Proyecto final: Documentación

Sebastian Andrade Cedano
Cristian Steven Motta Ojeda
Juan Esteban Santacruz Corredor

Ingeniería de software I

Universidad Nacional de Colombia

Bogotá D.C.
2025

Índice

1. Levantamiento de requerimientos

¿De dónde surge la idea?

El proyecto Codes nace de la experiencia compartida de todos los integrantes del equipo en la programación competitiva. Durante su práctica, identificaron la necesidad de contar con una plataforma integral que no solo ofrezca problemas de distintos niveles de dificultad, sino también explicaciones detalladas, estadísticas de rendimiento y herramientas interactivas para mejorar el aprendizaje. Aunque existen plataformas similares, muchas carecen de un enfoque unificado que combine estos elementos de manera accesible y estructurada.

¿Cómo se pusieron de acuerdo para elegir la idea?

La elección de la idea fue un proceso sencillo. Aunque se consideró la opción de desarrollar una plataforma para la búsqueda de empleo dentro de la universidad, la afinidad de todos los miembros con la programación competitiva inclinó rápidamente la balanza hacia Codes. El equipo llegó a un consenso sin dificultad, ya que todos compartían el interés por mejorar la experiencia de entrenamiento en este campo y veían en este proyecto una oportunidad para aplicar sus conocimientos de manera práctica.

¿Cuáles son las problemáticas que buscan resolver?

El público objetivo de Codes son estudiantes y entusiastas de la programación competitiva que buscan mejorar sus habilidades de resolución de problemas. Actualmente, muchas plataformas carecen de un sistema integral que combine la práctica con explicaciones detalladas, estadísticas de progreso y herramientas sociales para fomentar la interacción entre usuarios. Codes busca cubrir este vacío, ofreciendo una experiencia más completa y efectiva para el aprendizaje y la mejora en competencias algorítmicas.

¿Qué expectativas tienen los usuarios potenciales del sistema?

Los usuarios esperan encontrar un entorno que les permita practicar programación competitiva de manera eficiente. Entre sus principales expectativas se incluyen:

- Acceder a problemas.
- Evaluar automáticamente sus soluciones y recibir retroalimentación.
- Visualizar estadísticas detalladas sobre su desempeño.
- Interactuar con otros usuarios agregándolos como amigos y comparando su progreso.
- Acceder a explicaciones detalladas que faciliten el aprendizaje de conceptos clave.

¿Qué beneficios esperan ustedes al desarrollar este proyecto?

El desarrollo de Codes representa una oportunidad para que los integrantes del equipo apliquen los conocimientos adquiridos en cursos previos, como bases de datos, estructuras de datos, entre otros. Además, les permitirá mejorar sus habilidades de trabajo en equipo, algo fundamental en cualquier entorno profesional. Finalmente, el proyecto servirá como una experiencia práctica para interiorizar los conceptos de Ingeniería de Software I, fortaleciendo su formación en el desarrollo de software de manera estructurada y colaborativa.

Requerimientos funcionales:

1. Autenticación y Roles de Usuario
    a. Registro de usuarios.
    b. Inicio de sesión seguro con roles definidos (Administrador, Usuario o Problemsetter) mediante usuario y contraseña.

2. Problemset
    a. Lista de problemas filtrable por estado (resuelto/no resuelto).
    b. Función de búsqueda de problemas por título o descripción.

3. Checker
    a. Envío de soluciones y evaluador automático que compara la salida del código con los casos de prueba.

4. Problemsetter
    a. CRUD de problemas.

5. Amigos
    a. Lista de amigos.
    b. Gestión de solicitudes de amistad (Enviar, aceptar, rechazar).

6. Custom test
    a. Ejecución de código con entradas personalizadas en un entorno aislado.

7. Estadísticas del usuario
    a. Visualización de problemas resueltos, tasa de éxito e historial gráfico del progreso.

8. Editorial
    a. Sección vinculada a cada problema con explicaciones y ejemplos detallados.
    b. Explicaciones paso a paso.

9. Historial de envíos
    a. Tabla con detalles de los envíos: fecha, problema, estado (Aceptado, Wrong Answer, etc).
    b. Visualización del código fuente del envío seleccionado.

10. Blog
    a. Creación entradas de blog con título, contenido y etiquetas.
    b. Sistema de comentarios para interacción entre usuarios.

11. Competencias
    a. Competencias en tiempo real.
    b. Sistema de ranking basado en los resultados de los usuarios.

Requerimientos no funcionales:

1. Facilidad de Uso: Navegación clara con menús, botones bien definidos y proceso sencillo para navegar por la plataforma.

2. Accesibilidad y Rendimiento: Compatibilidad con navegadores modernos en dispositivos móviles y computadoras y capacidad de manejar al menos 100 usuarios simultáneos.

3. Seguridad: Cifrado de contraseñas y comunicaciones seguras mediante HTTPS.

2. Análisis de requerimientos

Se hace una clasificación de requerimientos usando el método Moscow, teniendo en cuenta la complejidad técnica, los recursos disponibles, el impacto en la experiencia del usuario y la relación con los objetivos del proyecto.

Must:

- *Registro de usuarios* ***(3 días)****:* El registro de usuarios es fundamental para la plataforma, ya que permite a los usuarios crear una cuenta y almacenar su progreso. Su implementación no es compleja, ya que solo requiere almacenamiento de credenciales y validaciones básicas, aunque debe garantizar una experiencia fluida para el usuario.

- *Inicio de sesión* ***(5 días)****:* El inicio de sesión seguro con roles definidos es esencial para diferenciar entre administradores, usuarios y problemsetters, asegurando que cada uno tenga los permisos adecuados. Esto implica manejar sesiones, cifrado de contraseñas y validaciones de seguridad, lo que aumenta su complejidad.

- *Lista de problemas filtrable por estado* ***(3 días)****:* La lista de problemas filtrable por estado (resuelto/no resuelto) facilita la organización y acceso a los problemas, mejorando la experiencia del usuario. Técnicamente, solo requiere una consulta optimizada a la base de datos, por lo que su implementación es sencilla.

- *Envío de soluciones y evaluador automático* ***(13 días)****:* El envío de soluciones y evaluador automático es la funcionalidad central del sistema. Requiere ejecutar código de manera segura, comparando su salida con los casos de prueba predefinidos, lo que aumenta significativamente su complejidad. Su correcta implementación garantiza la usabilidad y confiabilidad de la plataforma.

- *CRUD de problemas* ***(5 días)****:* El CRUD de problemas permite a los administradores y problemsetters gestionar los problemas disponibles en la plataforma. Aunque no es una funcionalidad compleja, sí requiere una interfaz clara y validaciones para garantizar la calidad de los problemas.

- *Tabla con detalles de los envíos **(3 días)**:* La tabla con detalles de los envíos proporciona información clave sobre los intentos realizados por el usuario, como fecha, problema y estado del envío. Técnicamente, solo requiere consultas a la base de datos y renderizado en tabla.

- *Seguridad **(8 días)**:* La seguridad es crucial para proteger la plataforma contra ataques como inyecciones SQL o XSS. Su implementación requiere aplicar buenas prácticas de desarrollo, validaciones y almacenamiento seguro de contraseñas.

Should:

- *Función de búsqueda de problemas **(3 días)**:* La función de búsqueda de problemas mejora la accesibilidad, permitiendo encontrar problemas por título o descripción. Su implementación implica optimizar consultas y manejar índices en la base de datos.

- *Sección vinculada a cada problema con explicaciones y ejemplos detallados **(3 días)**:* La sección de explicaciones detalladas ayuda a los usuarios a comprender mejor los problemas, proporcionando ejemplos y estrategias de resolución. Esto requiere una estructura para almacenar contenido y una interfaz adecuada para mostrarlo.

- *Visualización de problemas resueltos, tasa de éxito e historial gráfico del progreso **(5 días)**:* La visualización de problemas resueltos y estadísticas permite a los usuarios monitorear su progreso mediante gráficos y métricas clave. Su implementación requiere procesar datos históricos y generar visualizaciones dinámicas

- *Visualización del código fuente del envío seleccionado **(3 días)**:* La visualización del código fuente de envíos permite a los usuarios analizar sus intentos previos y aprender de sus errores. Dado que solo requiere almacenar y recuperar el código, su implementación es sencilla.

- *Facilidad de Uso **(5 días)**:* La facilidad de uso es clave para garantizar una experiencia intuitiva. Involucra el diseño de una interfaz clara, pruebas de usabilidad y ajustes según la retroalimentación de los usuarios.

- *Accesibilidad y Rendimiento **(3 días)**:* La accesibilidad y rendimiento asegura que la plataforma funcione bien en distintos dispositivos y sea compatible con tecnologías de asistencia. Su implementación implica optimizar tiempos de carga y mejorar compatibilidad.

Could:

- *Lista de amigos **(3 días)**:* La lista de amigos permite a los usuarios conectarse entre sí, fomentando la motivación y la competencia amistosa. Su implementación requiere manejar relaciones en la base de datos y una interfaz adecuada.

- *Gestión de solicitudes de amistad **(5 días)**:* La gestión de solicitudes de amistad complementa la lista de amigos, permitiendo enviar, aceptar o rechazar solicitudes. Aunque es una funcionalidad estándar en muchas plataformas, requiere lógica adicional para la gestión de estados y notificaciones.

- *Ejecución de código con entradas personalizadas en un entorno aislado **(5 días)**:* La ejecución de código con entradas personalizadas permite a los usuarios probar su código antes de enviarlo. Su implementación es similar al evaluador automático, pero sin comparación de salida, lo que aún requiere un entorno de ejecución seguro.

- *Explicaciones paso a paso **(5 días)**:* Las explicaciones paso a paso ofrecen guías interactivas para resolver problemas, facilitando el aprendizaje progresivo. Esto requiere una estructura para desglosar explicaciones y una interfaz que permita navegar entre pasos.

Wont:

- Creación entradas de blog con título, contenido y etiquetas.
- Sistema de comentarios para interacción entre usuarios.
- Competencias en tiempo real.
- Sistema de ranking basado en los resultados de los usuarios.

El análisis se resume en la siguiente tabla:

Requerimiento	**Prioridad**	**Estimación (Días)**
Registro de usuarios	Must	3
Inicio de sesión		5
Lista de problemas filtrable por estado		3
Envío de soluciones y evaluador automático		13
CRUD de problemas		5
Tabla con detalles de los envíos		3
Seguridad		8
Función de búsqueda de problemas	Should	3
Sección vinculada a cada problema con explicaciones y ejemplos detallados		3
Visualización de problemas resueltos, tasa de éxito e historial gráfico del progreso		5

Visualización del código fuente del envío seleccionado		3
Facilidad de Uso		5
Accesibilidad y Rendimiento		3
Lista de amigos	Could	3
Gestión de solicitudes de amistad		5
Ejecución de código con entradas personalizadas en un entorno aislado		5
Explicaciones paso a paso		5
Total		80

3. Análisis gestión de software

Costo

El proyecto planteado no representa ningún costo, sin embargo para este punto se contempla el escenario del despliegue web.

Rol	Días	Costo por día (USD)	Total (USD)
Desarrollador Full-Stack (Senior)	55	$187	$10,285
Diseñador UX/UI	8	$80	$640
Servicios en la nube	30	-	$200
Licencias y herramientas	30	-	$100
Infraestructura (servidores)	30	-	$150
Total estimado			$11,375

Dentro de licencias y herramientas se tienen en cuenta; figma, Adobe y asistente de IA. Aunque es probable que los profesionales contratados cuenten con estas herramientas, se deciden agregar aparte para conocer todos los valores relacionados al proyecto. Para los servidores se busca un

servicio que nos ofrezca una capacidad de 50,000 usuarios mensuales. Finalmente para servicios en la nube se buscan tarifas de AWS para tener almacenamiento en la nube.

Alcance

Incluido en el MVP

- Sistema de autenticación (Registro, login).
- Evaluación de soluciones enviadas por los usuarios y Runner.
- CRUD de problemas (crear, editar, eliminar, listar problemas).
- Página de perfil de usuario con estadísticas básicas.
- Interfaz básica en frontend con diseño funcional.
- Historial de envíos accesible para todos los usuarios.

Fuera del alcance del MVP

- Autenticación por terceros (Google, GitHub).
- Recuperación de contraseña
- Sistema de ranking avanzado con clasificaciones detalladas.
- Modo de competencia en tiempo real.
- Soporte para múltiples idiomas.

4. Diseño y arquitectura

Descripción de la arquitectura de la plataforma

La plataforma está diseñada utilizando una arquitectura de microservicios, lo que permite modularidad, escalabilidad y flexibilidad en su desarrollo. La arquitectura se compone de los siguientes elementos principales:

- *Frontend:* Una interfaz web que permite a los usuarios interactuar con la plataforma, explorar problemas, enviar soluciones, consultar su historial de submissions, entre otras acciones.
- *Backend:* Un servicio central que gestiona la lógica de negocio y se encarga de acceder a la base de datos relacional, almacenando información sobre usuarios, problemas, submissions y sus resultados.
- *API de compilación y ejecución:* Un microservicio especializado que recibe el código fuente enviado por los usuarios, lo compila y lo ejecuta con casos de prueba definidos, devolviendo los resultados al backend para su validación.

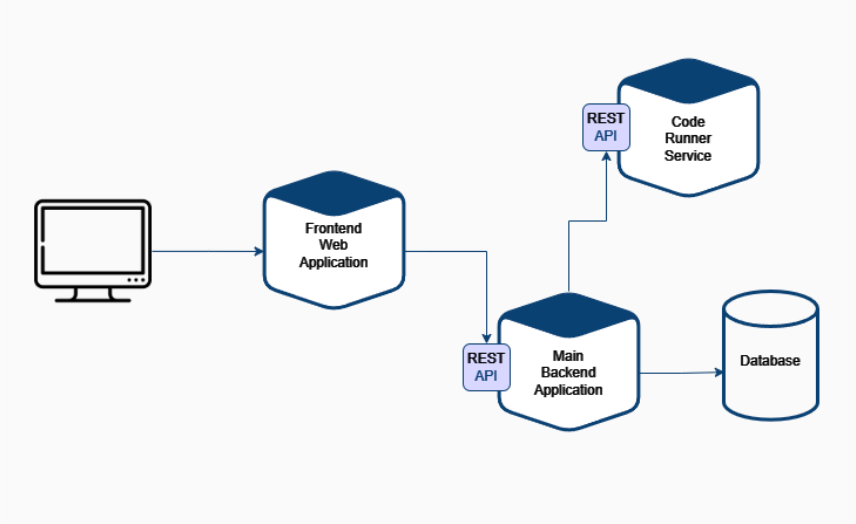

Diagrama de arquitectura del sistema

Como se observa en el diagrama, el cliente interactúa únicamente con el frontend de la aplicación web, el cual se comunica con la "main backend application", el cual es el microservicio encargado de gestionar toda la lógica de negocio, y gestionar la base de datos, vemos como este microservicio se comunica con el "code runner service", esto lo hace para compilar y ejecutar el código enviado por el usuario de manera independiente, para su posterior verificación y evaluación.

El uso de una arquitectura de microservicios ofrece varias ventajas en comparación con otros enfoques como por ejemplo un enfoque monolítico:

- *Escalabilidad:* Cada componente del sistema puede escalarse de manera independiente según la demanda. Por ejemplo, si la API de compilación y ejecución requiere más recursos debido a un alto volumen de submissions, puede desplegarse en instancias adicionales sin afectar al resto del sistema.
- *Flexibilidad y mantenimiento:* Los microservicios permiten desarrollar y actualizar componentes del sistema de forma independiente, lo que facilita la incorporación de nuevas funcionalidades y la resolución de errores sin interrumpir todo el sistema.
- *Resiliencia:* Un fallo en un microservicio no afecta al resto de la aplicación. Por ejemplo, si la API de compilación falla, el frontend y el backend pueden seguir operando con sus demás funciones, notificando a los usuarios de la imposibilidad de realizar envíos de problemas en el momento, pero permitiéndoles acceder al resto de sus datos y estadísticas.
- *Despliegue independiente:* En caso de llegar a desplegar la aplicación las actualizaciones pueden realizarse en un microservicio sin necesidad de desplegar todo el sistema, reduciendo tiempos de inactividad y facilitando la entrega continua.

En resumen, la arquitectura de microservicios proporciona una plataforma más robusta, escalable y mantenible, asegurando un mejor rendimiento y experiencia para los usuarios.

Diseño de base de datos

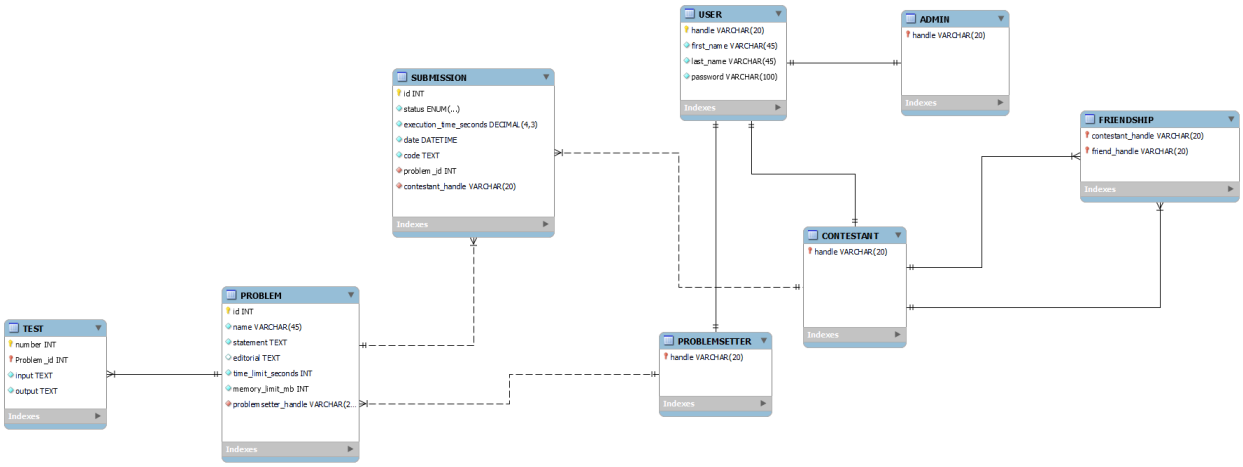

Diagrama entidad-relación

La base de datos se diseñó para cumplir las formas normales de las bases de datos, evitando así redundancia en la información y asegurando la facilidad para ser consultada. Se asignó la cardinalidad de cada relación pensando en la posterior escalabilidad del sistema. Por ejemplo, se asignó una relación de uno a muchos entre las entidades *TEST* y *PROBLEM*, en caso de que en un futuro se decida añadir más de un grupo de test cases a los problemas. Además se decidió utilizar herencia para los roles de usuario, teniendo una entidad de usuario general de la cual heredan *CONTESTANT*, *ADMIN* y *PROBLEMSETTER*, los cuales son los 3 roles existentes en nuestro sistema.

Con respecto a los índices, MySQL (el motor de base de datos utilizado) crea índices por defecto en las columnas *UNIQUE* y en las llaves primarias y foráneas. Estos fueron los únicos índices utilizados, pues la mayoría de las consultas van a utilizar estos atributos y no se consideran necesarios más índices.

Adicionalmente la base de datos cuenta con vistas, en las cuales se encapsularon algunas de las consultas más utilizadas por el backend. También se definieron varios procedimientos almacenados y funciones que son llamados desde el backend para evitar escribir el SQL textual en el backend (lo cual haría el sitio vulnerable a SQL Injection).

Se decidió hacer uso de una base de datos relacional debido a que este modelo presenta una mayor facilidad para estructurar la información de manera lógica y organizada.

5. Patrones de diseño

No se han implementado por el momento.