Herramientas y Tecnologías de Versionamiento para Canadian Visa Advise

Informe Técnico

Grupo 3: Emiliano Mendoza, Cesar Quiroga, Manuel Galindo, Deywid Mora

Tabla de Contenido

1. Introducción

El presente informe tiene como objetivo detallar las herramientas y tecnologías de control de versiones seleccionadas para el desarrollo del proyecto Canadian Visa Advise (CVA). Estas herramientas están orientadas a facilitar la colaboración en equipo, asegurar la integridad del código y permitir una integración continua efectiva.

2. Objetivo

El objetivo de este informe es establecer un sistema de control de versiones eficiente que permita un desarrollo colaborativo fluido, garantice la integridad del código fuente y facilite la integración continua para el proyecto CVA. Asimismo, se busca promover buenas prácticas en el control de versiones, asegurando la correcta gestión y trazabilidad del código.

3. Selección de Herramientas de Versionamiento

3.1 Sistema de Control de Versiones: Git

Para el sistema de control de versiones, se ha seleccionado **Git**. Git es una herramienta distribuida de control de versiones que permite gestionar cambios en el código de manera eficiente y colaborativa. Sus principales ventajas son:

- **Distribución**: Permite a los desarrolladores trabajar de manera independiente y offline, ya que cada uno tiene una copia completa del historial del proyecto.
- **Rapidez**: Las operaciones son muy rápidas en comparación con sistemas centralizados.
- **Branching y Merging**: Facilita la creación de ramas para el desarrollo de nuevas características o correcciones, y la posterior fusión de estas sin afectar la estabilidad del proyecto.
- **Escalabilidad**: Git puede manejar proyectos de cualquier tamaño, lo que lo convierte en una herramienta ideal para proyectos de crecimiento continuo como CVA.

3.2 Plataforma de Alojamiento: GitHub

Para el alojamiento del código y la gestión de repositorios, se ha elegido **GitHub**, que ofrece las siguientes ventajas:

- **Repositorios privados**: Permite alojar repositorios de forma privada sin coste adicional.
- **Integración con herramientas de CI/CD**: GitHub Actions facilita la automatización de flujos de trabajo, pruebas y despliegues.
- **Gestión de proyectos**: GitHub incorpora herramientas de seguimiento de issues y proyectos, lo que facilita la asignación de tareas y el control de progreso.
- **Colaboración**: GitHub proporciona mecanismos efectivos para la colaboración, como revisiones de código y solicitudes de fusión (pull requests).

4. Flujo de Trabajo Git

El flujo de trabajo que implementaremos es el modelo **Git Flow**, que permite gestionar el desarrollo en diferentes ramas para facilitar el trabajo colaborativo y la estabilidad del proyecto. Las ramas clave serán:

- `main` **(anteriormente master)**: Contiene el código que está listo para producción. Es la rama estable del proyecto y solo se actualiza con versiones completamente probadas.
- `develop`: Es la rama principal de desarrollo donde se integran todas las nuevas características. Esta rama puede contener código que aún está en prueba.
- `feature/`: Se crean ramas para desarrollar nuevas características o funcionalidades. Cada rama `feature` parte de `develop` y se fusiona en `develop` al completarse.
- `hotfix/`: Se usan para corregir errores críticos detectados en producción. Estas ramas parten de `main` y, una vez corregidos, se fusionan tanto en `main` como en `develop` para mantener la integridad del código.
- `release/`: Cuando se está preparando una nueva versión para lanzar, se crea una rama `release` desde `develop`. Esta rama sirve para realizar ajustes menores y pruebas finales antes de que el código pase a `main`.

Este flujo permite mantener versiones estables en producción mientras se trabaja de manera paralela en nuevas funcionalidades y correcciones.

5. Prácticas de Integración Continua

Para mantener un ciclo de desarrollo ágil y organizado, seguiremos estas prácticas de integración continua:

- **Commits frecuentes y pequeños**: Se recomienda realizar cambios pequeños y hacer commits frecuentes para evitar grandes conflictos al fusionar ramas.
- **Pruebas automatizadas**: Antes de integrar código nuevo, se ejecutarán pruebas automáticas para asegurar que los cambios no rompen la funcionalidad existente.
- **Integración diaria**: Los desarrolladores deben integrar sus cambios en la rama `develop` al menos una vez al día para detectar problemas de integración de manera temprana.
- **Revisión de código**: Todo código debe ser revisado antes de ser fusionado a las ramas `develop` o `main`, utilizando pull requests para realizar comentarios y sugerencias.

6. Configuración del Repositorio

La configuración inicial del repositorio de GitHub incluye la creación de las ramas `main` y `develop`, así como la estructura básica para nuevas características y correcciones. El repositorio se gestionará de forma privada en GitHub, y cada miembro del equipo debe clonar el repositorio en su máquina local, trabajar en una rama `feature/` correspondiente y, una vez completada la tarea, enviar una solicitud de fusión (pull request) a la rama `develop`.

El repositorio estará vinculado a GitHub Actions para realizar pruebas automatizadas y asegurar que los nuevos cambios no rompan el código existente. También se establecerán restricciones para fusionar código en `main`, requiriendo aprobación de al menos un revisor.

7. Conclusiones

La elección de Git y GitHub como las herramientas de control de versiones e integración continua proporcionará un entorno adecuado para el desarrollo colaborativo del proyecto CVA. Estas herramientas no solo facilitan la colaboración efectiva, sino que también aseguran la integridad del código y permiten gestionar versiones de manera eficiente a lo largo del ciclo de vida del proyecto.

8. Recomendaciones

- Capacitar al equipo en el uso de Git y el flujo de trabajo Git Flow para asegurar una correcta implementación de las ramas y fusiones.
- Implementar hooks de pre-commit para garantizar que los commits respeten las normas de calidad y formato del código.
- Establecer backups regulares del repositorio de GitHub para prevenir la pérdida de datos.
- Revisar periódicamente las prácticas de control de versiones y actualizar este plan según la evolución del proyecto.