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.
- Inicio de sesión.
- Lista de problemas filtrables por estado.
- Envío de soluciones y evaluador automático.
- CRUD de problemas.
- Tabla con detalles de los envíos.
- Seguridad.

Should:

- Función de búsqueda de problemas.
- Sección vinculada a cada problema con explicaciones y ejemplos detallados.
- Visualización de problemas resueltos, tasa de éxito e historial gráfico del progreso.
- Visualización del código fuente del envío seleccionado.
- Facilidad de Uso.
- Accesibilidad y Rendimiento.

Could:

- Lista de amigos.

- Gestión de solicitudes de amistad.
- Ejecución de código con entradas personalizadas en un entorno aislado.
- Explicaciones paso a paso.

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 de la complejidad de las funcionalidades y estimación de tiempo se presenta en la siguiente tabla:

Nombre	**Descripción**	**Complejidad**	**Estimación (Días)**
Registro de usuarios	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.	Baja	3
Inicio de sesión	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.	Baja	5
Lista de problemas filtrable por estado	La lista de problemas filtrable por estado (resuelto/no resuelto) facilita la organización y acceso a los problemas, mejorando la	Baja	3

	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	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.	Alta	13
CRUD de problemas	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.	Media	5
Tabla con detalles de los envíos	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.	Baja	3
Seguridad	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.	Alta	8
Función de	La función de búsqueda de	Baja	3

búsqueda de problemas	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	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.	Baja	3
Visualización de problemas resueltos, tasa de éxito e historial gráfico del progreso	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	Media	5
Visualización del código fuente del envío seleccionado	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.	Baja	3
Facilidad de Uso	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.	Baja	5
Accesibilidad y Rendimiento	La accesibilidad y rendimiento asegura que la plataforma funcione bien en distintos dispositivos y sea compatible con tecnologías de asistencia. Su implementación implica optimizar	Baja	3

	tiempos de carga y mejorar compatibilidad.		
Lista de amigos	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.	Baja	3
Gestión de solicitudes de amistad	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.	Baja	5
Ejecución de código con entradas personalizadas en un entorno aislado	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.	Media	5
Explicaciones paso a paso	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.	Media	5
Total			80

3. Análisis gestión de software

Contexto

Para la elaboración de esta tabla de costos se consideraron tres aspectos clave: desarrolladores, servicios/herramientas externas y host/dominio.

Dado que el equipo está compuesto por tres integrantes, se tomó como referencia el salario de tres desarrolladores full-stack para encontrar una estimación acorde a nuestras condiciones.

En cuanto a los servicios en la nube y herramientas externas, se incluyeron tanto API como almacenamiento. Si bien existen diversas opciones en el mercado, como Microsoft Azure, se optó por AWS debido a su amplio uso entre desarrolladores y su equilibrio entre precio y calidad. No obstante, la mayoría de estas herramientas presentan costos similares. Microsoft Azure, por ejemplo, tiene un costo aproximado de $92.600.

Para el host y el dominio, la recopilación de precios se realizó utilizando las calculadoras disponibles en las páginas web mencionadas en la tabla, lo que permitió obtener estimaciones precisas y comparables.

Rol	Costos mensuales (COP)	Descripción	Total
Desarrollador Full-Stack	$ 3'850.000	En este proyecto participarán tres desarrolladores, asegurando un equipo equilibrado y acorde a las necesidades del desarrollo.	$11'550,000 - mensual
Diseñador UX/UI	$ 4'150.000	Un diseñador será responsable del modelado, diseño de marca, creación del logo y selección de la paleta de colores, garantizando una identidad visual coherente.	$ 4'150.000 - mensual

Servicios en la nube	$ 87.000 (aprox)	Para el almacenamiento en la nube, se evaluaron las tarifas de AWS con el fin de seleccionar la opción más adecuada en términos de costo y rendimiento.	$ 87.000 - mensual
Licencias y herramientas	$ 0	Dado que Microsoft Visual Studio Code cuenta con una licencia privada y comercial, se considera una excelente elección como entorno de desarrollo para este proyecto.	$0
Host	$ 24.500	Se analizaron los precios de la plataforma Hostinger en USD para optimizar el costo del alojamiento web.	$ 24.500
Dominio	$ 5.000	También se tomaron en cuenta los precios ofrecidos por GoDaddy para comparar opciones y tomar la mejor decisión en cuanto a dominios y otros servicios.	$ 5.000
Total			**$15'779.500**

Fuentes:

- https://cart.hostinger.com/pay/8bf02091-c549-4b5e-ac17-18f4696badc2?from=websites
- https://www.godaddy.com/es/domainsearch/find?domainToCheck=codes
- El resto de valores se sacaron de estimaciones generales,

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

- Interacción entre usuarios por blogs o comentarios
- Competencias en tiempo real
- Sistema de clasificación de usuarios (Ranking)

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.

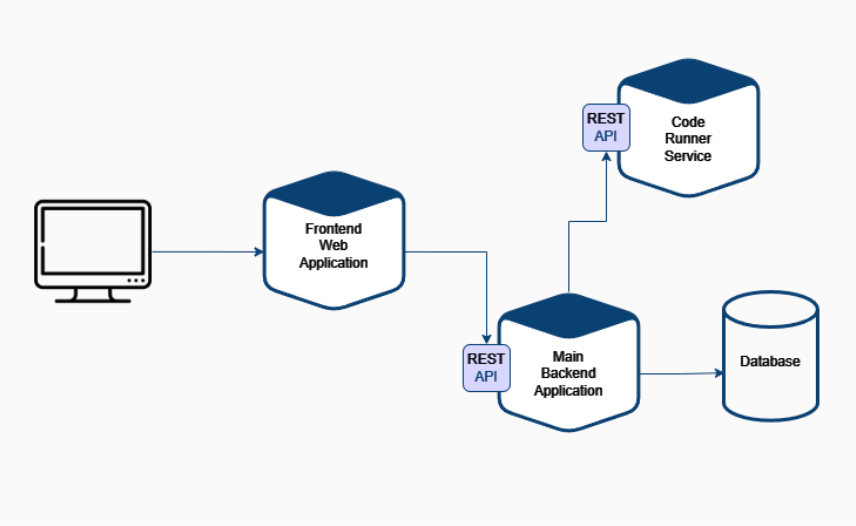

Diagrama de arquitectura del sistema

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

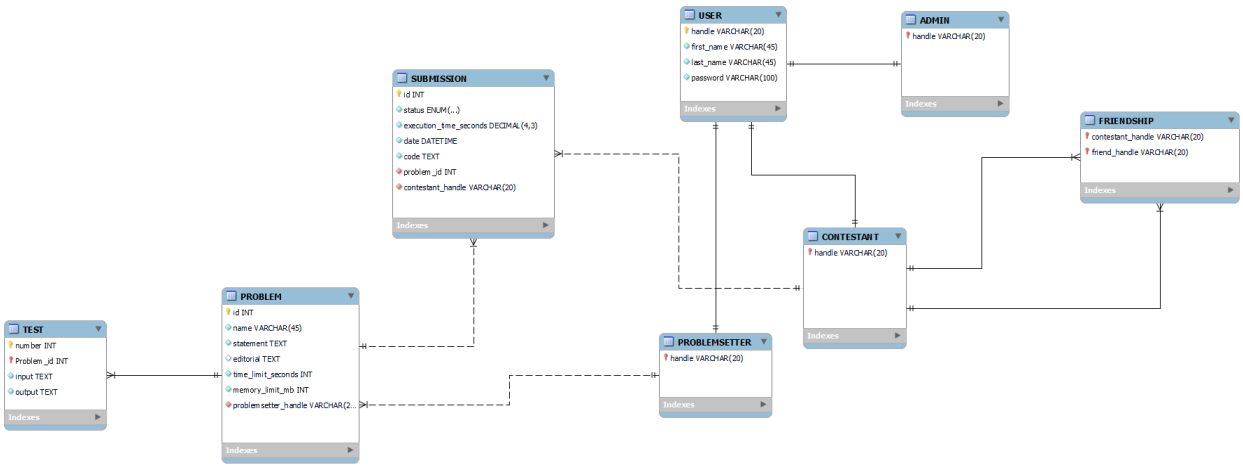

Diagrama entidad-relación

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

En el proceso de desarrollo del proyecto, se han identificado e implementado los siguientes patrones de diseño.

Patrón de diseño "Strategy"

- **¿Qué problema resuelve?**
  En una etapa inicial del proyecto, se tomaba cada menú de filtros como una instancia de menú completamente independiente, lo cual nos llevó a repetir el mismo código una y otra vez, pues el estilo del menú sería el mismo gráficamente, siempre a pesar de que cambiaran sus opciones de filtrado. El patrón **strategy** nos permitió tener un único menú completamente adaptable a cualquier grupo de filtros con su propia "estrategia".

- **¿Por qué es necesario en el proyecto?**
  El sistema requiere múltiples menús de filtros en distintas secciones, como el listado de problemas, el historial de envíos, el menú para agregar problemas, amigos, entre otros muchos, cada uno con criterios de filtrado específicos. Sin una solución adecuada, esto llevó en un principio a la duplicación de código y a una estructura rígida difícil de mantener. La implementación del patrón Strategy permitió como se mencionó anteriormente encapsular la lógica de filtrado en funciones separadas, haciendo que el menú sea reutilizable y adaptable sin modificar su estructura. Esto es indispensable para el proyecto, pues en la actualidad nos ahorra tiempo y código innecesario.

- **¿Cómo está siendo implementado?**
  Se ha creado un componente de React genérico de menú de filtros que recibe dinámicamente una lista de opciones y una función de estrategia que define cómo se aplican los filtros. Cada instancia del menú puede recibir una estrategia diferente, permitiendo, por ejemplo, filtrar problemas por dificultad o envíos por estado sin necesidad de reescribir el componente. Esta implementación refuerza la modularidad y la reutilización del código dentro del sistema.

Patrón de diseño "Composite"

- **¿Qué problema resuelve?**
  El sistema cuenta con múltiples páginas que comparten elementos comunes, como por ejemplo la barra de navegación del sitio, el pie de página del mismo y un selector de página para las secciones del sitio donde se presenta un gran volumen de registros y se requiere realizar "paginación". Sin una estructura adecuada, habría duplicación de código y dificultades para mantener la consistencia en la interfaz. Además, cada página debe poder organizar sus propios elementos de manera flexible sin perder la uniformidad del diseño.

- **¿Por qué es necesario en el proyecto?**
  Dado que React se basa en una arquitectura de **componentes reutilizables**, es fundamental organizar estos componentes de manera jerárquica y modular. Usar el **patrón Composite** permite que cada página actúe como un **contenedor** que agrupa distintos componentes, manteniendo una estructura clara y flexible. Esto facilita la escalabilidad y la reutilización del código, ya que cualquier cambio en los componentes principales se refleja en todas las páginas sin necesidad de modificar cada una por separado.

- **¿Cómo está siendo implementado?**
  El proyecto define componentes como *"Nav", "Footer", "Page Selector",* entre otros, los cuales se integran en estructuras principales. Cada página **compone** su contenido utilizando estos elementos reutilizables, asegurando una interfaz coherente sin acoplamiento innecesario. Gracias a este enfoque, se pueden añadir nuevos componentes o modificar los existentes sin afectar la estructura global de la aplicación.