INFORMACIÓN DEL DOCUMENTO

Solicitado por:	Dirección de Fábrica-escuela		
Preparado por:	Robinson Coronado Catalina Céspedes	**Revisado por:**	Diego Botia Wilmer Gil
Versión del documento:	V1	**Fecha del documento:**	12/02/2024
Versión del documento:	V2	**Fecha del documento:**	27/07/2024
Versión del documento:	V3	**Fecha del documento:**	28/04/2025
Versión del documento:	V4	**Fecha del documento:**	27/08/2025

HISTORIAL DE VERSIONES

Versión	Objetivo de la actualización	Fecha del documento	Autor
V1	Revisión y ajustes respecto de la edición de la fábrica-escuela 2023-2	12/02/2024	Robinson Coronado Catalina Céspedes
V2	Revisión y ajuste para el semestre 2024-2	27/07/2024 03/08/2024 13/09/2024	Robinson Coronado Juan Pablo Arango Catalina M. Céspedes T
V3	Revisión y ajuste para el semestre 2025-1	28/07/2025	Diego Botia Gustavo Palermo
V4	Revisión y ajuste para el semestre 2025-2	27/08/2025	Diego Botia Wilmer Gil

Stack Tecnológico definido

El stack tecnológico completo de la aplicación que incluye Docker, Spring Boot 3, Render o AWS, , GitHub Actions, PostgresQL (Supabase), React (NextJS), Grafana / Promethus, Sonar Cloud, Kubernetes, WebSockets entre otras herramientas y se estructura de la siguiente manera:

Backend:

Spring Boot:
Descripción: Framework de desarrollo de aplicaciones Java que simplifica el desarrollo de aplicaciones basadas en Java.
Uso: Desarrollo del backend de la aplicación, servicios web con GraphQL, y manejo de lógica de negocio.

PostgreSQL:
Descripción: Sistema de gestión de bases de datos relacional.
Uso: Almacenamiento de datos críticos de la aplicación, como información de usuarios, reservas y configuraciones.

Docker:
Descripción: Plataforma de contenedores que facilita la implementación y administración de aplicaciones.
Uso: Contenerización de la aplicación y servicios, facilitando la portabilidad y escalabilidad.

Supabase
Descripción: Plataforma de backend como servicio (BaaS) de código abierto que proporciona base de datos (PostgreSQL), autenticación, almacenamiento y funciones serverless listas para usar.
Uso principal: Crear de forma rápida un backend completo para aplicaciones web o móviles sin tener que programar desde cero toda la infraestructura.

Render
Descripción: Plataforma de hosting en la nube que facilita el despliegue de aplicaciones web, APIs, bases de datos y servicios estáticos de manera automática y continua.
Uso principal: Deploy rápido de aplicaciones (por ejemplo, backend en Node.js, Spring Boot, bases de datos PostgreSQL) sin necesidad de administrar servidores manualmente.

Prometheus
Descripción: Sistema de monitorización y base de datos de series temporales (time series) diseñado para capturar métricas numéricas de aplicaciones y sistemas.
Uso principal: Recolectar métricas (CPU, memoria, tiempos de respuesta de APIs, etc.) y alertar sobre comportamientos anómalos en tiempo real.

Grafana
Descripción: Plataforma de visualización de datos que permite construir dashboards dinámicos a partir de distintas fuentes (Prometheus, bases de datos, etc.).
Uso principal: Visualizar y analizar gráficamente las métricas recolectadas (por ejemplo, rendimiento del sistema, tráfico de red, errores de APIs).

Frontend:

Next JS:

Descripción: Framework de React que permite construir interfaces de usuario interactivas.

Uso: Desarrollo del frontend de la aplicación, creación de componentes de usuario y gestión del estado de la interfaz.

Figma con ShadCN

Descripción: Herramienta de diseño colaborativo en línea que permite crear prototipos y mockups de interfaces. ShadCN es una librería de componentes predefinidos que facilita mantener consistencia visual y buenas prácticas de diseño.

Uso: Creación de prototipos y maquetas de la aplicación, definiendo la experiencia de usuario y validando la estructura visual antes de la implementación. Permite que los equipos discutan, ajusten y aprueben el diseño de manera colaborativa, asegurando alineación entre analistas, diseñadores y desarrolladores.

Lovable o V0

Descripción: Plataforma de desarrollo asistido que genera código HTML y CSS limpio a partir de diseños, facilitando la implementación de interfaces web responsivas y modernas.

Uso: Traducción de los mockups diseñados en Figma hacia código real de frontend. Se utiliza para obtener la base del HTML y CSS de los componentes de la aplicación, optimizando tiempos de desarrollo y reduciendo errores en la implementación del diseño.

Vercel

Descripción: Plataforma en la nube especializada en el despliegue de aplicaciones frontend, optimizada para proyectos construidos con frameworks modernos como React y NextJS. Ofrece integración continua, escalabilidad automática y tiempos de carga rápidos mediante CDN distribuido.

Uso: Despliegue y publicación de la aplicación web. Permite que los equipos suban el frontend desde repositorios de GitHub o GitLab, generando entornos de prueba y producción. Facilita compartir versiones previas con los equipos de diseño y QA para validación, asegurando entregas continuas y eficientes en los Sprints.

Comunicación en Tiempo Real:

WebSockets:

Descripción: Protocolo de comunicación bidireccional que permite una comunicación en tiempo real entre el cliente y el servidor.

Uso: Implementación de funcionalidades en tiempo real, como notificaciones instantáneas o actualizaciones en vivo.

Contenedores y Orquestación:

Docker Compose:

Descripción: Herramienta para definir y ejecutar aplicaciones Docker en entornos multi-contenedor.

Uso: Orquestación de múltiples contenedores que componen la aplicación (backend, frontend, base de datos).

Herramientas Adicionales:

Nginx (Opcional):

Descripción: Servidor web de código abierto que puede actuar como proxy inverso y equilibrador de carga.

Uso: Manejo de tráfico web, equilibrio de carga y redirección a servicios backend.

Desarrollo y Gestión de la configuración (versionamiento):

Git (local):

Descripción: Herramienta de comandos para sincronización.

Uso: Sincronización de fuentes desde el local con el repo en la nube.

GitHub (nube):
Descripción: Repositorio de fuentes distribuido en nube
Uso: Control de versiones del código fuente de la aplicación.

Maven:
Descripción: Herramienta de construcción de software y administración de librerías externas.
Uso: Gestión de dependencias y construcción de proyectos en Java.

Gradle:
Descripción: Herramientas de construcción de software y gestión de librerías de terceros. Se utiliza para compilar el proyecto y lanzar las pruebas unitarias y/o automatizadas
Uso: Gestión de dependencias y construcción de proyectos en Java (repo del producto y/o repo de automatización)

IntelliJ IDEA:
Descripción: Entornos de desarrollo integrados (IDE) para el desarrollo de aplicaciones Java y React.
Uso: Desarrollo y depuración de código.

Gestión del Backlog:

Azure DevOps:

Descripción: Servicio en nube Azure que permite gestionar los proyectos de software bajo prácticas DevOps y ágiles.

Uso: Creación y gestión del producto backlog; Gestión de sprint y seguimiento a métricas ágiles.

Herramientas de CI/CD y Análisis Estático:

GitHub Actions:

Descripción: Plataforma de CI/CD integrada directamente en GitHub que permite la automatización y validación de flujos de trabajo.

Uso:

- Configuración mediante archivos YMAL
- Ejecución de pipelines de CI/CD.
- Integración con otras herramientas y servicios.

SonarCloud:

Descripción: Plataforma en la nube para análisis estático de código que proporciona informes detallados sobre la calidad del código.

Uso:

- Identificación de problemas de código, vulnerabilidades y malas prácticas.
- Métricas de calidad y cobertura de pruebas unitarias
- Identificación de la Deuda Técnica.
- Integración con GitHub para evaluar la calidad del código en cada confirmación.

SonarQube:

Descripción: Plataforma de código abierto para inspección continua de calidad de código a través de diferentes herramientas de análisis estático de código fuente.

Uso:

- Proporciona métricas para mejorar la calidad del código.
- Herramienta para la fase del testing y auditoría del código
- Integración e implementación para la calidad continua del código

SonarLint:

Descripción: Herramienta software para análisis estático de código usada a nivel local para usar en IDE y que detecta errores mientras escribe código.

Uso:

- Análisis de precisión sobre la marcha.
- Métricas de calidad y cobertura de pruebas unitarias

Herramientas de desarrollo seguro:

GitLab:

Descripción: Plataforma de CI/CD integrada que permite la automatización de flujos de trabajo y permite importar repositorios de GitHub para la gestión del mismo.

Uso:

- Configuración y ejecución de pipelines de CI/CD en pruebas de seguridad (SAST Y DAST)
- Integración con otras herramientas.

Snyk:

Descripción: es una plataforma que permite la búsqueda, priorización y corrección de vulnerabilidades en código, dependencias y contenedores, aplicada en flujos CI/CD para mejorar la seguridad de las soluciones.

Uso:

- Identificación de vulnerabilidades en código, dependencias y contenedores.
- Integración con GitLab para la gestión de vulnerabilidades de seguridad.

Owasp Zap:

Descripción: es una herramienta de prueba de penetración articulado con Open Web Application Security Project (OWASP). Está diseñado específicamente para probar aplicaciones web.

Uso:

- Identificación de vulnerabilidades en etapa de pruebas y despliegue.
- Herramienta integrada con GitLab para pruebas dinámicas.

Este stack tecnológico proporciona una base sólida para el desarrollo de una aplicación moderna, escalable y eficiente que abarca desde la capa de almacenamiento de datos hasta la interfaz de usuario, aprovechando tecnologías populares en el desarrollo de aplicaciones empresariales

Otros Aspectos Técnicos

Comunicación Síncrona y Asíncrona:

- Los servicios se comunican según sea el caso entre sí a través de API REST o GraphQL o mensajes asincrónicos (p. ej., RabbitMQ)

Persistencia de Datos:

- Bases de datos basado en PostgreSQL.

Lineamientos DevOps

Descripción

- Se construirá un pipeline CI/CD en GitHub Actions para generar los artefactos que serán desplegados tanto del producto software como el de automatización
- Los despliegues del FrontEnd serán en Vercel en un bucket S3 con CloudFront. El FrontEnd se desplegará en archivos estáticos.
- El pipeline de GitHub Actions será configurado para que se comunique directamente al bucket S3 y suba los archivos directamente al bucket y haga la validación en CloudFront.
- El backEnd se desplegará en Cloud (Plataforma de nube definido dentro del curso Cloud Computing.)
- La Base de datos se desplegará en el servicio de Supabase para cada reto.
- Se Configurará los Quality Gate a través de SONAR al igual el cálculo de la deuda técnica para la medición de calidad en el proceso de compilación y despliegue (creación de los artefactos)
- Se establecerá un porcentaje de cobertura para pruebas unitarias en el código Frontend y Backend de al menos un 40%.
- El pipeline de CI/CD **romperá** si no cumple el porcentaje mínimo establecido
- Se establecerá el análisis de código estático para determinar la deuda técnica. Para éste se utilizará SonarQube, Sonar Cloud, en local SonarLint. La deuda técnica debe tener mínimo un plazo de 3 días o un valor inferior.
- Se deben detectar las vulnerabilidades de seguridad y proponer posibles soluciones.
- Para el despliegue se crea un stage independiente de E2E para validar las pruebas funcionales automatizadas de acuerdo al cumplimiento del plan de pruebas.
- En caso de no poder crear el Stage E2E de pruebas automatizadas se podrá ejecutar las pruebas automatizadas a nivel local para dar cumplimento con las funcionalidades estipuladas para la herramienta software que se está construyendo

Manejo de Repositorio:

- El repositorio se propone en GitHub con metodología **trunk based** para ir fusionando los estudiantes pequeñas actualizaciones de código de forma frecuente en un “tronco” o rama principal de cada uno de los retos a desarrollar.
- Se define el manejo de un solo repo perteneciente a cada proyecto y por modalidad (uno para cada reto en el presencial y otro para el virtual).
- Cada equipo creará su propio repositorio en GitHub (por favor usar cuentas con dominio institucional) y deberá compartir con los profesores de cada curso el repositorio respectivo a su reto.
- El rol de desarrollador Backend creará el repo del Backend, su estructura y dará los permisos.
- El rol de desarrollador Frontend creará el repo del Frontend, su estructura y dará los permisos.
- El nombramiento de las carpetas en ambos casos debe hacer referencia al módulo a desarrollar.

Nombramiento de variables, métodos y clases

- El lenguaje definido para construcción del código es inglés.
- Se propone para el nombramiento de variables el estándar **CamelCase** para palabras compuestas al interior del desarrollo. Por ejemplo, para Java debemos usar **Upper Camel Case** en el nombre de las clases y **Lower Camel Case** en los nombres de las variables y los métodos.
- Usar nombres claros y coherentes para las variables y los métodos. No usar contracciones.

Ejemplo:
class SoyUnaClase{ }
int soyUnaVariables=0;
public void soyUnMetodo(){ }

Gestión de la configuración

- Se tendrá un único repo en GitHub por cada reto con la estrategia **Trunk-Base Development** para administrar los **branch** y los **release** más eficientemente.
- Se creará una rama principal y única llamada **trunk o main en GitHub** y en la cual todo el equipo colabora e integra directamente sus porciones de código desarrollada mediante un **push. Aplica tanto para frontend como para Back (dos repos con propósito diferente).**
- Para alinear el frontend se recomienda colocar todos los mockups y elementos por medio de **Figma** en un solo sitio o lugar para que todos los módulos tengan acceso y tengan uniformidad a la hora de proponer sus interfaces, objetos, componentes etc. Es posible generar las vistas a partir de herramientas Low Code con I.A si se considera necesario.
- Cuando un integrante del equipo inicia el desarrollo o va a agregar nuevo código al proyecto debe realizar un **pull para** traer lo nuevo que exista en el **trunk** al repo local. Y si es la primera vez deberá realizar una copia del repo que está en nube mediante el comando **clone.**
- **Al finalizar la jornada de desarrollo es ideal realizar un push** de lo nuevo para integrarlo al trunk (rama principal) con un **commit** y con un comentario sobre lo que se hizo nuevo o modificó.
- A nivel local los desarrolladores deben crear un **branch personalizado del trunk** actualizado que tengan en sus máquinas para evitar hacer cambios directamente a la rama principal del proyecto en nube (**trunk**) mediante el comando: git checkout –b "**feature/#HUDescripcionHU**"
- La forma de crear los **branch** en el local será nombrado: **feature/#HUDescripcionHU.** Esto es, el número la HU de usuario que están trabajando
- Antes de integrar o mezclar cualquier código a la rama trunk se debe hacer un **pull request** para que un grupo o una persona apruebe los cambios con una revisión previa (**pull request**)

Para garantizar que no se suba errores a la rama principal llamada **trunk**, todo el código que pretendemos mezclar de las diferentes ramas personales de los desarrolladores debe pasar por un pipeline de CI/CD que permita generar un compilado, build, y pruebas exitoso y así evitar mezclar errores en el **trunk** y que se cumpla los diferentes **Quality Gate** en conjunto con la **cobertura de pruebas unitarias** que se dispongan para tal propósito. Dichos Quality Gate y límites serán definidos por el equipo de calidad para todo el proyecto de fábrica escuela. **Una vez pasado el pipeline** CI/CD, se hace la aprobación en el pull request del nuevo código que se pretende subir a la rama trunk.

El equipo acordará quien o quienes realizarán las aprobaciones mediante un Pull Request para mezclar el código desde las ramas personales hacia la rama trunk (Sugerencia: Aprobadores de pull request por módulo).

Aclaración del modelo para evitar subir errores en la rama trunk

Una vez es exitoso el pipeline de CI/CD del nuevo código que se va a subir validando cobertura de pruebas unitaria y build, se solicita aprobación mediante un **pull request** a un integrante del equipo que tenga el rol de aprobación.

El nuevo código mezclado deberá ser compilado nuevamente en la rama definitiva del **trunk** para verificar que la nueva pieza de código no tiene ningún problema de compilación sobre el proyecto definitivo y que los demás miembros puedan bajar un código en perfecto estado para continuar con el desarrollo del producto.