**DOCUMENTACION INTEGRACION**

**JUAN DAVID SERRANO**

**MIGUEL ANGEL ROMERO**

**LAURA GAONA CASTAÑO**

**ANDRES LEONARDO PINZÓN BENÍTEZ**

**INGENIERIA DE SOFTWARE**

**WebSite Entrelineas**

**1. Introducción**

Esta documentación describe el proceso de integración del sitio web de la tienda online de libros, enfocándose en la comunicación entre frontEnd, backEnd y base de datos, así como el control de versiones con Git.

**2. Estructura del Proyecto**

El repositorio está organizado en las siguientes ramas:

* **FrontEnd**: Contiene los archivos HTML, CSS y JavaScript utilizando sus diferentes Frameworks.
* **Backend**: Contiene la lógica del servidor y la conexión a la base de datos.
* **Documentación**: Contiene los archivos de documentación del proyecto.
* **Integración**: Rama encargada de unir las diferentes partes del proyecto y realizar pruebas de funcionamiento.

**3. Pruebas de Integración**

Antes de realizar despliegues, se deben realizar pruebas en la rama de integración:

* Verificar la comunicación entre frontEnd y backend.
* Comprobar que los datos se almacenan correctamente en la base de datos.
* Revisar que el control de versiones mantiene el código organizado.

**POST MAN:**

Para las pruebas del Back-End usaremos la herramienta Postman para probar API RESTful , entonces usaremos postman para realizar peticiones HTTP y asi mismo obtener una respuesta sin tener aun la interfaz de usuario (Front-End) entonces eso quiere decir que podemos empezar a hacer pruebas al backend sin necesidad de depender del grupo de frontend.

El proceso de pruebas con Postman nos permitirá como grupo de integracion depurar errores en la comunicacion con la base de datos antes de conectar con el backend con el frontEnd.

**APIS DE TESTING:**

Para la integración de la base de datos con el backend , se usara una api rest en node.js para manejar autenticaciones , validar datos y realizar operaciones sobre la db.

Para la integración de frontend con backend tenemos pensado usar la API fetch() de javascript , que permite enviar y recibir datos de maneta asincrónica. Entonces buscamos que al momento de la integracion quede correctamente , el envío de peticiones HTTP desde el front al backend , el manejo de respuestas para actualizar la interfaz de usuario y el almacenamiento de tokens JWT para la autenticación y peticiones al backend.

**USO DE FETCH:**

La correcta integración entre frontend y backend asegura una comunicación fluida en la aplicación. Postman facilita la prueba de los endpoints antes de la conexión con la interfaz de usuario, mientras que el uso de fetch() en JavaScript permite consumir la API REST, garantizando una interacción dinámica y segura entre ambas capas del sistema.

**4. Despliegue**

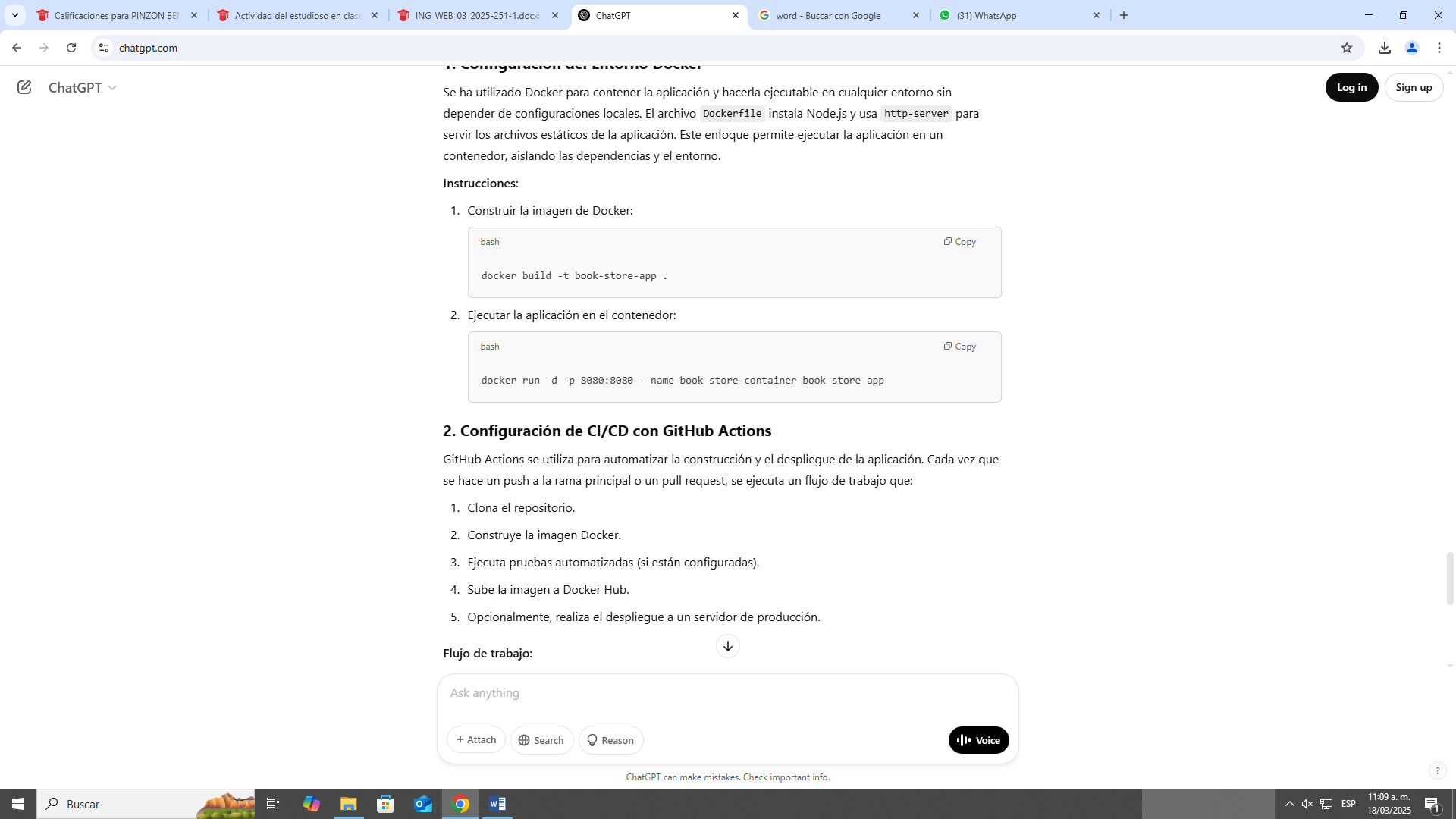
**CONFIGURACIÓN DEL ENTORNO DOCKER:**

Se ha utilizado Docker para contener la aplicación y hacerla ejecutable en cualquier entorno sin depender de configuraciones locales.

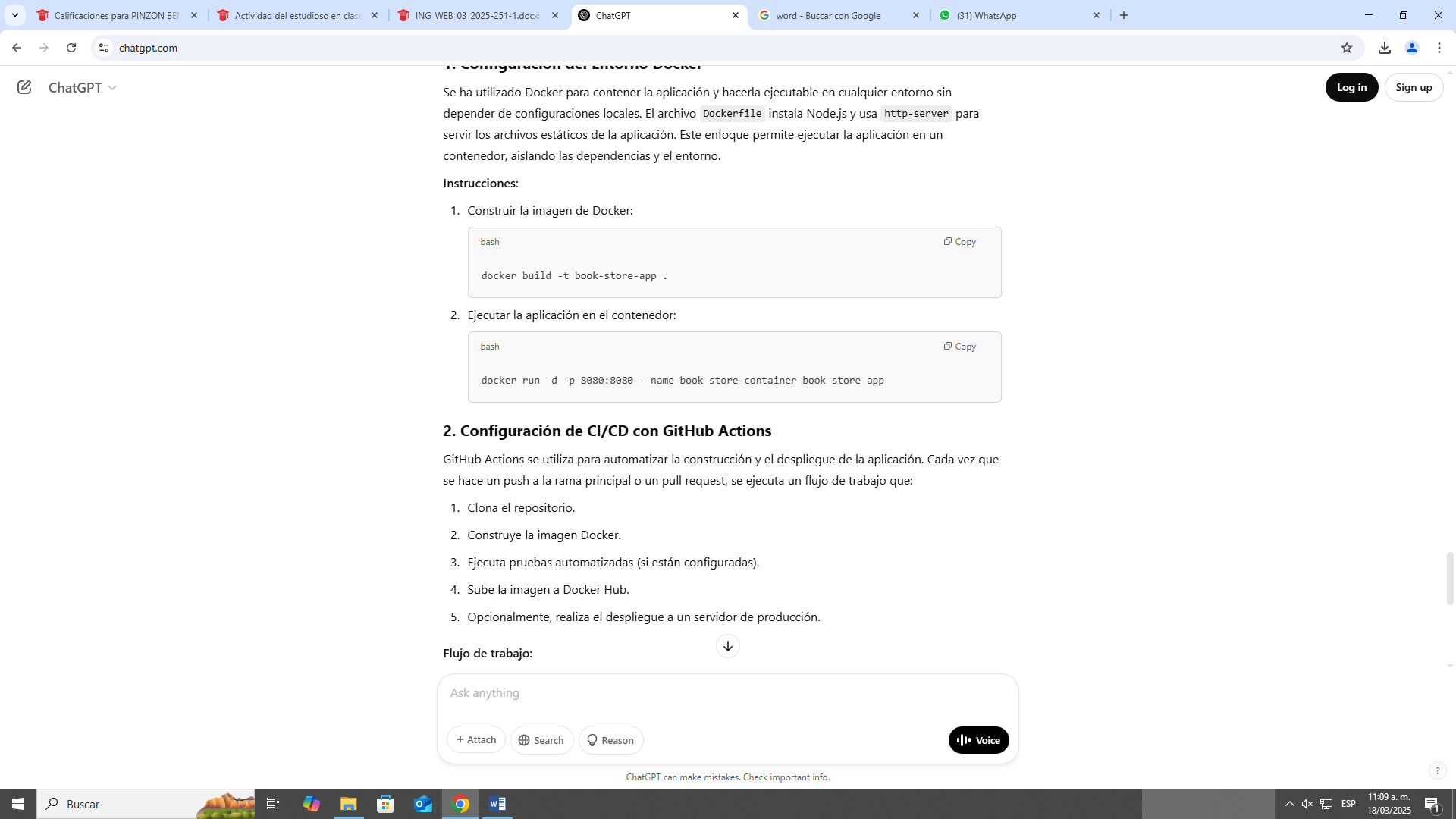
El archivo Dockerfile instala Node.js y usa http-server para servir los archivos estáticos de la aplicación. Este enfoque permite ejecutar la aplicación en un contenedor, aislando las dependencias y el entorno.

**Instrucciones:**

1. Construir la imagen de Docker:



1. Ejecutar la aplicación del Contenedor:



**CONFIGURACIÓN DE CI/CD CON GitHub Actions**

GitHub Actions se utiliza para automatizar la construcción y el despliegue de la aplicación. Cada vez que se hace un push a la rama principal o un pull request, se ejecuta un flujo de trabajo que:

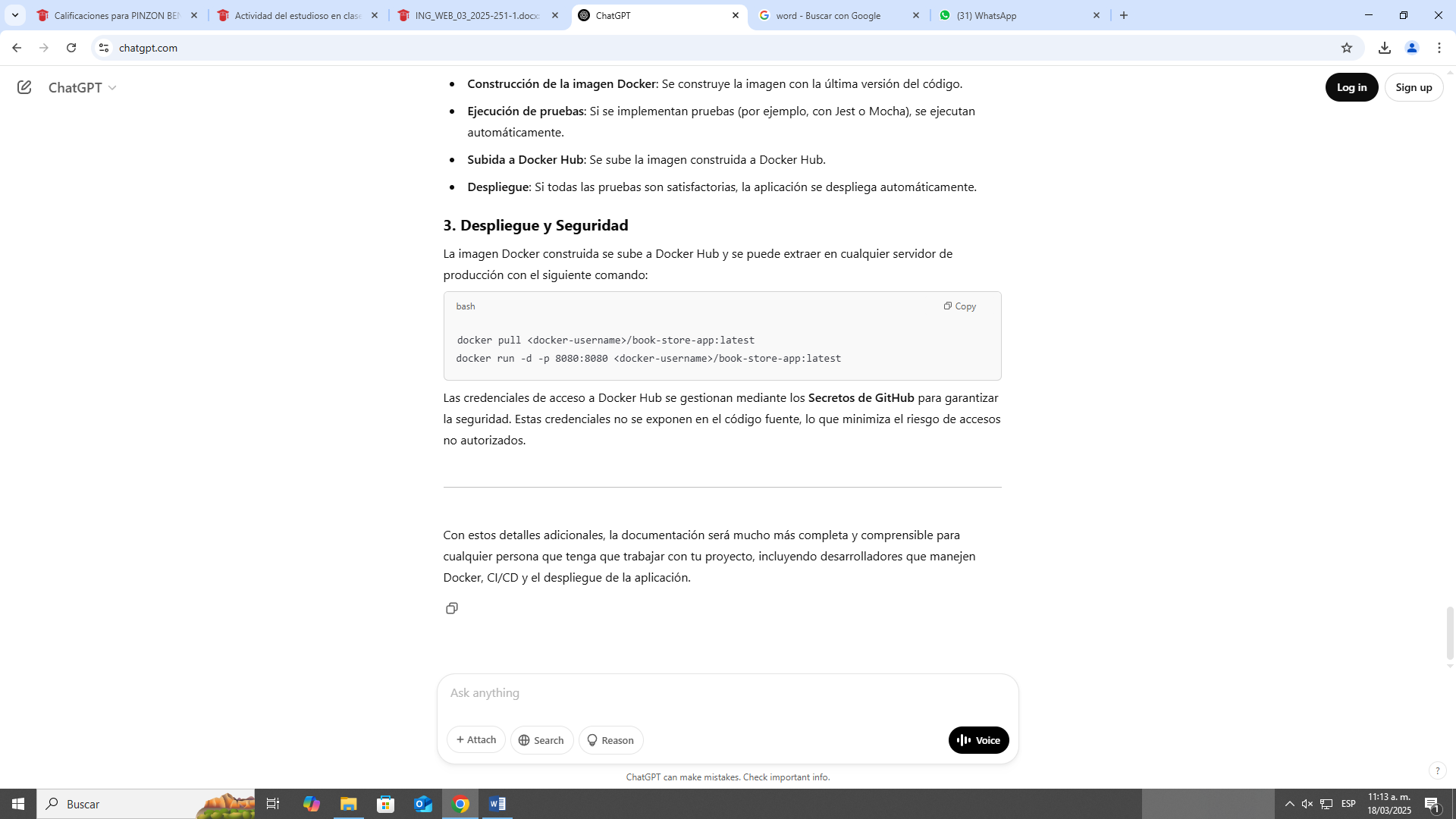
1. Clona el repositorio.
2. Construye la imagen Docker.
3. Ejecuta pruebas automatizadas (si están configuradas).
4. Sube la imagen a Docker Hub.
5. Opcionalmente, realiza el despliegue a un servidor de producción.

**Flujo de trabajo:**

* **Checkout del código**: Se clona el repositorio para obtener la versión más reciente.
* **Construcción de la imagen Docker**: Se construye la imagen con la última versión del código.
* **Ejecución de pruebas**: Si se implementan pruebas (por ejemplo, con Jest o Mocha), se ejecutan automáticamente.
* **Subida a Docker Hub**: Se sube la imagen construida a Docker Hub.
* **Despliegue**: Si todas las pruebas son satisfactorias, la aplicación se despliega automáticamente.

**DESPLIEGUE Y SEGURIDAD**

La imagen Docker construida se sube a Docker Hub y se puede extraer en cualquier servidor de producción con el siguiente comando:



Las credenciales de acceso a Docker Hub se gestionan mediante los **Secretos de GitHub** para garantizar la seguridad. Estas credenciales no se exponen en el código fuente, lo que minimiza el riesgo de accesos no autorizados.

**1. Sincronización de cambios entre backend y frontend**

**Uso de Git y control de versiones**

Para mantener un flujo de trabajo eficiente, se utilizará Git con una estructura de ramas bien definida:

* **Main**: Contiene la versión estable del proyecto, lista para despliegue en producción.
* **Develop**: Rama donde se integran los cambios antes de fusionarse en Main.
* **Feature**: Para el desarrollo de nuevas funcionalidades, se crean ramas individuales basadas en Develop.
* **Hotfix**: Se usa para corregir errores críticos en producción.

Cada cambio debe pasar por revisiones de código mediante pull requests antes de ser fusionado en Develop o Main.

**Integración Continua y Despliegue Continuo (CI/CD)**

* Uso de **GitHub Actions** o **Jenkins** para automatizar pruebas y despliegues.
* Configuración de **entornos de pruebas con Docker** para garantizar consistencia.
* Creación de scripts para automatizar la construcción del frontend y backend.
* Pruebas automáticas en cada commit para detectar fallos antes del despliegue.

**2. Verificación del funcionamiento de los componentes**

**Pruebas de Integración**

* Uso de **Postman** y **RestClient** para probar endpoints de la API.
* Validación de respuestas JSON y correcto manejo de errores en la comunicación entre frontend y backend.

**Pruebas Unitarias y de Integración**

* **Backend**: Uso de **Jest** y **Mocha** para pruebas unitarias en Node.js.
* **Frontend**: Uso de **Jest con React Testing Library** o **Cypress** para pruebas de componentes y UI.
* **Automatización con Selenium** para verificar interacciones en la interfaz.

**Monitoreo y Logging**

* Uso de **Winston** o **Log4js** para el registro de errores.
* Configuración de alertas en **Slack o Discord** para notificar fallos en el sistema.

**3. Corrección de Errores**

**Manejo de Conflictos en Git**

* Uso de git rebase y git merge según el caso.
* Resolución manual de conflictos con herramientas como VS Code o Sourcetree.
* Documentación de cambios mediante mensajes de commit claros.

**Depuración y Optimización**

* **Frontend**: Uso de DevTools de Chrome para analizar código y rendimiento.
* **Backend**: Uso de console.log, debugger y herramientas de profiling.
* **Base de Datos**: Monitoreo de consultas SQL con herramientas como **PgAdmin** o **MySQL Workbench**.

**4.Documentación del Proceso de Integración**

**Guía de Desarrollo**

* Pasos para configurar el entorno local.
* Convenciones de codificación y estructura de carpetas.
* Guía para contribuir al proyecto y realizar pull requests.

**Guía de Despliegue**

* Configuración de entornos en Docker.
* Estrategias de rollback en caso de errores en producción.

**Guía de Seguridad**

* Buenas prácticas para el manejo de credenciales.
* Uso de JWT para autenticación segura.
* Validación de entradas con Express Validator.