



Consolidación del Proyecto API en C++

Proyecto Prograthon: sistema integral de gestión académica con trazabilidad de competencias y concursos de programación

Rodolfo Ángel Gilart Pulido

Jhonatan Camilo Garrido Gomez

Laura Catalina Riveros Flórez

Julián Andrés Cuadros Muñoz

Universidad católica de Colombia

Desarrollo de software

Facultad de ingeniería

Bogotá D.C.

2025

Tabla de Contenido

1. Descripción Del Proyecto	6
1.0. Control De Cambios	6
1.1. Introducción	8
1.2. Planteamiento Del Problema	8
1.3. Pregunta Problema	9
1.4. Funcionalidad API	9
1.4.1. Arquitectura y Tecnologías	9
1.4.2. Gestión de Usuarios	9
1.4.3. Sistema de Maratones	9
1.4.4. Gestión de Problemas	10
1.4.5. Seguridad y Permisos	10
1.4.6. Operaciones de Base de Datos	10
1.5. Objetivos	10
1.5.1. Objetivo general:	10
1.5.2. Objetivos específicos:	11
1.6. Justificación	11
2. Tecnologías Utilizadas	11
2.1. API Rest en C++	11
2.2. JWT	12
2.3. Base De Datos PostgreSQL	12
2.4. Gestión de dependencias (vcpkg):	12
2.5. Sistema de construcción:	12
2.6. Frontend con React:	13
2.7. Autenticación y seguridad (JWT y Argon2):	13
2.8. Control De Versiones	14
3. Metodologías	17
3.1. Explicación de la metodología	17
3.2. Fases	17
3.3. Asignación De Roles	20
3.4. Requerimientos	20
3.4.1. Requerimientos Funcionales.	20

3.4.1. Requerimientos No Funcionales.	23
3.4.1.1 Rendimiento:	23
3.4.1.2 Mantenibilidad:	23
3.4.1.3 Usabilidad:	24
3.4.1.4 Fiabilidad:	24
3.4.1.5 Seguridad	25
3.4.1.6 Compatibilidad	26
3.5. Arquitectura	26
3.5.1. Casos De Uso.	27
3.5.2. Diagrama De Secuencia.	28
3.5.3. Diagrama De Componentes.	30
3.5.4. Diagrama De Vista De Despliegue.	31
3.5.4. Diagrama De Paquetes.	31
4. CI/CD – GitHub Actions + Docker	32
4.1. Integración continua	32
4.2. Despliegue continuo	33
4.3. Docker	34
4.3.1. Archivos Dockerfile y Dockerk-compose.yml	35
5. Costo / Beneficio	37
5.1. Tiempo Invertido VS Funcionalidades logradas	37
5.2. Beneficio técnico o comercial simulado	37
5.3. ¿Cómo escalaría o evolucionaría esta solución?	37
6. Errores Y Soluciones	38
8. Objetivos Logrados	39
8.1. Requerimientos Funcionales logrados	39
8.2. Checklist	40
9. Conclusiones	40
10. Bibliografía	41

Tabla de tablas

<i>Tabla 1 Control de cambios</i>	6
<i>Tabla 2 RF001</i>	20
<i>Tabla 3 RF002</i>	21
<i>Tabla 4 RF003</i>	21
<i>Tabla 5 RF004</i>	21
<i>Tabla 6 RF005</i>	22
<i>Tabla 7 RF006</i>	22
<i>Tabla 8 RNF001</i>	23
<i>Tabla 9 RNF002</i>	23
<i>Tabla 10 RNF003</i>	24
<i>Tabla 11 RNF004</i>	24
<i>Tabla 12 RNF005</i>	25
<i>Tabla 13 RNF006</i>	26

Tabla de ilustraciones

<i>Ilustración 1 Roadmap_1</i>	14
<i>Ilustración 2 Roadmap_2</i>	14
<i>Ilustración 3 Backlog iteración 1</i>	15
<i>Ilustración 4 Backlog interacción 2</i>	15
<i>Ilustración 5 Backlog interacción 3</i>	16
<i>Ilustración 6 Burn up proyecto</i>	16
<i>Ilustración 7 Assignees issues</i>	17
<i>Ilustración 8 Fases del proyecto</i>	19
<i>Ilustración 9 Roles del proyecto</i>	20
<i>Ilustración 10 Casos de uso</i>	27
<i>Ilustración 11 Vista de procesos</i>	28
<i>Ilustración 12 Procesos de diagrama</i>	29
<i>Ilustración 13 Componentes</i>	30
<i>Ilustración 14 Vista de despliegue</i>	31
<i>Ilustración 15 Paquetes</i>	31
<i>Ilustración 16 CI/CD</i>	32
<i>Ilustración 17 Dockerfile Backend</i>	35
<i>Ilustración 18 Dockerfile Frontend</i>	36
<i>Ilustración 19 Docker-compose.yml</i>	36
<i>Ilustración 20 Revisión de requerimientos cumplidos</i>	39
<i>Ilustración 21 Checklist de calificación</i>	40

1. Descripción Del Proyecto

1.0. Control De Cambios

Tabla 1 Control de cambios

Versión del documento	Fecha	Estado del documento	Nombre del archivo	Cambios realizados	Responsable/s de las modificaciones
0.1	24/06/2025	En revisión	Documentación general del proyecto	Se agregan la introducción, planteamiento del problema, requerimientos y diagramas del proyecto	Catalina Julián Rodolfo Jhonatan
0.2	25/06/2025	Finalizado	Documentación general del proyecto	Se modifica el objetivo general y los objetivos específico en base a las recomendaciones del profesor	Catalina
0.3	25/06/2025	Finalizado	Documentación general del proyecto	Se detallan las fases de la metodología a seguir en el proyecto	Catalina Julián Rodolfo Jhonatan
0.4	25/06/2025	Finalizado	Documentación general del proyecto	Se modifican los diagramas del proyecto en base a las recomendaciones del profesor	Catalina Julián Jhonatan
0.5	25/06/2025	Finalizado	Documentación general del proyecto	Se modifica la tabla de los roles de cada integrante del proyecto	Jhonatan
0.6	30/06/2025	Finalizado	Documentación general del proyecto	Se modifican los requerimientos no funcionales de acuerdo con la ISO25010	Catalina
0.7	8/07/2025	Finalizado	Codificación del programa	Después de haber presentada muchos	Catalina, Julian, Rodolfo, Jhonatan

				problemas de importacion y conexion se realiza un programa nuevo, teniendo en cuenta, lo solicitado por el cliente.	
0.8	8/07/2025	Finalizado	Documentacion proyecto	Se investiga y se documenta nuevamente las herramientas utilizadas,	Julian, Catalina
0.9	8/07/2025	Finalizado	Arquitectura	Se modifican los requerimientos, los diagramas de acuerdo con el nuevo programa	Julian, Catalina
10.0	9/07/2025	Finalizado	Frontend	Se mejora la parte visual del proyecto.	Rodolfo
11.0	9/07/2025	Finalizado	Dockers	Se realizan los dockers del proyecto	Jhonatan
12.0	9/07/2025	En proceso	Pruebas	Se realizan pruebas finales de funcionalidad del proyecto con todo lo implementado.	Rodolfo, Catalina, Julian, Jhonatan.
13.0	9/07/2025	En proceso	Actualización repositorio	Se actualiza con todo lo nuevo implementado el repositorio de GitHub	Julian

1.1. Introducción

Según el periódico ElColombiano Colombia es el tercer país con más aumento en el campo laboral de programadores en Latinoamérica. En el 2022 se presentó un aumento de programadores del 33%, de acuerdo con el informe el país alcanza un aumento de 663.000 personas. El ministerio de las MinTic (el ministerio colombiano de tecnologías y comunicación) estima que para este año actual (2025) existirá un déficit entre 68.000 y 112.000 desarrolladoras de software en el país.

Bogotá al ser una de las ciudades más importantes del país y con un alto desarrollo en el área tecnológica es la ciudad que lidera la demanda en el campo de la programación, concentrando un 78% de las oportunidades laborales (Clavijo, 2024).

La carrera de ingeniería en Sistemas en la actualidad es una de las más competidas y solicitadas por los estudiantes en la capital del país. Se estima que cerca de 1.257 egresados ingresan anualmente al campo laboral en la ciudad lo que refleja que esta carrera tiene una empleabilidad bastante grande (Tropicana, 2025).

1.2. Planteamiento Del Problema

En Colombia, existe una brecha crítica entre la formación académica en TI y las demandas del sector laboral. Según El Heraldo (2021), "cerca del 90% de los egresados en áreas informáticas ingresan al mercado laboral con habilidades de programación insuficientes" (p. 1A). Esto ha generado:

- Un desempleo del 18.5% en jóvenes del sector (Departamento Administrativo Nacional de Estadística [DANE], 2023, p. 34).
- 83,000 vacantes tecnológicas sin cubrir debido a la falta de competencias prácticas (Ministerio de Tecnologías de la Información y las Comunicaciones [Mintic], 2024, p. 12).

Algunas de las causas que han sido identificadas sobre este problema es:

- Desfase curricular: El 70% de las universidades priorizan contenidos teóricos sobre programación aplicada (Agencia Nacional de Evaluación de la Calidad y Acreditación [ANECA], 2022, p. 45).
- La investigación indica que la tasa promedio de aprobación en entrevistas técnicas se sitúa en torno al 54% (Stevens-Huffman, 2023). Sin embargo, esta cifra puede descender drásticamente a un rango del 15% al 20% en gigantes tecnológicos con procesos de contratación rigurosos y altamente competitivos (Stevens-Huffman, 2023). Las tasas de aceptación general en empresas como Google, Facebook, Amazon, Netflix y Apple son incluso más bajas, oscilando entre el 0.1% y el 3% de los solicitantes (Rosidi, 2023). Es importante señalar que la estadística del "18.75% de los estudiantes resolvió correctamente el problema de codificación más difícil" es una atribución errónea; esta cifra se refiere a la tasa de éxito de los candidatos que aprobaron ambos grupos del

examen final de Chartered Accountancy (CA) del ICAI, no a un desafío de codificación (The Economic Times, 2025; The Times of India, 2025).

1.3. Pregunta Problema

¿Cómo diseñar e implementar un aplicativo interactivo de programación competitiva, que reduzca la brecha entre las habilidades técnicas de los egresados de la Universidad Católica de Colombia y las exigencias del sector laboral tecnológico, mediante un espacio de entrenamiento práctico, evaluación continua?

1.4. Funcionalidad API

API C++

1.4.1. Arquitectura y Tecnologías

El sistema está construido usando C++ con varias librerías especializadas como httpplib para el servidor HTTP (Yamaoka, n.d.), nlohmann/json para manejo de JSON (Nlohmann, n.d.), y libpq para conectividad con PostgreSQL (PostgreSQL Global Development Group, n.d.). La aplicación sigue una arquitectura REST API, un estilo arquitectónico que se basa en un conjunto de principios para el diseño de servicios web (Fielding, 2000), y se conecta a una base de datos PostgreSQL. El backend implementa un sistema de autenticación robusto basado en JWT (JSON Web Tokens), un método estándar de código abierto para crear tokens de acceso seguros entre dos partes (Auth0, n.d.). Las contraseñas se hashen usando Argon2, un algoritmo de hashing criptográfico seguro y resistente a ataques de fuerza bruta y ataques laterales, que fue seleccionado como el ganador de la Password Hashing Competition (Biryukov et al., 2016). El sistema maneja tres roles de usuario: estudiantes, profesores y administradores, cada uno con diferentes niveles de permisos. La autenticación se verifica en cada endpoint protegido mediante tokens JWT que contienen información del usuario y expiran en 24 horas.

1.4.2. Gestión de Usuarios

La API proporciona endpoints completos para el manejo de usuarios, incluyendo registro, login, consulta de perfil actual, listado de usuarios (con restricciones según el rol), actualización de perfil propio, y operaciones administrativas como actualizar o eliminar usuarios. Los estudiantes solo pueden ver otros estudiantes, los profesores pueden ver estudiantes y profesores, mientras que los administradores tienen acceso completo a todos los usuarios.

1.4.3. Sistema de Maratones

El núcleo del sistema es la gestión de maratones de programación. Los profesores y administradores pueden crear maratones con un nombre, descripción y límite máximo de problemas. Cada maratón puede tener múltiples problemas asignados, pero está limitado por el parámetro `max_problems` establecido al crear la maratón. Los estudiantes pueden inscribirse en

las maratones disponibles, y los organizadores pueden ver qué estudiantes están registrados y eliminarlos si es necesario.

1.4.4. Gestión de Problemas

El sistema permite a profesores y administradores crear problemas de programación con título, descripción y nivel de dificultad (easy, medium, hard). Los problemas pueden ser asignados a maratones específicas, siempre respetando el límite máximo establecido para cada maratón. Los problemas pueden ser eliminados tanto individualmente como removidos de maratones específicas.

1.4.5. Seguridad y Permisos

La aplicación implementa un sistema de permisos granular donde cada endpoint verifica no solo la autenticación sino también los roles apropiados. Los estudiantes tienen permisos limitados (principalmente ver y inscribirse), los profesores pueden gestionar maratones y problemas, y los administradores tienen control total sobre usuarios, maratones y problemas. Además, implementa CORS para permitir acceso desde aplicaciones frontend y maneja adecuadamente las respuestas de error con códigos HTTP apropiados.

1.4.6. Operaciones de Base de Datos

Todas las operaciones de base de datos utilizan consultas parametrizadas para prevenir inyecciones SQL. El sistema maneja relaciones complejas entre usuarios, maratones y problemas, incluyendo tablas intermedias como `marathon_registrations` para inscripciones de estudiantes y `marathon_problems` para la asignación de problemas a maratones. Las operaciones de eliminación son cuidadosamente manejadas para mantener la integridad referencial. Este backend proporciona una API completa para gestionar competencias de programación, permitiendo a instituciones educativas organizar maratones, asignar problemas, gestionar participantes y mantener un control granular sobre los permisos y accesos del sistema.

1.5. Objetivos

1.5.1. Objetivo general:

Desarrollar un sistema web integrado de seguimiento académico y gestión de competencias de programación, con tecnologías escalables, para fortalecer las habilidades técnicas estudiantiles en un entorno educativo colaborativo-competitivo.

1.5.2. Objetivos específicos:

- Garantizar una arquitectura escalable que sustente la integración del seguimiento académico y gestión de competencias.
- Gestionar operaciones de usuarios, competencias y maratones mediante APIs seguras y base de datos en la nube.
- Implementar las funcionalidades esenciales de la aplicación web para permitir a los estudiantes registrarse en maratones de programación y acceder de forma clara a la lista de problemas asignados para cada maratón en la que se hayan inscrito.

1.6. Justificación

Este proyecto no solo resuelve una falla educativa demostrada con estadísticas nacionales, sino que se convierte en un catalizador de empleabilidad juvenil y modernización curricular. Con una inversión mínima y alta escalabilidad, posicionará a la Universidad Católica de Colombia como líder en innovación educativa para la Industria 4.0, cerrando la brecha entre las aulas y el mercado laboral tecnológico.

2. Tecnologías Utilizadas

Este proyecto implementa una API REST en C++ que gestiona registros, inicios de sesión y competencias de programación, enlazando con una base de datos PostgreSQL y un frontend en React. Cada tecnología cumple un rol específico y todas se integran para ofrecer un sistema completo: el frontend React ofrece la interfaz web, envía peticiones HTTP (por ejemplo con fetch) al servidor C++, el cual expone endpoints REST usando `cpp-httplib` (una biblioteca HTTP de C++ que permite definir rutas como `"/login"` o `"/maraton"`). Las solicitudes JSON del frontend se convierten en objetos C++ usando la biblioteca `nlohmann::json` (a través del header `json.hpp`), facilitando el manejo de datos estructurados. El servidor C++, compilado con `g++/MinGW`, atiende estas peticiones, usa `libpq` para enviar consultas SQL a PostgreSQL y devuelve respuestas JSON al frontend. En resumen, `cpp-httplib` y `json.hpp` proporcionan al C++ la capacidad de ser servidor HTTP/REST y procesar JSON.

2.1. API Rest en C++

Permite al servidor escuchar rutas y responder en formato JSON. `cpp-httplib` es una librería header-only que simplifica la creación de un servidor HTTP en C++. Con ella se definen handlers (funciones) para rutas (GET, POST, etc.), por ejemplo, `svr.Get("/ruta", ...)`. La biblioteca `nlohmann::json` (`json.hpp`) permite representar datos JSON como objetos C++ de forma intuitiva. Juntas hacen que el servidor C++ pueda recibir peticiones JSON del frontend, parsearlas a objetos nativos, procesar la lógica (por ejemplo, autenticación) y responder con JSON al cliente (MacLean, 2023).

2.2. JWT

JWT es un componente esencial en sistemas de maratones de programación con MERN, ofreciendo un equilibrio óptimo entre seguridad, rendimiento y escalabilidad. Su integración con el ecosistema JavaScript (desde React hasta Node.js) simplifica la gestión de identidades en entornos de alta demanda, donde la agilidad y confiabilidad son críticas (MaestrosWeb, s. f.).

2.3. Base De Datos PostgreSQL

PostgreSQL es un sistema gestor de bases de datos relacional open-source poderoso y escalable. Se usa para almacenar usuarios, roles y datos de las maratones. La biblioteca libpq es la interfaz C nativa de PostgreSQL, que permite que programas en C/C++ envíen consultas SQL al servidor de BD y reciban resultados. Desde el servidor C++ se llamaría a funciones de libpq (como PQconnectdb, PQexec, etc.) para conectarse, insertar, actualizar y consultar datos. En síntesis, libpq “traduce” las peticiones del código C++ a la base de datos PostgreSQL, garantizando que la API pueda gestionar datos persistentes con alta integridad y transaccionalidad (PostgreSQL Global Development Group, s. f.; PostgreSQL Documentation, 2025).

2.4. Gestión de dependencias (vcpkg):

vcpkg es un gestor de paquetes para C/C++ de Microsoft que simplifica instalar bibliotecas necesarias (como httpplib, json, libpq, JWT, Argon2) en Windows, Linux o macOS. En este proyecto, se usa vcpkg para obtener e integrar todas estas dependencias externas de forma automática, sin tener que compilarlas manualmente. Por ejemplo, al instalar con vcpkg install cpp-httpplib, el compilador encontrará el header httpplib.h. Esto garantiza versiones compatibles (ABI) y un entorno reproducible (Microsoft, 2024; Wikipedia, 2025).

2.5. Sistema de construcción:

CMake es una herramienta de generación de sistemas de construcción multiplataforma. En lugar de escribir makefiles específicos, se define un CMakeLists.txt que indica cómo compilar el proyecto. CMake puede generar los archivos de construcción nativos (en Windows, típicamente proyectos de Visual Studio o makefiles para MinGW). Luego se usa el compilador g++ (la parte de C++ de GCC) proporcionado por MinGW-w64 para compilar en Windows.

MinGW-w64 es esencialmente GCC adaptado a Windows, con headers y librerías que permiten compilar programas nativos de Windows. El archivo build.bat automatiza este proceso: al ejecutarlo, invoca cmake para generar la carpeta build, luego ejecuta cmake --build (o equivalente) usando g++, produciendo el ejecutable final. Así, con un comando, se construye todo el servidor C++ incluyendo librerías de vcpkg y se obtiene el ejecutable .exe listo (The CMake Team, 2025; MinGW-w64 Project, 2025).

2.6. Frontend con React:

React es una biblioteca de JavaScript para construir interfaces de usuario basadas en componentes. En este proyecto se emplea Create React App (`npx create-react-app`) como boilerplate que configura instantáneamente el proyecto frontend con hot reload. React permite organizar la UI en componentes (por ejemplo, formulario de login, vistas de maratones) que actualizan eficientemente la vista cuando cambian los datos. Además, se instala `react-router-dom`, que habilita enrutamiento en la aplicación de una sola página (SPA): sin recargar la página, se pueden definir diferentes “rutas” de URL internas que cargan componentes distintos (por ejemplo, `/login` o `/dashboard`), todo gestionado en el cliente. La comunicación con la API se hace típicamente usando `fetch()` o `axios`, enviando solicitudes HTTP al servidor C++ (React Core Team, s. f.; GeeksforGeeks, 2025).

2.7. Autenticación y seguridad (JWT y Argon2):

Para manejar seguridad y sesiones se utilizan JSON Web Tokens (JWT) y Argon2. Al iniciar sesión, el servidor C++ genera un JWT firmado (usando bibliotecas C++ específicas), que el cliente React almacenará (por ejemplo, en `localStorage`) y enviará en futuros requests como `Authorization: Bearer <token>`. Un JWT es un token compacto que contiene claims (información del usuario) y está firmado criptográficamente, permitiendo al servidor verificarlo sin consulta adicional a la base de datos. Esto permite que cada petición REST esté autenticada y autorizada según los roles del token. Para las contraseñas, se emplea Argon2, un algoritmo moderno de hash de contraseñas, ganador del Password Hashing Competition en 2015. Argon2 es resistente a ataques por GPU/ASIC porque es configurable en tiempo y memoria; se usa aquí para derivar hashes de contraseñas al registrarse, almacenando sólo el hash en la BD. Luego, al loguear, se verifica la contraseña ingresada con Argon2, asegurando que nunca se almacenen contraseñas en texto claro y protegiendo contra ataques de fuerza bruta (JWT.io, s. f.; Argon2 Documentation, 2025).

2.8. Control De Versiones

El control de cambios se realiza para llevar un seguimiento de lo que se ha realizado y lo que se debe realizar, este control de cambios lo llevamos mediante GitHub.

Ilustración 1 Roadmap_1

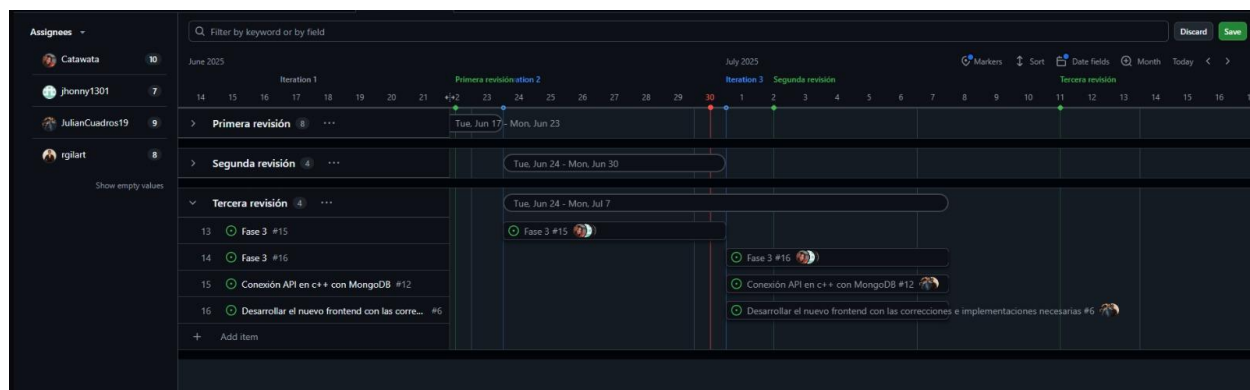


Ilustración 2 Roadmap_2

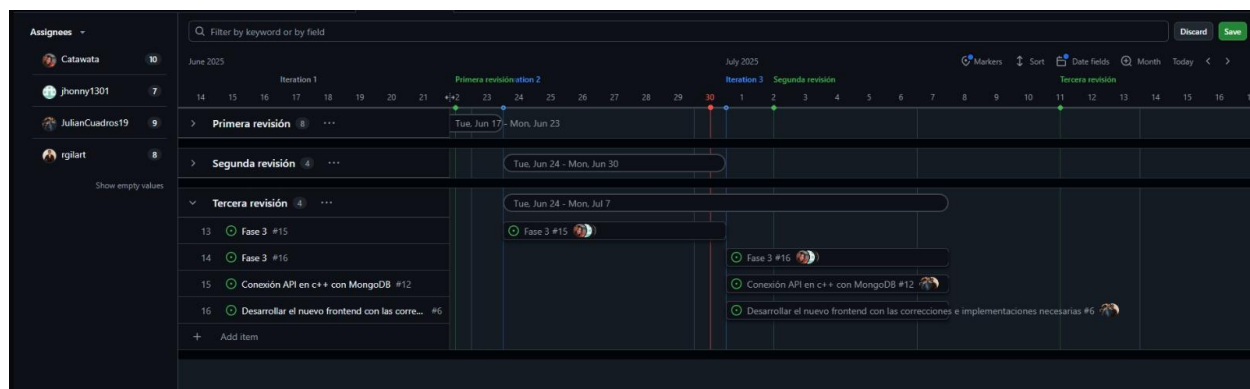


Ilustración 3 Backlog iteración 1

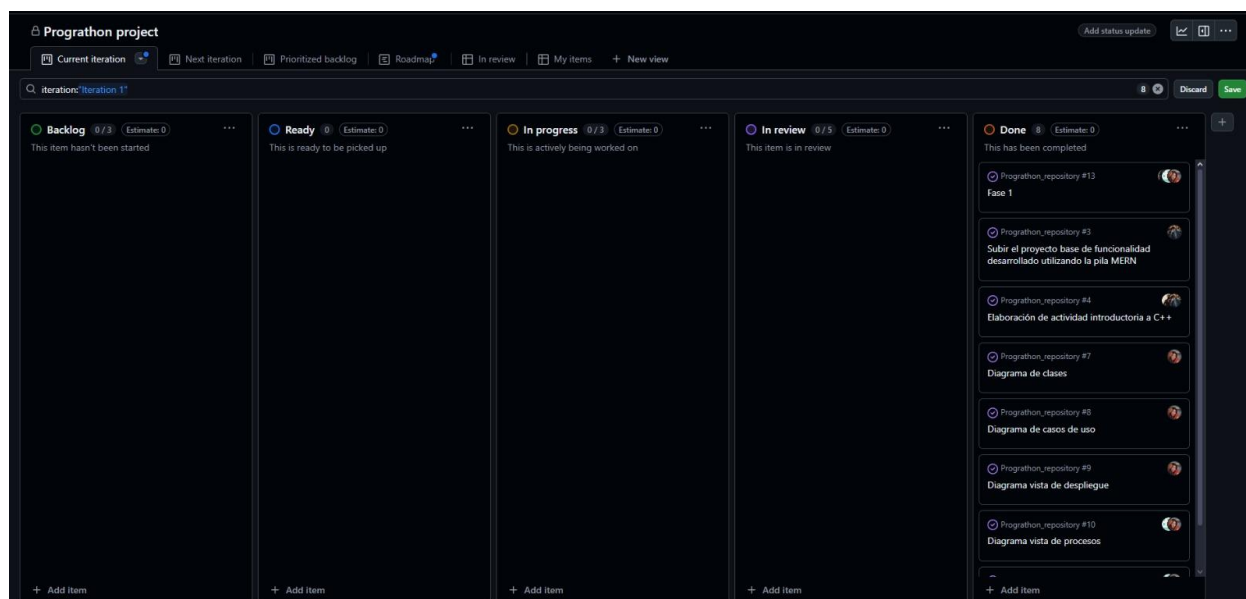


Ilustración 4 Backlog interacción 2

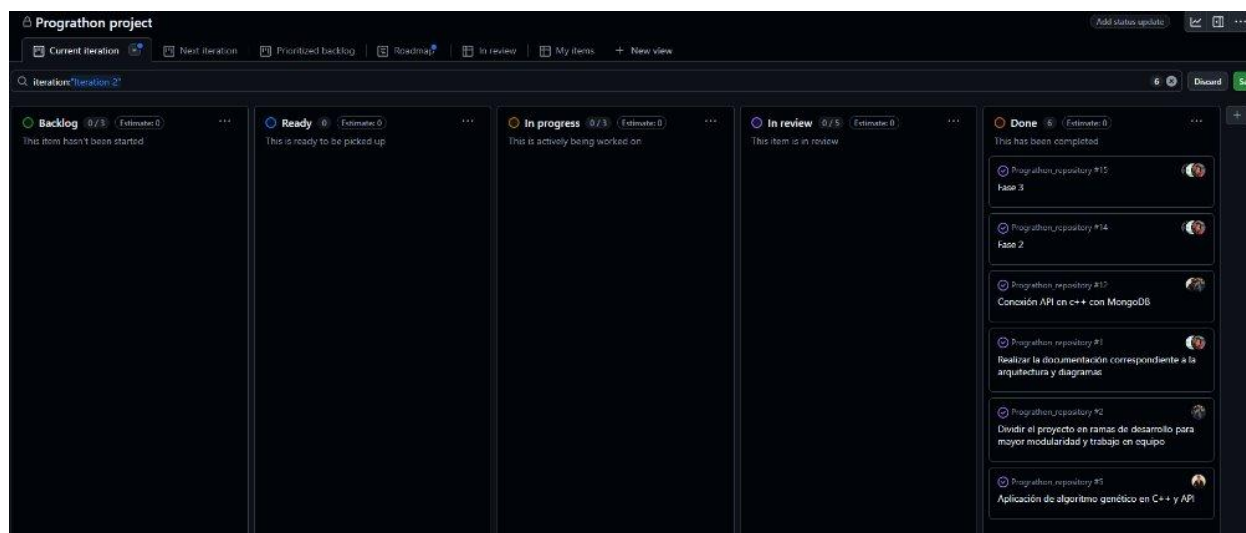


Ilustración 5 Backlog interacción 3

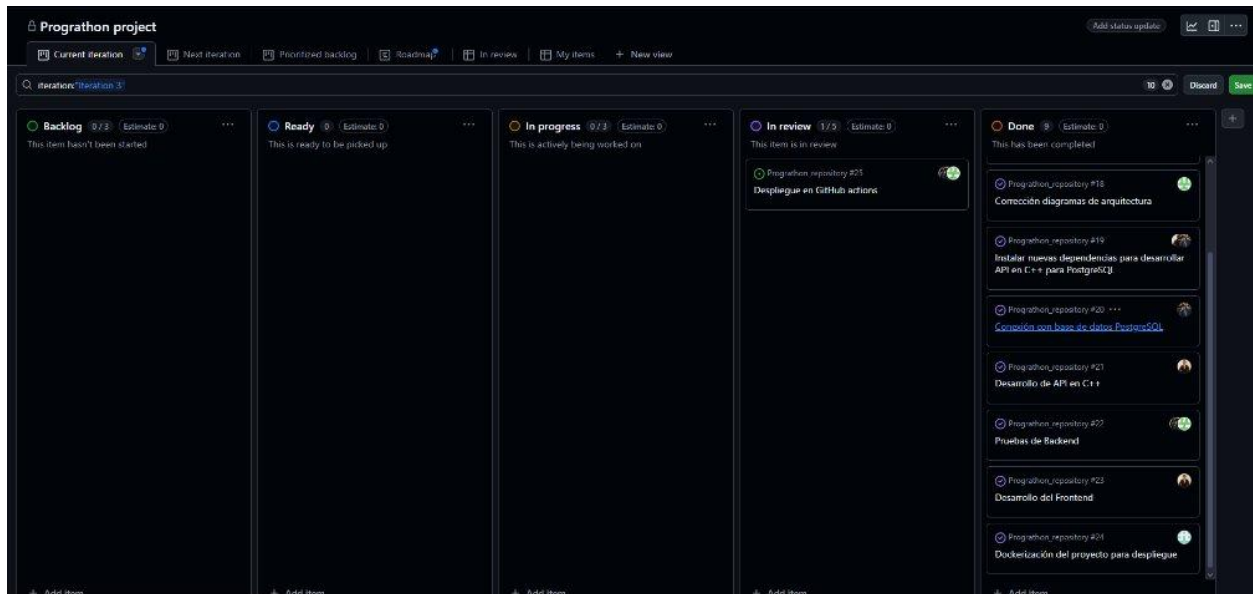


Ilustración 6 Burn up proyecto



Ilustración 7 Assignees issues



3. Metodologías

3.1. Explicación de la metodología

El Proceso Unificado de Rational (RUP) es la metodología ideal para desarrollar un sistema de maratones de programación, debido a su enfoque iterativo, gestión de riesgos proactiva y adaptabilidad a proyectos complejos.

RUP enfatiza la definición temprana de la visión y requisitos mediante casos de uso. Esto es crucial en un sistema educativo, donde deben priorizarse necesidades como la integración con currículas académicas y roles de usuarios (estudiantes, profesores).

RUP es la mejor opción porque:

- Maneja complejidad técnica mediante arquitectura temprana.
- Alinea iteraciones con valor educativo
- Mitiga riesgos proactivamente: Prototipos en Fase 2 evitan fallos algoritmos.
- Facilita adopción en entornos educativos: Despliegue gradual con beta controlado.

3.2. Fases

Fase 1:

- Objetivo: Alinear el sistema con el contexto educativo y definir el alcance integrado.
- Definir la visión del producto.

- Identificar los requisitos clave.
- Análisis de riesgo inicial.

Fase 2:

- Objetivo: Establecer la arquitectura escalable y validar la integración.
- Diseñar arquitectura base.
- Prototipar núcleos críticos.
- Pruebas de usabilidad.

Fase 3

Iteración 1: Sistema Central de Autenticación y Roles

Objetivo: Establecer la base segura de usuarios y permisos

Backend (C++):

- Implementación del sistema de registro/login usando JWT para tokens de autenticación
- Integración de Argon2 para hash seguro de contraseñas
- Creación de endpoints REST para gestión de usuarios (estudiantes, profesores, admin)
- Configuración de roles y permisos en la base de datos PostgreSQL

Frontend (React):

- Desarrollo de interfaces de registro e inicio de sesión
- Implementación de almacenamiento seguro de tokens en el navegador
- Mecanismo de redirección basado en roles después del login

Tecnologías clave: JWT, Argon2, libpq, React Router

Iteración 2: Gestión de Competencias e Inscripciones

Objetivo: Habilitar creación de maratones y sistema de inscripciones

Backend (C++):

- Desarrollo de endpoints para creación, edición y eliminación de maratones
- Implementación de lógica de inscripción de estudiantes a competencias
- Validación de permisos (solo profesores/admin pueden crear maratones)
- Optimización de consultas SQL para operaciones masivas

Frontend (React):

- Interfaz de creación de maratones con formularios controlados
- Panel de gestión de competencias para profesores
- Sistema de inscripción para estudiantes con listados disponibles

- Paneles diferenciados por roles (admin, profesor, estudiante)

Tecnologías clave: httpLib, PostgreSQL, Axios, Formik

Iteración 3: Automatización y Optimización Final

Objetivo: Implementar despliegue continuo y mejoras de rendimiento

Infraestructura:

- Configuración completa del pipeline CI/CD con GitHub Actions
- Automatización de compilación con CMake y script build.bat
- Gestión centralizada de dependencias mediante vcpkg
- Implementación de despliegues en entornos controlados

Optimizaciones:

- Conexiones persistentes a PostgreSQL para reducir sobrecarga
- Documentación técnica de APIs y componentes

Tecnologías clave: GitHub Actions, vcpkg, CMake, PostgreSQL

Ilustración 8 Fases del proyecto



3.3. Asignación De Roles

Ilustración 9 Roles del proyecto

Rol	Participante	Responsabilidades	Tecnologías	Entregables	Fases de participación
Arquitecta y líder técnica	Catalina	<ul style="list-style-type: none"> - Diseñar la arquitectura del sistema multicapa (frontend, backend, base de datos) - Definir y validar contratos de APIs - Definir y buenas prácticas - Coordinar revisiones técnicas y refactors 	Node.js, NestJS, PostgreSQL, Redis, GitHub.	<ul style="list-style-type: none"> - Diagrama de arquitectura - Especificaciones de APIs - Documentación técnica (API + Infraestructura) 	Fase 1 (planificación) Fase 2 (arquitectura) Fase 3 (supervisión técnica)
Frontend Developer	Jhonatan	<ul style="list-style-type: none"> - Desarrollar interfaces reactivas y accesibles - Implementar flujo de navegación, formularios, validaciones y manejo de estado - Conectar interfaces con backend vía - Asegurar el cumplimiento de principios de usabilidad y accesibilidad 	React.js / Next.js, TailwindCSS.	interfaces funcionales y Pruebas unitarias frontend	Fase 3 - Iteraciones 2 y 3
Backend Developer	Rodolfo	<ul style="list-style-type: none"> - Gestionar roles y permisos (Auth y RBAC) - Implementar lógica de negocio para maratones, competencias, recompensas y rankings - Modelar y normalizar la base de datos relacional - Implementar pruebas unitarias y de integración 	Node.js/NestJS, , PostgreSQL, Redis, JWT.	<ul style="list-style-type: none"> - APIs documentadas y seguras - Scripts de base de datos - Pruebas automatizadas backend 	Fase 3 - Iteraciones 1 y 4
DevOps Engineer, QA Funcional	Julian	<ul style="list-style-type: none"> - Configurar canal CI/CD (build, test, deploy) en GitHub Actions - Automatizar Pruebas funcionales y de rendimiento - Implementar monitoreo básico (logs, métricas) - Elaborar Documentación de despliegue para el equipo 	GitHub Actions	<ul style="list-style-type: none"> - Pipeline CI/CD funcional - Reportes de Pruebas funcionales 	Fase 2 (Infraestructura) Fase 3 (QA y Deploy)

3.4. Requerimientos

3.4.1. Requerimientos Funcionales.

Tabla 2 RF001

Identificación del requerimiento	RF001
Nombre:	Registro y Autenticación
Entradas:	Datos del usuario: nombre, apellido, edad, correo, contraseña, rol
Proceso:	El usuario se registra ingresando sus datos personales y rol. El sistema valida la información, encripta la contraseña y guarda el registro. Para iniciar sesión, se verifica la contraseña y se genera un token que permite el acceso seguro según el rol del usuario.
Salidas:	Usuario registrado exitosamente, acceso habilitado para iniciar sesión.
Descripción:	Los usuarios deben poder acceder de manera acertada a su inicio de sesión, dependiendo del rol asignado.
Prioridad:	Alta.
Categoría:	Registro y autenticación.

Tabla 3 RF002

Identificación del requerimiento	RF002
Nombre:	Gestión De Competencias.
Entradas:	Correo, contraseña.
Proceso:	El profesor autenticado accede a una interfaz donde puede crear, editar o eliminar competencias académicas. Estas acciones se reflejan directamente en la base de datos y quedan disponibles para los estudiantes.
Salidas:	Acceso permitido dependiendo su rol.
Descripción:	De acuerdo a las credenciales de acceso se permiten a los usuarios asignados las funciones del CRUD en el modulo de maratones.
Prioridad:	Alta.
Categoría:	Gestión de Maratones

Tabla 4 RF003

Identificación del requerimiento	RF003
Nombre:	Registro de avances en competencia.
Entradas:	Datos de la competencia: nombre, descripción, criterios, nivel de dificultad.
Proceso:	El estudiante visualiza las competencias disponibles y registra sus avances en cada una. El sistema guarda esta información y la muestra en su panel de control como parte de su progreso académico.
Salidas:	Confirmación de actualización.
Descripción:	Actualización de los avances en el perfil del estudiante.
Prioridad:	Media.
Categoría:	Gestión de Maratones.

Tabla 5 RF004

Identificación del requerimiento	RF004
Nombre:	Sistema de recompensas.
Entradas:	ID de la competencia, porcentaje o nivel alcanzado, comentario opcional.
Proceso:	El profesor configura una maratón asignando una competencia específica, su

	nivel o porcentaje requerido, y los retos correspondientes. Los estudiantes, al participar, resuelven los problemas planteados. El sistema evalúa automáticamente sus avances y actualiza el progreso en el perfil de cada estudiante en tiempo real.
Salidas:	Actualización del progreso reflejado en el perfil del estudiante.
Descripción:	Los profesores podrán asignar un premio a los estudiantes que muestren un buen desempeño en la competencia que el mismo haya creado.
Prioridad:	Media.
Categoría:	Gestión de Maratones.

Tabla 6 RF005

Identificación del requerimiento	RF005
Nombre:	Editar Configuración del Sistema.
Entradas:	Parámetros de configuración.
Proceso:	Validación y aplicación de cambios.
Salidas:	Confirmación de cambios.
Descripción:	Los administradores deben poder modificar configuraciones generales .
Prioridad:	Media.
Categoría:	Administración del Sistema.

Tabla 7 RF006

Identificación del requerimiento	RF006
Nombre:	Gamificación y Rankings.
Entradas:	Detección de avance significativo o competencia completada.
Proceso:	El sistema detecta avances relevantes o competencias completadas por los estudiantes. En respuesta, desbloquea logros, insignias o envía alertas de reconocimiento que se muestran en su perfil. Además, se actualiza el ranking general, incentivando la participación mediante elementos de gamificación.
Salidas:	Desbloqueo de logros, alertas o insignias en el perfil del estudiante.

Descripción:	Los estudiantes al lograr completar una maratón o lograr un avance realmente significativo, aumentará en el nivel de la maratón en comparación a los demás competidores, y se le reconocerá su esfuerzo mediante algún tipo de incentivo visual.
Prioridad:	Baja.
Categoría:	Gestion de Maratones.

3.4.1. Requerimientos No Funcionales.

3.4.1.1 Rendimiento:

Tabla 8 RNF001

Requerimiento	Descripción	Característica ISO 25010	Sub - Característica
PERF – 001 (respuesta <1 min)	El sistema debe responder a las solicitudes de usuario en menos de 1 minuto bajo condiciones normales	Eficiencia de rendimiento	Comportamiento temporal
PERF – 002 (100 usuarios concurrentes)	Debe soportar hasta 100 usuarios concurrentes	Eficiencia de rendimiento	Utilización de recursos
PERF – 003 (80% disponibilidad)	La disponibilidad del sistema debe ser del 80% durante horario hábil	Fiabilidad	Disponibilidad

Justificación:

- PERF-001/PERF-002 miden eficiencia temporal y uso de recursos bajo carga.

3.4.1.2 Mantenibilidad:

Tabla 9 RNF002

Requerimiento	Descripción	Característica ISO 25010	Sub - característica
MANT - 001 (código documento)	El código debe estar documentado	Mantenibilidad	Analizabilidad
MANT – 002 (Actualizaciones sin interrupción)	Debe permitir actualizaciones sin interrumpir el servicio	Mantenibilidad	Modificabilidad
MANT – 003	Debe generar logs de error detallados.	Mantenibilidad	Analizabilidad

(Logs de error detallados)			
----------------------------	--	--	--

Justificación:

- Documentación y logs mejoran analizabilidad (detección de fallos).
- Actualizaciones sin downtime requieren modularidad (componentes independientes).

3.4.1.3 Usabilidad:

Tabla 10 RNF003

Requerimiento	Descripción	Característica ISO 25010	Sub - característica
USA - 001	Usuarios sin formación técnica completarán tareas clave en ≤ 3 intentos	Usabilidad	Aprendibilidad
USA – 002	Debe proporcionar mensajes de error claros	Usabilidad	Tolerancia a errores
USA – 003	100% de las funciones críticas tendrán ayuda contextual visible	Usabilidad	Operabilidad

Justificación:

- Enfoque en usuarios no técnicos → aprendibilidad y operabilidad.
- Mensajes claros son parte de tolerancia a errores (ISO 25010).

3.4.1.4 Fiabilidad:

Tabla 11 RNF004

Requerimiento	Descripción	Característica ISO 25010	Sub - característica
FIAB-001	El sistema debe mantener operativas las funcionalidades esenciales durante fallos en componentes no críticos.	Fiabilidad	Tolerancia a fallos
FIAB-002	Garantizar integridad transaccional incluso tras interrupciones abruptas	Fiabilidad	Recuperabilidad

FIAB-003	Simulación mensual de escenarios catastróficos para validar mecanismos de recuperación.	Fiabilidad	Recuperabilidad
----------	---	------------	-----------------

Justificación:

- Aislamiento de servicios críticos (Bulkheads) → Mantiene funcionalidad esencial durante fallos.
- Circuit breakers → Evita propagación de errores.
- Transacciones Sagas → Garantizan integridad eventual.
- Mecanismos de compensación → Recuperación proactiva de transacciones.
- Simulacros con Chaos Engineering → Eliminan "puntos ciegos" de resiliencia.
- Validación de RTO/RPO → Mejora continua del diseño.

3.4.1.5 Seguridad:

Tabla 12 RNF005

Requerimiento	Descripción	Característica ISO 25010	Sub - característica
SEG-001	Garantizar que solo profesores autorizados puedan crear/modificar maratones. Evita que estudiantes hackeen el sistema para cambiar reglas o ver soluciones ajenas.	Seguridad	Autenticidad
SEG-002	Proteger soluciones de estudiantes y datos personales. Aunque hackers roben la base de datos, no podrán leer la información.	Seguridad	Confidencialidad
SEG-003	Saber quién, cuándo y qué hizo cada profesor. Permite auditar malas prácticas.	Seguridad	Rendición de cuentas

Justificación:

- Cifrado garantiza confidencialidad.
- Autenticación y logs aseguran autenticidad y no repudio.

3.4.1.6 Compatibilidad:

Tabla 13 RNF006

Requerimiento	Descripción	Característica ISO 25010	Sub - característica
COMP-001	Permitir que cada parte del sistema funcione de forma independiente. Así, si falla un componente, los demás siguen operando.	Fiabilidad	Tolerancia a fallos
COMP-002	Actualizaciones sin Interrupción: Agregar mejoras mientras los estudiantes usan activamente el sistema.	Mantenibilidad	Modificabilidad

Justificación:

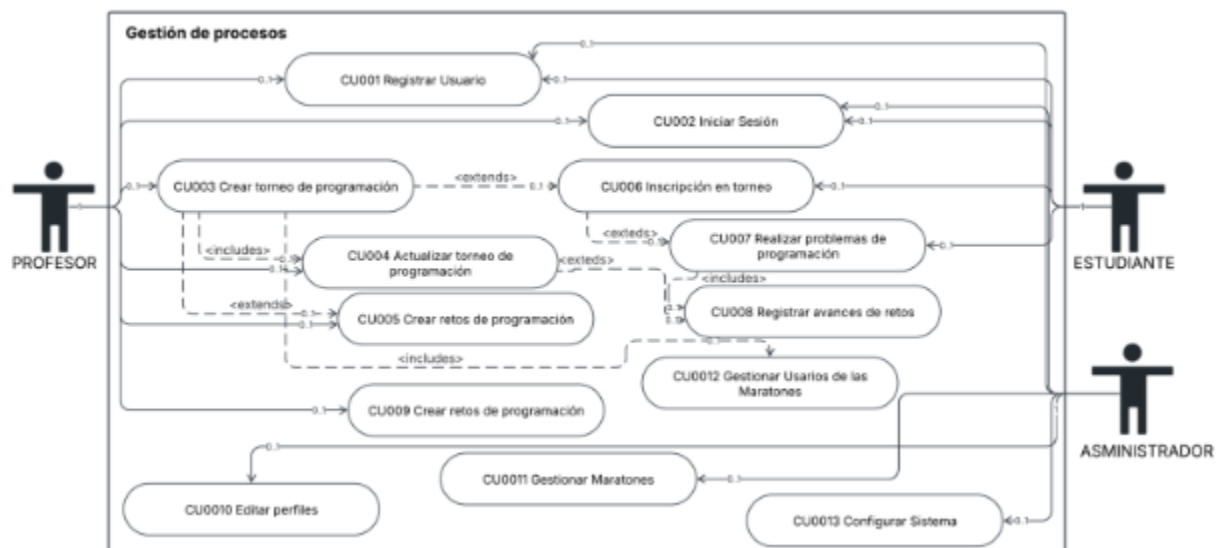
- Modularidad permite coexistencia con futuros componentes (interoperabilidad implícita).
- Actualizaciones sin downtime → compatibilidad con versiones anteriores

3.5. Arquitectura

Para la realización de este proyecto se manejará una arquitectura 4+1.

3.5.1. Casos De Uso.

Ilustración 10 Casos de uso



Nota: Muestra la interacción de los usuarios con el sistema, detallando las funcionalidades clave y relaciones entre casos de uso.

3.5.2. Diagrama De Secuencia.

Ilustración 11 Vista de procesos

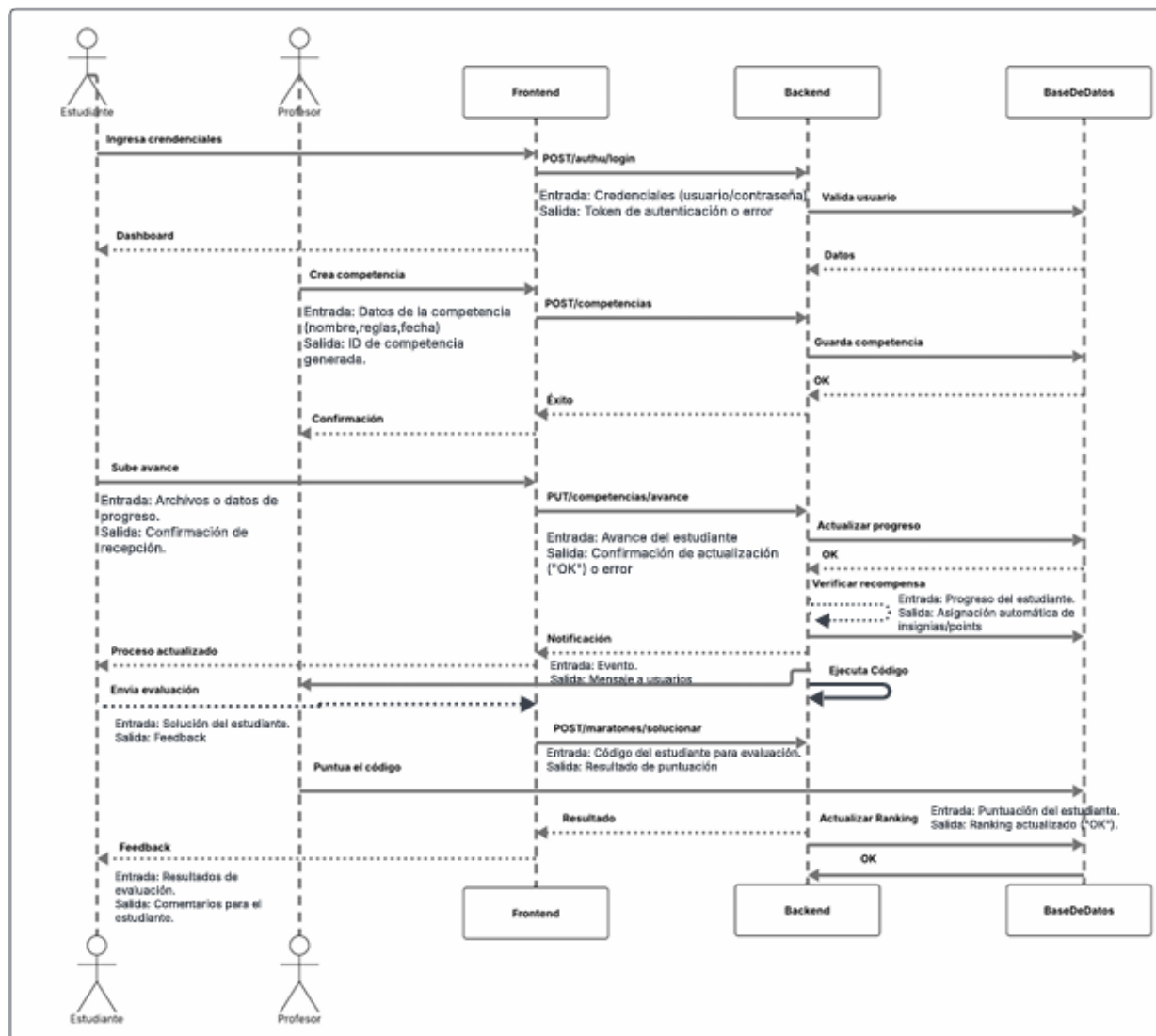
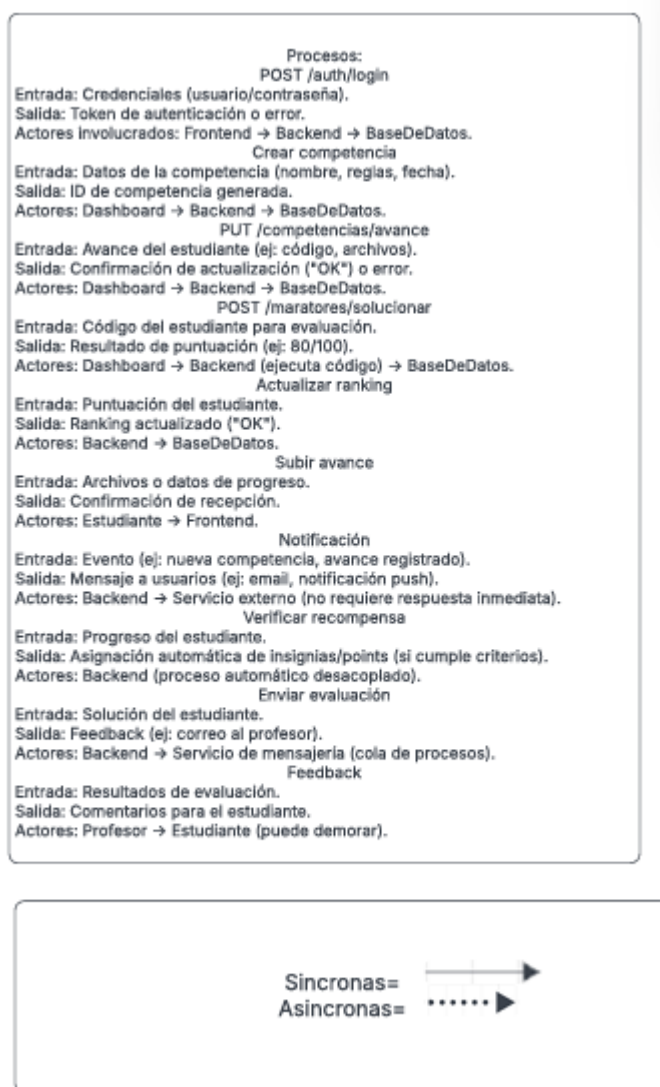


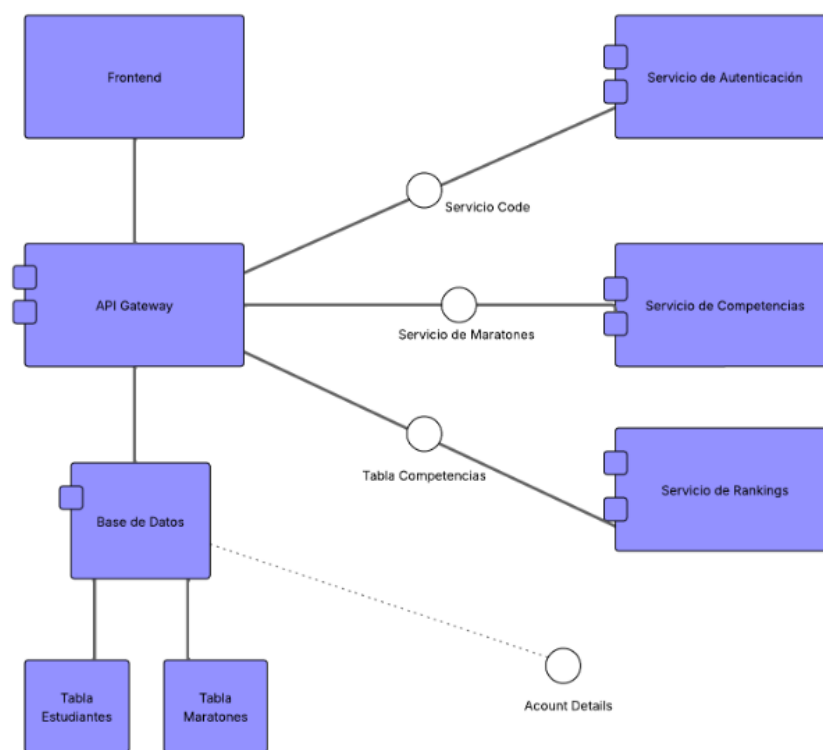
Ilustración 12 Procesos de diagrama



Nota: Representa el flujo de datos entre frontend, backend y la base de datos, destacando la ejecución de tareas clave como autenticación y evaluación de código.

3.5.3. Diagrama De Componentes.

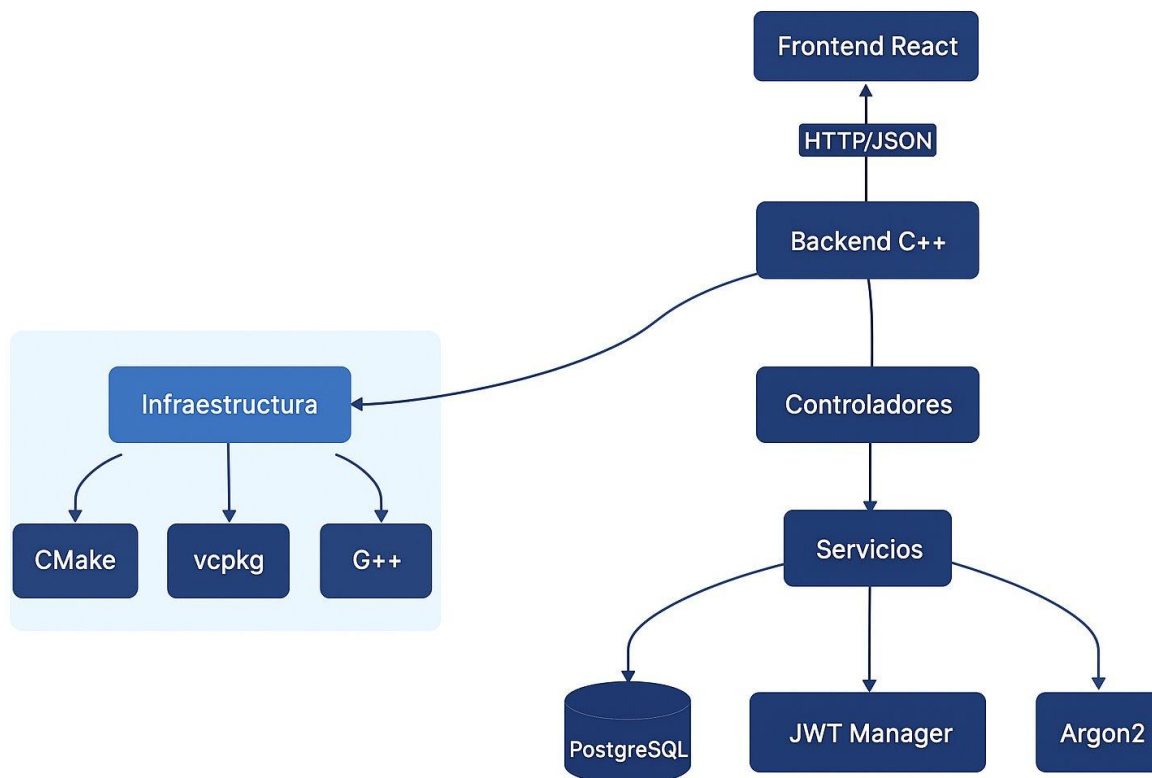
Ilustración 13 Componentes



Nota: Describe cómo se organizan los servicios dentro del API Gateway, separando módulos de autenticación, maratones y evaluación de código.

3.5.4. Diagrama De Vista De Despliegue.

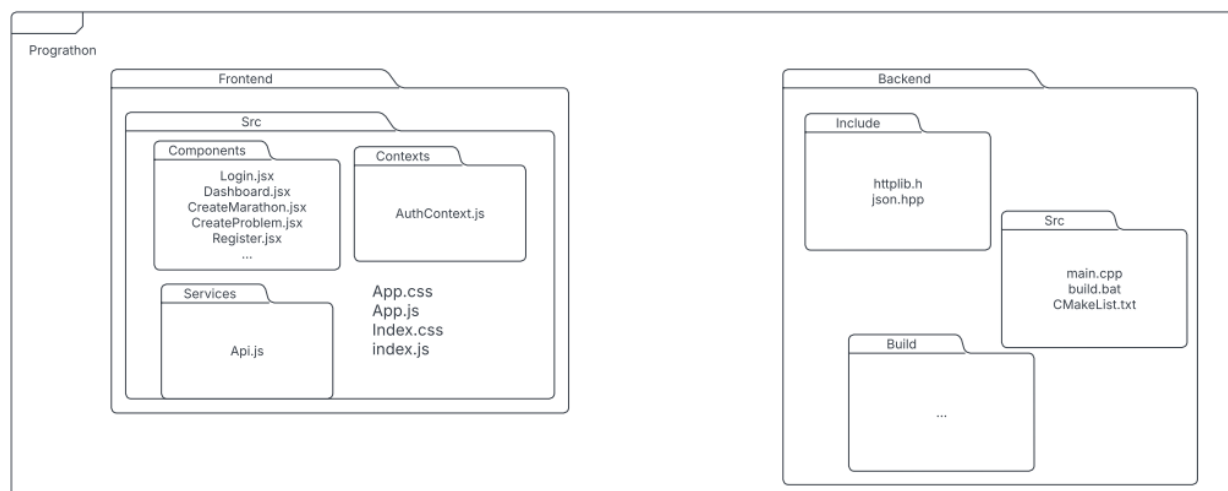
Ilustración 14 Vista de despliegue



Nota: Muestra la infraestructura del sistema en la nube, incluyendo la interacción entre frontend, backend, base de datos y servicios externos.

3.5.4. Diagrama De Paquetes.

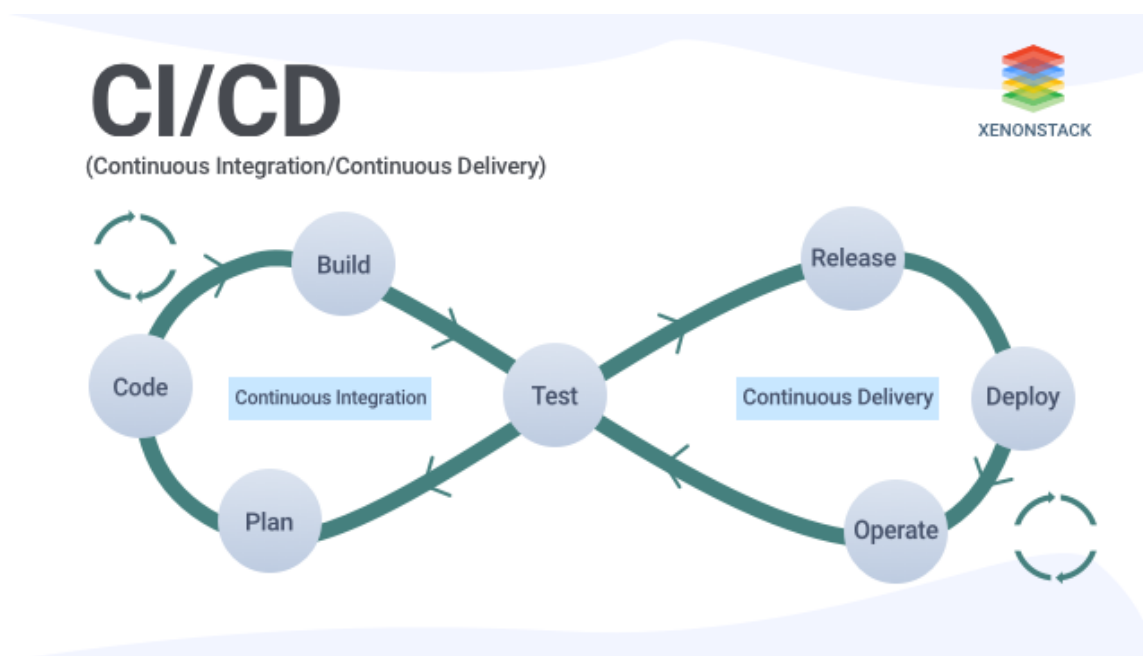
Ilustración 15 Paquetes



Nota: Este diagrama de paquetes representa la estructura modular del sistema Prograthon, destacando las dependencias entre frontend, backend, según los estándares de UML 2.0.

4. CI/CD – GitHub Actions + Docker

Ilustración 16 CI/CD



4.1. Integración continua

La Integración Continua (CI) es una práctica de ingeniería de software que automatiza la construcción, prueba y validación de cada cambio que se envía al control de versiones (Fowler, 2023; GitHub, 2025). Para ello, se configura un webhook o hook en el repositorio (GitHub, GitLab) que dispara un pipeline en un agente o “runner” dedicado cada vez que detecta un push o la apertura de un merge request (GitLab, 2025; Jenkins Project, 2025). Ese pipeline se define en un archivo de configuración donde se establecen las etapas y la secuencia de tareas, sean estas secuenciales o paralelas.

La primera etapa suele ser la build, donde el código se compila, se resuelven todas las dependencias y se generan los artefactos binarios (Circle Internet Services, 2025). A continuación, en la fase de prueba, se ejecutan automáticamente las suites de pruebas unitarias y, si está configurado, también tests de integración o de contrato que pueden valer de contenedores Docker o entornos simulados para levantar servicios auxiliares.

Paralelamente, se lleva a cabo un análisis estático de código con herramientas como ESLint, RuboCop o golangci-lint para garantizar consistencia de estilo y detectar errores potenciales sin ejecutar el programa. Además, se mide la cobertura de tests y, si existe un repositorio de artefactos, los binarios, informes de test e imágenes Docker generadas se almacenan para su posterior despliegue o auditoría (Microsoft, 2025).

Entre los beneficios técnicos destaca el feedback temprano y determinístico, pues al ejecutarse siempre en entornos controlados se asegura la reproducibilidad de la build; la escalabilidad y paralelismo, ya que runners distribuidos pueden procesar varias fases al mismo tiempo; y la **trazabilidad**, con variables de entorno que permiten identificar exactamente qué cambio generó cada ejecución y quién lo envió.

4.2. Despliegue continuo

La entrega continua (Continuous Delivery) y la implementación continua (Continuous Deployment), agrupadas a menudo bajo las siglas CD, son prácticas que extienden el flujo de trabajo de Integración Continua (CI) automatizando el empaquetado del software y su despliegue en entornos controlados (GitLab, 2025; Microsoft, 2025). Mientras CI se encarga de compilar el código y ejecutar pruebas en cada commit, CD añade las etapas necesarias para tomar los artefactos resultantes (por ejemplo, paquetes JAR, imágenes Docker o binarios) y trasladarlos de forma reproducible y versionada a distintos entornos, como testing, preproducción y producción.

En un pipeline de CD se incluye una fase de empaquetado en la que los artefactos generados por CI se envían a un repositorio de artefactos (como Artifactory, Nexus o un bucket en AWS S3) (Circle Internet Services, 2025). A partir de ahí, el pipeline define los entornos de despliegue y gestiona la promoción de versiones entre ellos, ya sea de manera automática (en Continuous Deployment) o tras una aprobación manual (en Continuous Delivery). Para ello se utilizan herramientas de orquestación e infraestructura como código, como Kubernetes (mediante Helm charts), Terraform o Ansible (Jenkins Project, 2025).

Cada etapa de despliegue en el pipeline suele consistir en scripts o tareas declaradas en YAML o en un DSL específico, que ejecutan comandos del tipo "kubectl apply", "helm upgrade --install" o "ansible-playbook". Estos pasos se aíslan en contenedores ligeros para asegurar que siempre se ejecutan en un entorno limpio y reproducible (GitHub, 2025). Además, se configuran puertas de control (gates) como comprobaciones de salud de los servicios, pruebas end-to-end o smoke tests. Si alguno de estos gates falla, el sistema puede revertir automáticamente al último despliegue exitoso usando versiones etiquetadas en el repositorio de artefactos o mediante control de versiones de la infraestructura.

La principal diferencia entre Continuous Delivery y Continuous Deployment radica en el grado de automatización de la última etapa. En Continuous Delivery el artefacto siempre está listo para desplegarse con un clic o tras una aprobación manual, mientras que en Continuous Deployment esa aprobación se elimina y cada build que supera todas las pruebas y gates se publica directamente en producción sin intervención humana. De este modo, CD convierte el lanzamiento de software en un proceso repetible, seguro y trazable reduciendo el tiempo de entrega al mercado y minimizando los riesgos asociados a los despliegues manuales (Fowler, 2023).

4.3. Docker

Docker es una plataforma de contenedorización que empaqueta una aplicación junto con todas sus dependencias en una unidad llamada imagen (Docker Inc., 2025a). Esa imagen se construye capa por capa y, cuando se ejecuta, se convierte en un contenedor, que no es más que un proceso aislado mediante namespaces y limitado con cgroups dentro del kernel de Linux (Open Container Initiative, 2024). De ese modo, cada contenedor tiene su propio sistema de archivos, su pila de red, sus identificadores de procesos y sus usuarios, pero se ejecuta sobre el mismo núcleo que el host; esto permite arrancar, detener y descartar entornos de forma casi inmediata sin gastos de virtualización completa (Docker Inc., n.d.-a).

En nuestro proyecto se utilizaron tres contenedores. El primero alberga el backend en C++; la imagen se genera en dos fases: la fase de build instala las bibliotecas de desarrollo, compila el código con CMake y luego, en una fase final más ligera, sólo se copia el binario resultante y las bibliotecas necesarias, exponiendo la variable PORT para que el servidor escuche en el puerto que asigna en tiempo de ejecución (Docker Inc., 2025b). El segundo contenedor contiene el frontend de React; se compila con Node en la fase de construcción y después se sirve con un Nginx mínimo en la fase final (Docker Inc., n.d.-b). El tercer contenedor, empleado sólo en desarrollo local, es la imagen oficial de PostgreSQL, configurada con un volumen persistente y con scripts SQL de inicialización colocados en la carpeta que el entrypoint de Postgres ejecuta automáticamente al primer arranque (Docker Inc., 2025c).

Durante el desarrollo local se usa un archivo docker-compose que describe estos tres servicios, define una red interna y establece dependencias para que el backend no arranque hasta que la base de datos haya superado su health-check (Docker Inc., 2025b). En producción, Render despliega cada componente por separado: crea una base PostgreSQL gestionada, construye o descarga la imagen del backend, la lanza en un servicio web y sirve los estáticos del frontend como sitio estático.

4.3.1. Archivos Dockerfile y Dockerk-compose.yml

A continuación, se presenta los Dockerfile de configuración del frontend, backend junto con el docker-compose.yml del proyecto.

Ilustración 17 Dockerfile Backend

```

1  # ----- BUILD STAGE -----
2  FROM debian:bookworm-slim AS build
3
4  # 1. Herramientas + dependencias básicas para compilar
5  RUN apt-get update && apt-get install -y \
6      build-essential cmake git \
7      libpq-dev libssl-dev libargon2-dev \
8      nlohmann-json3-dev && \
9      rm -rf /var/lib/apt/lists/*
10
11 WORKDIR /deps
12
13 # 2. Clona e instala jwt-cpp (solo headers + súper ligero)
14 RUN git clone --depth 1 https://github.com/Thalhammer/jwt-cpp.git && \
15     cmake -B jwt-cpp/build -S jwt-cpp \
16         -DJWT_BUILD_EXAMPLES=OFF \
17         -DJWT_BUILD_TESTS=OFF \
18         -DJWT_BUILD_CMAKEMODULE=ON && \
19     cmake --install jwt-cpp/build && \
20     rm -rf jwt-cpp
21
22 # 3. Copia tu código y compílalo
23 WORKDIR /app
24 COPY . /app
25 RUN cmake -B build -S . -DCMAKE_BUILD_TYPE=Release && \
26     cmake --build build -j$(nproc)
27
28 # ----- RUNTIME STAGE -----
29 FROM debian:bookworm-slim
30
31 RUN apt-get update && apt-get install -y \
32     libpq5 libssl3 libargon2-1 && \
33     rm -rf /var/lib/apt/lists/*
34
35 WORKDIR /app
36 COPY --from=build /app/build/server /usr/local/bin/server
37
38 EXPOSE 8080
39 CMD ["server"]
40

```

Ilustración 18 Dockerfile Frontend

```

dockerfile M X
frontend > dockerfile
1 # ----- fase de build -----
2 FROM node:20-alpine AS build
3 WORKDIR /app
4 COPY package*.json ./
5 RUN npm ci
6 COPY . .
7 RUN npm run build          # (React Scripts)
8 #   ó npm run build --if-present (si usas Vite)
9
10 # ----- fase de ejecución -----
11 FROM nginx:1.25-alpine
12 # Copiamos el build estático
13 COPY --from=build /app/build /usr/share/nginx/html
14 # Config SPA: cualquier ruta que no exista ⇒ index.html
15 RUN printf 'server {\n\
16     listen 80;\n\
17     root /usr/share/nginx/html;\n\
18     index index.html;\n\
19     try_files $uri $uri/ /index.html;\n\
20 };\n' > /etc/nginx/conf.d/default.conf
21
22 EXPOSE 80

```

Ilustración 19 Docker-compose.yml

```

docker-compose.yml M X
docker-compose.yml
1 # docker-compose.yml
2 # (se omite el campo 'version' para evitar la advertencia de Compose)
3
4 services:
5   db:
6     image: postgres:16
7     container_name: ppsc-db
8     environment:
9       POSTGRES_DB: contestdb
10      POSTGRES_USER: contest
11      POSTGRES_PASSWORD: supersecret    # + cámbiala a gusto
12     volumes:
13       - db_data:/var/lib/postgresql/data
14     healthcheck:
15       test: ["CMD-SHELL", "pg_isready -U $$POSTGRES_USER -d $$POSTGRES_DB"]
16       interval: 5s
17       timeout: 2s
18       retries: 5
19
20   backend:
21     build: ./backend                # Dockerfile en ./backend
22     container_name: ppsc-backend
23     depends_on:
24       db:
25         condition: service_healthy    # espera a que Postgres esté «san»
26     restart: on-failure              # se reinicia si falla al arrancar
27     environment:
28       DB_HOST: db
29       DB_PORT: 5432
30       DB_NAME: contestdb
31       DB_USER: contest
32       DB_PASS: supersecret          # + misma contraseña que arriba
33     ports:
34       - "8080:8080"
35
36   frontend:
37     build: ./frontend              # Dockerfile en ./frontend
38     container_name: ppsc-frontend
39     depends_on:
40       - backend
41     ports:
42       - "3000:80"
43
44   volumes:
45     db_data:
46

```

5. Costo / Beneficio

5.1. Tiempo Invertido VS Funcionalidades logradas

Durante las 4 semanas de desarrollo se logró implementar un conjunto sólido de funcionalidades básicas pero significativas:

- Semana 1: Diseño de la arquitectura del sistema, estructura de carpetas, configuración del backend en C++ y frontend en React.
- Semana 2: Creación de componentes clave del sistema (login, registro, creación de maratones, listado de problemas).
- Semana 3: Conexión entre frontend y backend, manejo de contexto para autenticación, validaciones y pruebas.
- Semana 4: Ajustes finales, pruebas de usabilidad, mejoras visuales y simulación de uso completo.

5.2. Beneficio técnico o comercial simulado

El sistema desarrollado aporta una solución práctica y escalable al problema de la brecha entre la formación académica y las exigencias del sector tecnológico en Colombia, al ofrecer un espacio interactivo para la práctica de programación competitiva desde el entorno universitario.

- **Entrenamiento aplicado y contextualizado:** El sistema permite a los profesores diseñar maratones personalizadas con problemas relevantes al nivel académico de los estudiantes, fomentando la aplicación de conocimientos en escenarios reales.
- **Evaluación continua y retroalimentación:** A través de las maratones, los estudiantes reciben una retroalimentación estructurada sobre su desempeño, facilitando una mejora constante y medible de sus habilidades técnicas.
- **Cierre de brechas prácticas:** En lugar de depender únicamente de contenidos teóricos, los estudiantes participan en desafíos reales que simulan entrevistas técnicas o competencias de la industria, contribuyendo al desarrollo de habilidades que hoy el 90% de los egresados no dominan (El Heraldo, 2021).

5.3. ¿Cómo escalaría o evolucionaría esta solución?

- Integrar sistemas como DOMjudge, HackerRank API o construir un microservicio de compilación segura (sandboxing).
- Permitir a los estudiantes recibir feedback inmediato sobre errores de compilación, lógica o eficiencia.
- Implementar dashboards para estudiantes y profesores, con métricas como:
 - Porcentaje de problemas resueltos
 - Tiempo promedio por solución
 - Curva de dificultad superada

- Permitir competencias interuniversitarias como entrenamientos previos a concursos tipo ICPC, o pruebas simuladas para entrevistas laborales.
- Integrar retos reales de empresas aliadas, generando una vinculación temprana con el entorno laboral.

6. Errores Y Soluciones

1. Migración forzada de pila MERN a C++ :

Problema: El proyecto originalmente fue concebido para funcionar con la pila MERN (MongoDB, Express, React y Node.js), lo que ofrecía una integración fluida entre frontend y backend. Sin embargo, ante la exigencia académica de implementar la API en C++, se generó un cambio abrupto de tecnología.

Solución: Se rediseñó completamente el backend utilizando httplib en C++ para las rutas HTTP y nlohmann::json para el manejo de datos. Se documentó la nueva arquitectura para el equipo.

2. Problemas con la descarga e integración de drivers de MongoDB

Problema: Se intentó conectar un microservicio previamente construido con MongoDB desde C++, pero la instalación de los drivers oficiales de MongoDB en C++ (mongocxx) presentó problemas de dependencias, compilación cruzada y versiones incompatibles.

Solución: Se decidió cambiar la base de datos a PostgreSQL, integrándola mediante libpq (cliente C++). Esto eliminó las dependencias problemáticas de MongoDB y unificó el backend.

3. Incompatibilidad de estructuras entre React y C++

Problema: La comunicación entre el frontend en React y el backend en C++ no fue inmediata, ya que:

- C++ requiere procesamiento manual de JSON con nlohmann::json
- Las rutas y respuestas deben ser programadas desde cero, sin middleware como Express

Solución: Se estandarizó el intercambio de datos en formato JSON y se diseñaron endpoints RESTful manualmente. Se incluyó un middleware para manejo de errores y estatus HTTP.

4. Curva de aprendizaje del equipo

Problema: La mayoría del equipo tenía mayor familiaridad con JavaScript y Node.js. El cambio a C++, un lenguaje más bajo nivel y con manejo manual de memoria, generó una curva de aprendizaje considerable.

Solución: Se organizó una división de tareas según fortalezas técnicas de los integrantes. Además, se compartieron tutoriales, plantillas, y se realizaron sesiones de revisión entre compañeros.

5. Configuración compleja del entorno de desarrollo







Problema: Integrar correctamente vcpkg, CMake, MinGW, librerías externas y scripts de build en Windows fue una tarea compleja y propensa a fallos.

Solución: Se automatizó el proceso de compilación mediante CMakeLists.txt y un script build.bat. Se definió una guía de instalación paso a paso para todo el equipo.

8. Objetivos Logrados

8.1. Requerimientos Funcionales logrados

Ilustración 20 Revisión de requerimientos cumplidos

#	Requerimineto	Estado	Observaciones	Fecha de finalización
1	RF001	Completado 	Funcionando correctamente	Wednesday, June 18, 2025
2	RF002	Completado 	Funcionando correctamente	Tuesday, June 17, 2025
3	RF003	En proceso 	Funcionando correctamente	
4	RF004	En proceso 	Se encuentra en proceso de desarrollo	
5	RF005	Completado 	Funcionando correctamente	Wednesday, July 9, 2025
6	RF006	En proceso 	Se encuentra en proceso de desarrollo	

8.2. Checklist

Ilustración 21 Checklist de calificación

	Descripción	Estado
1	La API está desarrollada en C++ utilizando un framework como Crow, Dragon o similar.	Hecho ✓
2	La API responde correctamente a peticiones GET y POST.	Hecho ✓
3	Al menos una ruta realiza operaciones CRUD sobre una base de datos.	Hecho ✓
4	El proyecto contiene un archivo CMakeLists.txt funcional.	Hecho ✓
5	Hay una estructura de carpetas clara: src/, include/, build/, etc.	Hecho ✓
6	El equipo ha redactado un informe técnico explicando: objetivos, herramientas, errores y soluciones.	Hecho ✓
7	Se ha descrito la metodología aplicada (xp, rup, lean, etc.).	Hecho ✓
8	Se incluye evidencia de los objetivos alcanzados y funcionalidades implementadas.	Hecho ✓
9	Existe una interfaz mínima que consume la API.	Hecho ✓
10	El frontend puede estar desarrollado en HTML/JS, React, u otro.	Hecho ✓

9. Conclusiones

- El proyecto desarrollado responde de manera concreta a la problemática actual de la brecha entre la formación académica en TI y las exigencias del sector tecnológico en Colombia. A través del sistema Prograthon, se brindó una herramienta funcional que promueve el entrenamiento práctico en programación, fortaleciendo las competencias técnicas que hoy carecen gran parte de los egresados en informática.
- Gracias a la adopción de la metodología RUP y la distribución clara de roles, el equipo logró avanzar en fases bien definidas, adaptándose a cambios y solucionando obstáculos técnicos durante el proceso. Esto permitió alcanzar los objetivos propuestos y culminar con una solución funcional en solo 4 semanas.
- La solución tiene un alto potencial de evolución futura: integración con jueces automáticos, visualización de estadísticas, gamificación, e incluso expansión como plataforma SaaS educativa. Esto podría posicionar a la Universidad Católica de Colombia como pionera en el uso de tecnología educativa orientada a cerrar la brecha laboral en el sector TI.

10. Bibliografía

- Diego Oliveros. (2025). Desarrollo de software. Universidad Católica de Colombia
- Beck, K. et al. (2001). Manifiesto para el desarrollo ágil de software. Agile Alliance.
- Chacon, S. & Straub, B. (2014). Pro Git (2a ed.). Apress.
- Elmasri, R. & Navathe, S. B. (2016). Fundamentals of database systems (7a ed.). Pearson.
- Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. University of California.
- Flanagan, D. (2011). JavaScript: The definitive guide (6a ed.). O'Reilly Media.
- Gamma, E. et al. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.
- Graham, D. & Fewster, M. (2012). Experiences of test automation. Addison-Wesley.
- IEEE Computer Society. (1990). IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). IEEE.
- Knuth, D. E. (1997). The art of computer programming: Fundamental algorithms (3a ed., Vol. 1). Addison-Wesley.
- Larman, C. (2004). Agile and iterative development: A manager's guide. Addison-Wesley.
- Loeliger, J. & McCullough, M. (2012). Version control with Git (2a ed.). O'Reilly Media.
- McConnell, S. (2004). Code complete: A practical handbook of software construction (2a ed.). Microsoft Press.
- Myers, G. J. et al. (2012). The art of software testing (3a ed.). Wiley.
- Norman, D. A. (1988). The psychology of everyday things. Basic Books.
- Norman, D. A. (2013). The design of everyday things (Ed. revisada). Basic Books.
- Richardson, L. & Ruby, S. (2007). RESTful web services. O'Reilly Media.
- Schwaber, K. (2004). Agile project management with Scrum. Microsoft Press.
- Schwaber, K. & Sutherland, J. (2020). La guía de Scrum. Scrum Guides.
- Stroustrup, B. (2013). The C++ programming language (4a ed.). Addison Wesley.
- Zeller, A. (2009). Why programs fail: A guide to systematic debugging. Morgan Kaufmann.
- Gutiérrez, E., & Vargas Riaño, D. A. (2023, 30 de septiembre). Colombia, el tercer país de la región donde más aumentó el número de programadores. El Colombiano.
- <https://www.elcolombiano.com/negocios/colombia-tercer-pais-de-america-latina-dondemas-nuevos-programadores-hay-2023-FF22500063>.
- Clavijo, M. A. (2024, 2 de febrero). Mercado laboral de talento TI en Colombia. DB SYSTEM. <https://www.db-system.com/mercado-laboral-de-talento-ti-en-colombia/>
- Redacción ELHERALDO.CO. (2021, 3 de septiembre). La covid-19 agudizó la demanda de talento TI en Colombia. El Herald.
- <https://www.elheraldo.co/tecnologia/2021/09/03/la-covid-19-agudizo-la-demanda-detalento-ti-en-colombia/>

- Express.js. (2024). Express.js API documentation. <https://expressjs.com/en/api.html>
- The Times of India. (2025, 7 de julio). CA results: Many from Raj in top 50. Recuperado de <https://timesofindia.indiatimes.com/city/jaipur/ca-results-many-from-raj-in-top-50/articleshow/122284283.cms>
- Dice.com. (2023, 31 de agosto). Turning a failed tech interview into a learning opportunity. <https://www.dice.com/career-advice/turning-a-failed-tech-interview-into-a-learning-opportunity>
- KDnuggets. (2023, 31 de mayo). How hard is it to get into FAANG companies? <https://www.kdnuggets.com/2023/05/hard-get-faang-companies.html>
- TNN. (2025, 7 de julio). CA results: Many from Raj in top 50. The Times of India. <https://timesofindia.indiatimes.com/city/jaipur/ca-results-many-from-raj-in-top-50/articleshow/122284283.cms>
- Auth0. (n.d.). What is a JSON Web Token? Retrieved from [Insert relevant Auth0 URL, e.g., <https://auth0.com/docs/secure/tokens/json-web-tokens>]
- Biryukov, A., Dinu, D., & Khovratovich, D. (2016). Argon2: The Memory-Hard Function for Password Hashing.
- Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. [Doctoral dissertation, University of California, Irvine]. Retrieved from [Insert URL if available, e.g., <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>]
- Nlohmann. (n.d.). JSON for Modern C++. [Insert relevant GitHub URL, e.g., <https://github.com/nlohmann/json>]
- PostgreSQL Global Development Group. (n.d.). libpq — C Library. [Insert relevant PostgreSQL documentation URL, e.g., <https://www.postgresql.org/docs/current/libpq.html>]
- Yamaoka, Y. (n.d.). cpp-httplib. [Insert relevant GitHub URL, e.g., <https://github.com/yhirose/cpp-httplib>]
- MacLean, A. (2023, agosto 15). Building RESTful Interfaces in C++ With nlohmann and cpp-httplib [Video]. YouTube. <https://www.youtube.com/watch?v=uqPTzUdNLZk>
- MaestrosWeb. (s. f.). Implementando Seguridad en MEAN/MERN: Guía Práctica. Recuperado de <https://maestrosweb.puntanetwork.com/full-stack-development/seguridad-meanmern-implementando-autenticacion-autorizacion/>
- PostgreSQL Global Development Group. (s. f.). PostgreSQL: Powerful open source relational database. Recuperado de <https://www.postgresql.org/about/>
- PostgreSQL Documentation. (2025). Chapter 32: libpq — C Library. En PostgreSQL 17 Documentation. Recuperado de <https://www.postgresql.org/docs/current/libpq.html>
- Microsoft. (2024, agosto 8). vcpkg overview [Documentación]. Microsoft Learn. Recuperado de https://learn.microsoft.com/en-us/vcpkg/get_started/overview
- Wikipedia. (2025). Vcpkg [Artículo]. En Wikipedia. Recuperado de <https://en.wikipedia.org/wiki/Vcpkg>.

- The CMake Team. (2025). CMake: Cross-platform build tool [Software documentation]. CMake.org. Recuperado de <https://cmake.org/>
- MinGW-w64 Project. (2025, junio 4). MinGW-w64: GCC for native Windows [Software documentation]. MinGW-w64.org. Recuperado de <https://www.mingw-w64.org/>
- React Core Team. (s. f.). React – A JavaScript library for building user interfaces [Documentación]. legacy.reactjs.org. Recuperado de <https://legacy.reactjs.org/>
- GeeksforGeeks. (2025, febrero 27). What is react-router-dom? En GeeksforGeeks. Recuperado de <https://www.geeksforgeeks.org/reactjs/what-is-react-router-dom/>
- JWT.io. (s. f.). Introduction to JSON Web Tokens. Recuperado de <https://jwt.io/introduction/> (jwt.io)
- Argon2 Documentation. (2025). Argon2 – Password hashing function [Read the Docs]. Recuperado de <https://argon2-cffi.readthedocs.io/> (argon2-cffi.readthedocs.io)
- Docker Inc. (2025, enero 15). *Docker Engine – Getting Started* [Documentación]. docs.docker.com. Recuperado de <https://docs.docker.com/get-started/>
- Docker Inc. (s. f.). *Dockerfile reference* [Software documentation]. docs.docker.com. Recuperado de <https://docs.docker.com/engine/reference/builder/>
- Docker Inc. (2025, marzo 22). *Compose file version 3 reference* [Documentación]. docs.docker.com. Recuperado de <https://docs.docker.com/compose/compose-file/compose-file-v3/>
- Docker Inc. (s. f.). *Docker Hub overview* [Documentación]. docs.docker.com. Recuperado de <https://docs.docker.com/docker-hub/>
- Open Container Initiative. (2024, diciembre 12). *OCI Runtime Specification 1.1.0* [Especificación técnica]. opencontainers.org. Recuperado de <https://github.com/opencontainers/runtime-spec>
- Docker Inc. (2025, abril 10). *Docker networking overview* [Documentación]. docs.docker.com. Recuperado de <https://docs.docker.com/network/>
- Fowler, M. (2023, mayo 9). *Continuous Integration* [Artículo técnico]. martinowler.com. Recuperado de <https://martinfowler.com/articles/continuousIntegration.html>
- GitHub. (2025, febrero 5). *GitHub Actions – Documentation* [Documentación]. docs.github.com. Recuperado de <https://docs.github.com/actions>
- GitLab. (2025, abril 14). *GitLab CI/CD pipelines* [Documentación]. docs.gitlab.com. Recuperado de <https://docs.gitlab.com/ee/ci/>
- Jenkins Project. (2025). *Jenkins User Handbook – Pipeline as Code* [Manual]. jenkins.io. Recuperado de <https://www.jenkins.io/doc/book/pipeline/>
- Circle Internet Services, Inc. (2025, enero 12). *CircleCI Concepts – Configuration reference* [Documentación]. circleci.com. Recuperado de <https://circleci.com/docs/configuration-reference/>

- Microsoft. (2025, marzo 30). *Build, Test, and Deploy with Azure Pipelines* [Documentación]. learn.microsoft.com. Recuperado de <https://learn.microsoft.com/azure/devops/pipelines/>