ello desde el primer momento se han implementado medidas de seguridad robustas como la gestión adecuada de permisos el cifrado de datos y la autenticación segura con el objetivo de minimizar cualquier riesgo de vulnerabilidad.

Además la plataforma ha sido diseñada teniendo en cuenta la escalabilidad lo que significa que a medida que aumente el número de usuarios o cursos el sistema podrá manejar la carga sin perder rendimiento. Para conseguir esto se han utilizado tecnologías modernas y buenas prácticas en el desarrollo de software como la separación de responsabilidades dentro del código el uso de patrones de diseño eficientes y la optimización de consultas a la base de datos.

Para facilitar el trabajo en equipo y agilizar el desarrollo se han utilizado herramientas como GitHub Codespaces que permite programar en la nube sin necesidad de configurar entornos locales complejos y Docker que ha sido clave para contenerizar la aplicación asegurando que funcione igual en cualquier servidor sin problemas de compatibilidad. También se ha integrado un pipeline de integración y entrega continua que automatiza el proceso de pruebas y despliegue reduciendo errores y asegurando que la plataforma esté siempre en su mejor estado antes de ser usada por los usuarios.

En este informe se detallará todo el proceso de desarrollo desde la elección del lenguaje de programación hasta la estrategia de optimización de entornos pasando por la arquitectura del software la implementación de medidas de seguridad y el uso de herramientas que han permitido mejorar la eficiencia del trabajo. Todo ello con el objetivo de garantizar que la plataforma sea fiable segura y fácil de usar para estudiantes profesores y administradores.

**Elección del Lenguaje de Programación**

Para el desarrollo de esta plataforma se ha optado por utilizar Java como lenguaje de programación principal debido a sus múltiples ventajas en términos de seguridad estabilidad y compatibilidad con diferentes sistemas operativos. Java es un lenguaje ampliamente utilizado en entornos empresariales y educativos gracias a su robustez y a la posibilidad de ejecutar aplicaciones en cualquier dispositivo que tenga instalada la Máquina Virtual de Java lo que lo convierte en una excelente opción para un sistema que debe ser accesible desde distintos equipos sin necesidad de realizar adaptaciones específicas.

Una de las razones más importantes para elegir Java es su enfoque en la seguridad. Java cuenta con mecanismos integrados como el control de acceso la gestión automática de memoria y el manejo de excepciones lo que ayuda a prevenir errores comunes que podrían generar vulnerabilidades en el sistema. Además el uso de bibliotecas de seguridad avanzadas y la compatibilidad con protocolos de autenticación modernos permiten reforzar la protección de los datos dentro de la plataforma educativa.

Otro punto a favor de Java es su estructura basada en Programación Orientada a Objetos que facilita la organización del código en clases y objetos lo que hace que sea más fácil mantener y ampliar la plataforma en el futuro. Este enfoque permite definir claramente las relaciones entre los diferentes componentes del sistema como los usuarios los cursos y la gestión de notas haciendo que el código sea más legible y escalable.

Si bien Java tiene algunas desventajas como un consumo de memoria mayor en comparación con otros lenguajes y una velocidad de ejecución que puede ser algo más lenta en ciertos casos estos inconvenientes se han mitigado con estrategias de optimización adecuadas. Se han aplicado técnicas como la reducción de instancias innecesarias la optimización de consultas a la base de datos y el uso de herramientas de profiling para identificar cuellos de botella y mejorar el rendimiento general de la plataforma.

Además la amplia comunidad de desarrolladores que utiliza Java y la gran cantidad de documentación disponible han sido factores clave para decantarse por este lenguaje. Al contar con una gran cantidad de recursos y soluciones ya existentes se ha podido acelerar el desarrollo del proyecto resolviendo problemas de manera eficiente y asegurando que se sigan las mejores prácticas en todo momento.

En definitiva Java ha sido la elección ideal para este proyecto porque ofrece un equilibrio entre seguridad escalabilidad y facilidad de mantenimiento lo que garantiza que la plataforma pueda evolucionar y mejorar con el tiempo sin comprometer su rendimiento ni su estabilidad.

La arquitectura del software de la plataforma ha sido diseñada para que sea modular flexible y fácil de mantener. Desde el principio se ha apostado por una estructura basada en clases bien definidas que permitan organizar el código de manera lógica y clara. El corazón del sistema está compuesto por una clase principal llamada Plataforma que gestiona la interacción entre los diferentes usuarios los cursos y las evaluaciones.

En la base del diseño se encuentra una clase abstracta llamada Usuario que define los atributos y métodos básicos que comparten todos los usuarios del sistema. A partir de esta clase se han creado tres subclases específicas para cada rol dentro de la plataforma Estudiante Profesor y Administrador. Gracias a esta estructura orientada a objetos es posible añadir nuevos roles en el futuro sin necesidad de modificar demasiado el código existente lo que aporta una gran flexibilidad y escalabilidad al sistema.

La clase Plataforma se encarga de centralizar todas las operaciones importantes de la aplicación. Esta clase gestiona la creación de cursos la matriculación de estudiantes la asignación de profesores y la administración de notas. Además, incorpora métodos para consultar información de los cursos acceder al historial académico de los estudiantes y permitir que los profesores realicen evaluaciones de manera sencilla.

Para asegurar la separación de responsabilidades cada funcionalidad clave de la plataforma se encuentra encapsulada en clases específicas por ejemplo la gestión de los cursos está a cargo de la clase Curso mientras que la administración de notas y calificaciones se maneja mediante la clase Evaluación. Esta organización permite que cada componente del sistema se pueda modificar y mejorar sin afectar al resto del código reduciendo así el riesgo de errores y facilitando el mantenimiento del software a largo plazo.

Además, la arquitectura ha sido diseñada teniendo en cuenta la integración con bases de datos para almacenar de manera segura la información de los usuarios los cursos y las evaluaciones. Se ha optado por utilizar una base de datos relacional debido a su fiabilidad y capacidad para manejar grandes volúmenes de datos estructurados. El acceso a la base de datos se realiza mediante una capa de persistencia que encapsula todas las operaciones de lectura y escritura evitando que el resto del sistema tenga que interactuar directamente con la base de datos y mejorando así la seguridad y el modularidad.

Otra característica clave del diseño arquitectónico es la implementación de un sistema de autenticación y autorización basado en tokens que garantiza que cada usuario solo pueda acceder a las funciones y datos para los que tiene permisos. Esto ayuda a proteger la plataforma contra accesos no autorizados y refuerza la seguridad del sistema en general.

Para garantizar un desarrollo eficiente y seguro se han utilizado diversas herramientas que han facilitado la implementación prueba y despliegue de la plataforma. Desde el inicio del proyecto se ha trabajado con Git y GitHub como herramientas principales de control de versiones permitiendo gestionar los cambios en el código de manera organizada y colaborativa. GitHub ha sido clave para mantener un flujo de trabajo estructurado mediante la creación de ramas pull requests y revisiones de código antes de integrar nuevas funcionalidades en la rama principal.

En cuanto al entorno de desarrollo se ha optado por usar GitHub Codespaces que permite programar en la nube sin necesidad de instalar y configurar herramientas localmente. Esto ha hecho que el desarrollo sea más ágil y que todos los miembros del equipo trabajen en un entorno homogéneo sin preocuparse por problemas de compatibilidad entre sus dispositivos. Además, se ha utilizado IntelliJ IDEA como entorno de desarrollo integrado local debido a su compatibilidad con Java y sus herramientas avanzadas para depuración y análisis de código.

Para la contenerización y despliegue se ha recurrido a Docker una herramienta que permite empaquetar la aplicación junto con todas sus dependencias asegurando que funcione de la misma manera en cualquier entorno de ejecución. Esto ha sido especialmente útil para evitar problemas de configuración y garantizar que la plataforma pueda ser desplegada sin inconvenientes en distintos servidores.

Otro aspecto fundamental ha sido la automatización del proceso de desarrollo y despliegue mediante un pipeline de integración y entrega continua CI/CD. Se han configurado workflows en GitHub Actions que permiten ejecutar automáticamente pruebas compilar el código y desplegar la aplicación en entornos de prueba y producción sin intervención manual. Gracias a este enfoque cualquier cambio en el código se valida de inmediato reduciendo errores y asegurando que la plataforma esté siempre en su mejor estado antes de ser utilizada por los usuarios.

En cuanto a la base de datos se ha optado por PostgreSQL un sistema de gestión de bases de datos relacional que ofrece un alto rendimiento y una gran capacidad de escalabilidad. Se han diseñado esquemas optimizados para manejar de manera eficiente la información de los cursos usuarios y calificaciones asegurando que las consultas sean rápidas y que la integridad de los datos esté garantizada.

Para la monitorización del sistema y detección de posibles fallos se han implementado herramientas como Prometheus y Grafana que permiten visualizar métricas clave sobre el rendimiento del servidor la carga del sistema y el tiempo de respuesta de las consultas. Esto facilita la identificación de cuellos de botella y permite optimizar el uso de los recursos para mejorar la experiencia de los usuarios.

El pipeline de integración y entrega continua CI/CD se ha diseñado para que cada fase del desarrollo se haga de manera automática asegurando que cualquier cambio en el código sea probado y validado antes de que llegue a producción. Esto hace que se eviten errores inesperados mejora la calidad del software y permite que el equipo trabaje mejor sin necesidad de hacer despliegues manuales que podrían provocar fallos o retrasos en el desarrollo.

El pipeline está dividido en siete fases que tienen cada una su función en el proceso de desarrollo y entrega del software. La primera fase es la activación y se inicia sola cuando un desarrollador sube cambios al repositorio principal o crea una solicitud de fusión pull request. Esto hace que cualquier modificación en el código se evalúe inmediatamente evitando que se integren cambios defectuosos en la rama principal.

La segunda fase es la compilación y contenerización donde el código fuente se compila para ver si tiene errores de sintaxis o problemas con las dependencias. Después el software se mete en una imagen de Docker asegurando que funcione igual en cualquier entorno sin importar el servidor donde se ejecute.

En la tercera fase se hacen pruebas automáticas para asegurarse de que el código cumple con lo que se espera y que no causa errores en el sistema. Se hacen pruebas unitarias para comprobar que cada parte del software funciona bien y pruebas de integración para ver que todas las partes se comunican correctamente. También hay pruebas de regresión para asegurarse de que los nuevos cambios no rompen funciones que ya existían.

Después de pasar las pruebas el software entra en la cuarta fase que es el despliegue en staging. Aquí se instala en un entorno de pruebas que imita al de producción para ver cómo se comporta antes de lanzarlo a los usuarios reales. Esto sirve para detectar fallos antes de que lleguen a producción.

Antes de hacer el lanzamiento definitivo el equipo de desarrollo revisa todo en la quinta fase que es la aprobación manual. En este punto se miran los resultados de las pruebas y se comprueba que todo funcione sin problemas. Solo cuando todo está bien se puede seguir adelante.

La sexta fase es el despliegue en producción donde el software se lanza para que lo usen los usuarios reales. Se hace de forma controlada para reducir riesgos utilizando estrategias como el despliegue azul-verde o el despliegue progresivo que permite liberar la nueva versión primero a un grupo pequeño de usuarios antes de hacerla pública para todos.

Por último la séptima fase es la monitorización y retroalimentación. Se usan herramientas para vigilar el rendimiento del software en producción. Se configuran alertas para avisar si algo va mal y se analizan datos clave para ver que todo esté funcionando bien y estable.

Planificar este pipeline permite que cualquier cambio o mejora en el software se haga de forma rápida y segura sin que el servicio se interrumpa y asegurando que los usuarios tengan siempre una buena experiencia. Gracias a la automatización de todas estas fases se reducen los errores humanos y el equipo puede centrarse en mejorar el producto sin perder tiempo en tareas repetitivas.

El desarrollo de un protocolo de seguridad en el software es muy importante para proteger los datos de los usuarios y evitar vulnerabilidades. Desde el principio del proyecto se han seguido buenas prácticas de seguridad para reducir riesgos y asegurar que el sistema sea resistente frente a ataques.

Para detectar y evitar amenazas se han usado metodologías como STRIDE y DREAD que sirven para analizar los puntos débiles del sistema y medir el impacto que podrían tener distintos ataques. STRIDE clasifica amenazas en seis tipos como suplantación de identidad manipulación de datos y denegación de servicio mientras que DREAD calcula el nivel de riesgo según el daño que causaría y lo probable que es que ocurra.

Un principio fundamental que se ha aplicado es el de privilegios mínimos lo que significa que cada usuario solo tiene los permisos que necesita para hacer su trabajo. Esto reduce el peligro de accesos no autorizados y evita que un atacante pueda hacer demasiado daño si consigue entrar en una cuenta.

También se ha añadido un sistema de autenticación fuerte que pide más de un factor para verificar la identidad del usuario. Se usa autenticación en dos pasos que combina una contraseña con un código de verificación que se envía por teléfono o correo electrónico haciendo más difícil que alguien acceda sin permiso.

Para proteger los datos que viajan por la red se ha implementado el protocolo TLS 1.3 que cifra toda la comunicación entre el cliente y el servidor para que nadie pueda interceptar o modificar la información. Además los datos guardados en la base de datos se cifran con AES-256 para que incluso si un atacante accede a los servidores no pueda leer la información sin la clave correcta.

Otra medida importante ha sido usar herramientas de análisis de seguridad en el desarrollo del software. Se han hecho pruebas con herramientas como SAST análisis estático del código que encuentra vulnerabilidades en el código fuente y DAST análisis dinámico de seguridad que prueba la aplicación mientras está en ejecución para detectar fallos.

Para manejar credenciales de forma segura se ha usado HashiCorp Vault que permite almacenar claves API contraseñas y otros secretos sin que se expongan en el código.

Además se han establecido reglas para mantener el software siempre actualizado y protegido. Se supervisan las dependencias del proyecto con Dependabot que avisa si hay vulnerabilidades en alguna biblioteca usada en el código para actualizarlas rápido.

El protocolo de seguridad también incluye protecciones contra ataques comunes como la inyección SQL y el cross-site scripting XSS. Para evitarlos se han puesto controles en los formularios de entrada de datos y se ha configurado una política de seguridad que bloquea scripts maliciosos en el navegador.

Otro aspecto importante es la defensa contra ataques de denegación de servicio DDoS que pueden colapsar la plataforma enviando miles de solicitudes falsas. Para evitar esto se han implementado medidas como firewalls y limitación de accesos para restringir la cantidad de peticiones que un usuario puede hacer en poco tiempo.

La estrategia de gestión y optimización de entornos es fundamental para asegurar que el software funcione de manera eficiente en todas las etapas del desarrollo. Se han diseñado diferentes entornos de trabajo para que cada fase del proyecto tenga su espacio propio y así evitar problemas que podrían surgir si todo se hiciera en un mismo entorno.

El primer entorno es el de desarrollo que es donde los programadores escriben y prueban el código antes de enviarlo al repositorio. En este espacio cada desarrollador tiene su propia copia del proyecto en su ordenador lo que permite hacer cambios sin afectar a los demás. Además se han configurado entornos virtuales con Docker para que todos los desarrolladores trabajen en un ambiente igual al de producción evitando problemas de compatibilidad.

El siguiente entorno es el de pruebas o staging donde se evalúan los cambios antes de que lleguen a producción. Aquí se replican las condiciones reales del sistema para asegurarse de que las nuevas funciones trabajan bien sin causar fallos. Se hacen pruebas automatizadas y manuales para detectar errores y mejorar el rendimiento.

El último entorno es el de producción que es donde los usuarios finales acceden a la plataforma. En esta fase se han aplicado medidas para que el sistema sea estable y pueda aguantar un alto número de conexiones sin problemas. Se han configurado servidores con balanceo de carga que reparten las peticiones entre varias máquinas para evitar sobrecargas y mejorar la velocidad del servicio.

Para optimizar el rendimiento de los entornos se han implementado varias estrategias. Una de ellas es el escalado automático que ajusta los recursos del sistema según la demanda. Si hay muchos usuarios conectados a la vez se añaden más servidores y cuando baja la actividad se reducen para ahorrar costes.

También se ha optimizado el uso de bases de datos mediante técnicas como la indexación y la caché. La indexación permite buscar información más rápido reduciendo el tiempo de respuesta y la caché almacena temporalmente datos usados con frecuencia para evitar hacer consultas repetitivas que puedan ralentizar el sistema.

Otra estrategia clave ha sido la monitorización continua de los entornos con herramientas que analizan el estado de los servidores y detectan cualquier problema antes de que afecte a los usuarios. Se han establecido alertas para notificar si hay caídas de servicio o si el consumo de recursos es demasiado alto.

El mantenimiento preventivo es otro aspecto importante en la gestión de entornos. Se han programado actualizaciones periódicas del software y revisiones de seguridad para corregir vulnerabilidades antes de que puedan ser explotadas. También se hacen copias de seguridad automáticas para asegurar que la información no se pierda en caso de fallo del sistema.

Además, se ha optimizado la distribución de contenido mediante una red CDN que almacena copias de la plataforma en distintos lugares del mundo para que los usuarios accedan más rápido al servicio sin importar su ubicación. Esto reduce la latencia y mejora la experiencia de navegación.

Para garantizar una buena gestión de los entornos se han establecido reglas claras sobre cómo y cuándo se deben hacer cambios en el sistema. Se ha definido un flujo de trabajo donde cualquier modificación pasa por un proceso de revisión antes de implementarse asegurando que no cause problemas en producción.





