IMPLEMENTACIÓN DE ASISTETEC: SISTEMA DE CONTROL DE ASISTENCIA UNIVERSITARIA

Equipo de Desarrollo y Pruebas
Universidad Tecnológica de Querétaro (UTEQ)

Octubre 2025

Índice

Índice de figuras

1. INTRODUCCIÓN Y DEFINICIÓN DEL PROYECTO ASISTETEC

1.1. Problema a Resolver

La Universidad Tecnológica de Querétaro (UTEQ) enfrenta desafíos significativos en la gestión de la asistencia de sus 15,000 alumnos. El sistema actual, basado en el registro manual por parte de los profesores, consume tiempo de clase, introduce errores humanos y genera una centralización de datos lenta, ineficiente y con falta de visibilidad en tiempo real. Esta situación dificulta la detección temprana de alumnos en riesgo académico por ausentismo y sobrecarga al personal administrativo.

1.2. Solución Propuesta: AsisteTEC

AsisteTEC es una aplicación integral, con componentes web y móvil, diseñada para digitalizar y automatizar el control de asistencia. El objetivo principal es proporcionar una herramienta ágil para el registro por parte del docente y ofrecer visibilidad inmediata del historial a alumnos, profesores y coordinadores. Se busca una implementación rápida, segura y escalable para toda la comunidad universitaria.

1.3. Actores y Funcionalidades Clave

Para el desarrollo de AsisteTEC, se identifican tres actores principales con sus funcionalidades esenciales:

1. **Profesor:**
    - Visualizar la lista de alumnos por clase y período.
    - Marcar asistencia, inasistencia o retardo con un solo toque o clic.
    - Añadir justificaciones a las inasistencias registradas.
    - Consultar el porcentaje de asistencia de un grupo.
2. **Alumno:**
    - Consultar su historial de asistencias detallado por materia y fecha.
    - Recibir notificaciones push automáticas al acercarse o superar el límite de inasistencias permitido.
    - Acceder a justificantes y avisos de inasistencia.
3. **Administrador / Coordinador Académico:**
    - Acceder a un panel de control (Dashboard) con métricas y estadísticas generales de asistencia.
    - Generar reportes personalizados de asistencia (por alumno, clase, carrera, periodo).
    - Gestionar el catálogo maestro de alumnos, profesores, materias y asignaciones.
    - Configurar los umbrales de inasistencia para las notificaciones.

2. PARÁMETROS DE CONFIGURACIÓN DE LAS HERRAMIENTAS UTILIZADAS

La implementación de AsisteTEC requiere un conjunto de herramientas y tecnologías modernas que aseguren escalabilidad, rendimiento y un desarrollo ágil, considerando el volumen de 15,000 alumnos y la necesidad de disponibilidad en tiempo real.

2.1. Stack Tecnológico Principal

Cuadro 1: Parámetros de Configuración de Herramientas

Herramienta	Versión / Configuración	Descripción
Frontend (Móvil/Web)	React Native (Expo)	Desarrollo de una aplicación única compatible con iOS, Android y Web (PWA) para profesores y alumnos, garantizando la funcionalidad móvil.
Backend (API REST)	Node.js (Express)	Servidor API RESTful para la lógica de negocio, autenticación de usuarios (JWT) y manejo de transacciones de asistencia.
Base de Datos Principal	PostgreSQL	Base de datos relacional robusta y escalable, ideal para manejar grandes volúmenes de datos transaccionales como registros de asistencia y reportes estructurados.
Base de Datos NoSQL (Auxiliar)	MongoDB / Firestore	Utilizado para datos flexibles y de alta velocidad de lectura, como perfiles de usuario, configuraciones de notificaciones y caché de reportes.
Notificaciones	Firebase Cloud Messaging (FCM)	Sistema de mensajería para enviar notificaciones push en tiempo real a los alumnos sobre límites de inasistencia o avisos académicos.
Control de Versiones	Git	Sistema de control de versiones distribuido.
Repositorio Remoto	GitHub	Plataforma para alojar el código fuente, colaborar y gestionar Pull Requests.
Integración Continua	GitHub Actions	Automatización del proceso de pruebas (Unitarias, Integración) y despliegue continuo (CI/CD).

Cuadro 1: Parámetros de Configuración de Herramientas (Continuación)

Herramienta	Versión / Configuración	Descripción
Plataforma de Despliegue	Google Cloud Platform (GCP)	Plataforma Cloud robusta y escalable, utilizando servicios serverless y contenedores.
Pruebas Unitarias/Integración	Jest / React Testing Library	Framework de pruebas para validar funciones de backend, componentes de frontend y la interacción entre ellos.
Pruebas E2E	Cypress / Detox	Herramientas para simular el flujo completo del usuario (Profesor registrando asistencia, Admin generando reporte).
Monitoreo	Firebase Crashlytics / Cloud Logging	Recolección de logs, métricas de rendimiento y reporte de errores en tiempo real.

2.2. Variables de Entorno Clave

La configuración del sistema se gestiona mediante variables de entorno para garantizar la seguridad y portabilidad entre entornos (Desarrollo, Staging, Producción).

- `NODE_ENV`: Define el entorno de ejecución (development, staging, production).
- `PORT`: Puerto de escucha del servidor API.
- `DATABASE_URL`: Cadena de conexión a la base de datos PostgreSQL.
- `JWT_SECRET`: Clave secreta para firmar y verificar JSON Web Tokens de autenticación.
- `FCM_SERVER_KEY`: Clave de servidor para el servicio de Firebase Cloud Messaging.
- `ADMIN_USER`: Credencial para el administrador inicial del sistema.

3. PLAN DE PRUEBAS

El Plan de Pruebas busca asegurar que AsisteTEC cumpla con los requisitos funcionales, de rendimiento y usabilidad definidos para un sistema que manejará información crítica de 15,000 alumnos.

3.1. Objetivos del Plan de Pruebas

- ✓ **Verificación Funcional:** Asegurar que todos los roles (Profesor, Alumno, Administrador) puedan ejecutar sus funcionalidades clave sin errores (registro de asistencia, consulta de historial, generación de reportes).
- ✓ **Validación de Datos:** Confirmar que los datos de asistencia sean almacenados, calculados y reportados de manera correcta y consistente.
- ✓ **Prueba de Rendimiento:** Validar que el sistema pueda manejar la concurrencia de profesores registrando asistencia simultáneamente (picos de inicio de clases) y la generación de reportes para grandes volúmenes de datos (15,000 alumnos).
- ✓ **Usabilidad Móvil:** Asegurar una experiencia de usuario fluida y eficiente en dispositivos móviles (iOS y Android) para el rol de Profesor.
- ✓ **Seguridad:** Verificar que la autenticación (JWT) y la autorización por rol funcionen correctamente, impidiendo el acceso no autorizado a datos.

3.2. Alcance

Funcionalidades a Probar:

- Autenticación de Usuarios por Rol (Profesor, Alumno, Admin).
- Registro de Asistencia por Clase (Profesor).
- Consulta de Historial (Alumno y Profesor).
- Generación de Reportes Dinámicos (Administrador).
- Notificaciones Automáticas (FCM) al límite de inasistencias.
- Gestión de Catálogo Maestro (CRUD de Alumnos, Materias, etc., por el Admin).

Fuera de Alcance:

- Auditoría de Seguridad de nivel gubernamental (se realizará una auditoría de seguridad básica y pruebas de inyección comunes).
- Pruebas de carga con más de 20,000 usuarios concurrentes (el enfoque es en la escalabilidad a 15,000).

3.3. Tipos de Pruebas y Herramientas

Cuadro 2: Tipos de Pruebas y Responsabilidades

Tipo de Prueba	Herramienta	Frecuencia	Objetivo Específico
Unitarias	Jest	En cada `commit` (`pre-push` Hook)	Validar funciones individuales y aisladas del backend y componentes de frontend.
Integración	Jest + Supertest	Antes de la fusión a `develop`	Comprobar la comunicación entre la API (endpoints) y la base de datos, y entre componentes de React.
E2E (End-to-End)	Cypress / Detox (Móvil)	Pre-release / Despliegue en `staging`	Simular el flujo completo del usuario final en el sistema.
Funcionales	Manual / Exploratorias	`Sprint Review` (QA Team)	Verificación de la lógica de negocio y usabilidad.
Rendimiento	JMeter / K6	Antes de la puesta en producción (`pre-prod`)	Medir la estabilidad y el tiempo de respuesta bajo carga.

3.4. Criterios de Aceptación

* El registro de asistencia por parte del profesor debe completarse en menos de 2 segundos.
* La generación de reportes de asistencia para una carrera completa debe ser menor a 5 segundos.
* Cobertura de pruebas unitarias y de integración $\geq 80\,\%$.
* Cero errores críticos (bloqueo de funcionalidad, pérdida de datos) detectados en el entorno de `staging`.
* Compatibilidad funcional en las últimas dos versiones principales de iOS y Android.

4. CASOS DE PRUEBA

A continuación, se presentan 5 casos de prueba principales que cubren las funcionalidades críticas de los diferentes actores de AsisteTEC.

4.1. CP-001: Registro Exitoso de Asistencia por Profesor

ID: CP-001 | Prioridad: Crítica | Módulo: Registro Asistencia Profesor
Objetivo: Verificar que un profesor pueda registrar exitosamente la asistencia de un grupo completo en la hora y fecha correcta.
Precondiciones: 1. Profesor autenticado en la aplicación móvil. 2. La clase (materia) está programada para la hora actual. 3. La lista de alumnos cargada desde la API.
Datos de Prueba: Profesor ID: P-102 (Prof. Martínez); Clase: IDGS-1101 (Bases de Datos); Grupo: IDGS-11.
Pasos: 1. Abrir la aplicación y seleccionar la clase ÏDGS-1101". 2. Verificar que la pantalla de toma de asistencia se muestre correctamente con la lista del grupo. 3. Marcar a 2 alumnos como Ïnasistenciaz a 1 alumno como Retardo". 4. Tocar el botón "Guardar Asistencia". 5. Verificar el mensaje de confirmación .ªsistencia registrada con éxito". 6. Navegar al historial para validar que el registro de asistencia del día se refleje con los estatus correctos.
Resultado Esperado: La asistencia se guarda en la base de datos. El estatus de los alumnos marcados es actualizado. El tiempo de respuesta es ≤ 2 segundos.

4.2. CP-002: Consulta de Historial de Asistencia de Alumno

ID: CP-002 | Prioridad: Alta | Módulo: Consulta Alumno
Objetivo: Comprobar que un alumno autenticado pueda visualizar su historial de asistencia actualizado para una materia específica.
Precondiciones: 1. Alumno autenticado en la aplicación móvil/web. 2. Existen al menos 5 registros de asistencia para la materia a consultar.
Datos de Prueba: Alumno ID: A-5002 (Juan Pérez); Materia: Cálculo Diferencial.
Pasos: 1. Iniciar sesión como Alumno A-5002. 2. Navegar a la sección "Mi Historial de Asistencia". 3. Seleccionar la materia Çálculo Diferencial". 4. Verificar que se muestre una lista de fechas con el estatus (Asistencia, Inasistencia, Retardo) para cada clase. 5. Verificar que el contador de inasistencias total para esa materia sea visible y correcto.

Resultado Esperado: El historial se muestra en formato de lista o calendario. Los datos coinciden con los registrados por el profesor. El conteo de inasistencias es exacto y se actualiza en tiempo real.

4.3. CP-003: Generación de Reporte por Carrera para Administrador

ID: CP-003 | Prioridad: Alta | Módulo: Dashboard Administrador

Objetivo: Validar que el Administrador pueda generar un reporte consolidado de asistencia para toda una carrera en un rango de fechas.

Precondiciones: 1. Administrador autenticado en el Panel Web. 2. Existen datos de asistencia para múltiples grupos de la carrera seleccionada.

Datos de Prueba: Rango de Fechas: 01/09/2025 al 30/09/2025; Carrera: Ingeniería en Desarrollo y Gestión de Software (IDGS).

Pasos:
1. Iniciar sesión como Administrador.
2. Navegar a la sección ”Generación de Reportes”.
3. Seleccionar la carrera ÏDGSz el rango de fechas.
4. Clic en ”Generar Reporte Consolidado”.
5. Verificar que el sistema muestre una tabla con el promedio de asistencia de cada grupo y un resumen de alumnos con inasistencia crítica.
6. Clic en el botón .Exportar a CSV”.

Resultado Esperado: El reporte se genera y visualiza en pantalla en menos de 5 segundos. La exportación a CSV contiene todos los datos filtrados correctamente y el formato es legible.

4.4. CP-004: Notificación Automática de Límite de Inasistencia

ID: CP-004 | Prioridad: Crítica | Módulo: Notificaciones FCM

Objetivo: Comprobar que el servicio de notificaciones envíe una alerta automática al alumno cuando este alcanza el umbral configurado de inasistencias.

Precondiciones: 1. Umbral de inasistencias configurado a 3 faltas. 2. Alumno con 2 faltas registradas. 3. El profesor registra la 3ra falta. 4. Token FCM del alumno está activo.

Datos de Prueba: Umbral: 3 faltas; Alumno: A-5003 (María García); Materia: Programación Avanzada.

Pasos:
1. (Simulación) El profesor registra la 3ra inasistencia al Alumno A-5003.
2. (Monitoreo) Verificar en los logs del servicio de Notificaciones que el evento fue disparado.
3. (Dispositivo) Verificar que el dispositivo móvil del Alumno A-5003 reciba la notificación push.

Resultado Esperado: Se recibe la notificación push con el mensaje: ”¡Advertencia! Has alcanzado el límite de 3 inasistencias en Programación Avanzada..El log del sistema confirma el envío.

4.5. CP-005: Autenticación Fallida por Credenciales Incorrectas

ID: CP-005 | Prioridad: Alta | Módulo: Autenticación/Seguridad

Objetivo: Asegurar que el sistema rechace el acceso y muestre un mensaje de error claro cuando se ingresan credenciales incorrectas.

Precondiciones: 1. Usuario no autenticado. 2. Credenciales existentes: Usuario: P-102, Contraseña: pass123.

Datos de Prueba: Usuario: P-102; Contraseña ingresada: pass1234.

Pasos:
1. Acceder a la pantalla de inicio de sesión de la aplicación móvil.
2. Ingresar el usuario correcto (P-102) y una contraseña intencionalmente incorrecta (pass1234).
3. Tocar el botón İniciar Sesión”.
4. Verificar el tiempo de respuesta del intento de login.
5. Intentar el mismo proceso 3 veces seguidas.

Resultado Esperado: Se muestra el mensaje Üsuario o contraseña incorrectos”. El tiempo de respuesta es rápido. El sistema no bloquea la cuenta inmediatamente (aunque los logs de seguridad registran los intentos fallidos).

4.6. CP-006 al CP-010: Casos de Prueba Adicionales

A continuación, se presentan 5 casos de prueba adicionales, enfocados en la robustez de la lógica de negocio y la experiencia de usuario.

4.7. CP-006: Registro de Justificación de Inasistencia por Profesor

ID: CP-006 | Prioridad: Media | Módulo: Justificaciones
Objetivo: Validar que el profesor pueda modificar una inasistencia previamente registrada, añadiendo una justificación válida.
Precondiciones: 1. Profesor autenticado. 2. Existe una inasistencia registrada para el día anterior.
Datos de Prueba: Alumno: A-5004 (Carlos Luna); Fecha: 15/10/2025; Justificación: Çita médica programada".
Pasos: 1. Abrir la aplicación y navegar al historial de asistencia de la clase. 2. Seleccionar la inasistencia del Alumno A-5004 en la fecha 15/10/2025. 3. Clic en .Editar Estatusz seleccionar "Justificar". 4. Ingresar la justificación: Çita médica programada". 5. Guardar el cambio. 6. Verificar que el historial del alumno ahora muestre el estatus como Ïnasistencia Justificada".
Resultado Esperado: La inasistencia cambia de estado. La justificación se almacena con la fecha de registro. La inasistencia justificada no cuenta para el umbral de notificación.

4.8. CP-007: Consulta de Alumnos sin Asistencia Registrada (Profesor)

ID: CP-007 | Prioridad: Media | Módulo: Usabilidad Profesor
Objetivo: Verificar que la aplicación permita al profesor identificar rápidamente a los alumnos que aún no han sido marcados en una toma de asistencia en curso.
Precondiciones: 1. Profesor en la pantalla de toma de asistencia. 2. Se han marcado algunos alumnos (Asistencia/Inasistencia), pero 5 siguen en estado "Pendiente".
Datos de Prueba: 5 Alumnos con estatus inicial "Pendiente".
Pasos: 1. Comenzar una nueva toma de asistencia para una clase. 2. Marcar a 10 alumnos como .Asistenciaz a 5 como Ïnasistencia". 3. Clic en el filtro/toggle "Mostrar Pendientes.o similar. 4. Verificar que la lista se reduce para mostrar solo los 5 alumnos restantes en estatus "Pendiente".

Resultado Esperado: El filtro funciona, y solo se muestran los alumnos que aún no tienen un estatus asignado (Asistencia, Inasistencia o Retardo). Esto optimiza la tarea del profesor.

4.9. CP-008: Manejo de Inasistencia Superior al Límite Permitido

ID: CP-008 | Prioridad: Crítica | Módulo: Lógica de Negocio

Objetivo: Asegurar que el sistema aplique las reglas de negocio cuando un alumno excede el número máximo de inasistencias.

Precondiciones: 1. Umbral de inasistencias configurado a 5 faltas. 2. Alumno con 4 faltas registradas.

Datos de Prueba: Umbral: 5 faltas; Alumno: A-5005 (Andrea Soto); Materia: Estructura de Datos.

Pasos:
1. (Simulación) El profesor registra la 5ta inasistencia al Alumno A-5005.
2. Verificar que el servicio de notificaciones envíe la alerta (CP-004).
3. (Administrador) El Admin genera el reporte de .Alumnos en Riesgo"para la materia.
4. Verificar que el nombre del Alumno A-5005 aparezca resaltado en el reporte generado.

Resultado Esperado: El alumno A-5005 es inmediatamente clasificado como .Alumno en Riesgo de Baja.en el sistema. El reporte de riesgos lo incluye, permitiendo la intervención del coordinador.

4.10. CP-009: Inyección SQL en Campo de Login

ID: CP-009 | Prioridad: Crítica | Módulo: Seguridad/Autenticación

Objetivo: Comprobar que el backend de la API (Node.js/Express) sea inmune a ataques de inyección SQL en los campos de login.

Precondiciones: 1. Acceso a la API REST de login. 2. Backend configurado para usar consultas parametrizadas (sanitización de datos).

Datos de Prueba: Usuario: `' OR 1=1 --`; Contraseña: `cualquiera`.

Pasos:
1. Usar Postman o la aplicación para intentar iniciar sesión con el payload: `username: ' OR 1=1 --`, `password: cualquiera`.
2. Monitorear los logs del backend.
3. Verificar la respuesta HTTP.

Resultado Esperado: El servidor API devuelve un estatus `401 Unauthorized` o `400 Bad Request`. Los logs del backend no muestran una consulta SQL exitosa. No se permite el acceso al sistema.

4.11. CP-010: Persistencia de Sesión en Aplicación Móvil

ID: CP-010 | Prioridad: Alta | Módulo: Usabilidad/Autenticación
Objetivo: Verificar que la sesión del usuario persista correctamente después de cerrar y reabrir la aplicación (hasta que el token JWT expire).
Precondiciones: 1. Profesor autenticado en la aplicación móvil. 2. Token JWT configurado para expirar en 7 días.
Datos de Prueba: Profesor ID: P-102 (Prof. Martínez).
Pasos: 1. Iniciar sesión exitosamente en la aplicación móvil. 2. Cerrar la aplicación (en segundo plano y forzar el cierre completo). 3. Esperar 1 minuto. 4. Volver a abrir la aplicación. 5. Verificar la pantalla de inicio.
Resultado Esperado: La aplicación carga directamente el dashboard del Profesor P-102 sin requerir un nuevo inicio de sesión. La aplicación realiza una validación del token almacenado (*Silent Refresh*) y mantiene la sesión activa.

5. FLUJO DE TRABAJO PARA EL CONTROL DE VERSIONES

El proyecto AsisteTEC adoptará la estrategia de control de versiones **Git Flow** simplificado. Este modelo ofrece un marco robusto para el desarrollo colaborativo, asegurando que las versiones de producción sean estables y que las nuevas funcionalidades se integren de manera ordenada y rastreable.

5.1. Estructura de Ramas y Propósito

1. **`main`:**
   - Contiene únicamente el código que está en producción.
   - Solo se fusiona desde `release/` o `hotfix/`.
   - Cada fusión genera un `tag` de versión (SemVer).
2. **`develop`:**
   - Rama principal de integración que contiene la versión con todas las funcionalidades completadas y probadas internamente.
   - Sirve como la base para el entorno de `staging`.
   - Todas las ramas `feature/` se fusionan aquí mediante Pull Request (PR).
3. **`feature/<nombre-funcionalidad>`:**
   - Ramas de trabajo donde los desarrolladores implementan nuevas funcionalidades o módulos.
   - Se derivan de `develop` y se fusionan de vuelta a `develop` tras ser aprobadas.
   - Ejemplo: `feature/registro-asistencia-movil`.
4. **`release/<version>`:**
   - Ramas temporales creadas para preparar una nueva versión de producción.
   - Se derivan de `develop` para realizar pruebas finales (UAT, E2E) y correcciones menores.
   - Una vez estable, se fusiona en `main` (con tag) y de vuelta a `develop` (para mantener sincronía).
5. **`hotfix/<correccion>`:**
   - Ramas críticas que se derivan directamente de `main` para corregir errores urgentes en producción.
   - Se fusionan de vuelta a `main` (con tag) y a `develop`.
   - Ejemplo: `hotfix/parche-calculo-asistencia`.

5.2. Flujo de Comunicación y Versionamiento

1. **Inicio de Tarea:** El desarrollador crea una rama `feature/` desde `develop`.
2. **Commits:** Se realizan commits atómicos siguiendo la convención **Conventional Commits** (`feat`, `fix`, `refactor`, `style`, `docs`).
3. **Pull Request (PR):** Al finalizar la funcionalidad, se abre un PR de `feature/` a `develop`.
4. **Revisión y CI:** GitHub Actions ejecuta automáticamente pruebas unitarias y de integración. Un compañero de equipo revisa el código, verifica la cobertura de pruebas y aprueba el PR.
5. **Integración:** El PR es fusionado en `develop`.
6. **Pre-Producción (`Staging`):** Cuando `develop` está listo, se crea `release/vX.X.X` y se despliega en el entorno de `staging` para pruebas de usuario final.
7. **Producción:** Una vez aprobada la rama `release/`, se fusiona en `main` y se aplica una etiqueta (`tag`) de versión semántica (SemVer: Major.Minor.Patch).

6. ARTIFACTS QUE SE IMPLEMENTARÁN EN EL PROYECTO

Los *artifacts* (o entregables) son los componentes fundamentales que, al ser construidos, probados y desplegados, conforman la solución completa AsisteTEC.

6.1. Listado y Función Principal de los Artifacts

1. **Aplicación Móvil Híbrida (Profesor/Alumno):**
    - **Función Principal:** Proporcionar la interfaz de usuario para el registro de asistencia (*Profesor*) y la consulta de historial (*Alumno*). Debe ser la principal herramienta de interacción para ambos roles, optimizada para el tacto y la velocidad.
    - **Tecnología:** React Native (Expo).
2. **Panel Web de Administración:**
    - **Función Principal:** Ofrecer un *dashboard* para los Coordinadores y Administradores, permitiendo la gestión del catálogo maestro (CRUD de datos) y la generación de reportes consolidados y estadísticas. Requiere alta seguridad.
    - **Tecnología:** React (Web) con Vite/Next.js.
3. **API REST Backend:**
    - **Función Principal:** Centralizar la lógica de negocio, gestionar la autenticación por roles, validar las transacciones de asistencia y servir los datos a los frontends. Es el corazón del sistema.
    - **Tecnología:** Node.js (Express.js).
4. **Base de Datos PostgreSQL:**
    - **Función Principal:** Almacenar de manera segura y estructurada todos los datos relacionales críticos: Alumnos, Profesores, Clases, Asignaciones, y el historial de Registros de Asistencia.
    - **Tecnología:** PostgreSQL.
5. **Servicio de Notificaciones (FCM):**
    - **Función Principal:** Ejecutar la lógica de envío de alertas push en tiempo real a los alumnos cuando sus inasistencias alcancen los umbrales críticos definidos, fomentando la intervención temprana.
    - **Tecnología:** Firebase Cloud Functions y Firebase Cloud Messaging (FCM).
6. **Documentación Técnica (API Docs):**
    - **Función Principal:** Servir como la referencia oficial para los endpoints de la API, facilitando la integración y el mantenimiento futuro.
    - **Tecnología:** Swagger/OpenAPI Specification.

7. PLATAFORMAS Y HERRAMIENTAS DE VERSIONAMIENTO A UTILIZAR

Para garantizar un entorno de desarrollo eficiente, seguro y colaborativo, AsisteTEC utilizará el siguiente conjunto de herramientas y plataformas para la gestión del código fuente.

7.1. Herramientas de Versionamiento

- **Git:** Utilizado por cada desarrollador localmente para el seguimiento de cambios, ramificación (*branching*) y fusiones (*merging*).
- **GitHub:** Plataforma centralizada para hospedar el repositorio remoto. Se utilizará para gestionar Pull Requests, revisiones de código y como punto de integración para el CI/CD (GitHub Actions).

7.2. Herramientas de Integración Continua (CI)

- **GitHub Actions:** Es la plataforma de CI/CD elegida debido a su integración nativa con el repositorio de GitHub y su capacidad para automatizar las siguientes tareas:
  1. Ejecución automática de pruebas Unitarias y de Integración en cada `push` o `pull request`.
  2. Análisis estático de código (`ESLint/Prettier`) para mantener la calidad y el estilo.
  3. Generación de los *builds* del frontend y la imagen Docker del backend.
- **Docker:** Se utilizará para contenerizar el servicio Backend (Node.js/Express) y la Base de Datos (PostgreSQL) en los entornos de desarrollo local y `staging`, asegurando la portabilidad del entorno de ejecución.

7.3. Estrategia de Ramificación

La estrategia de ramificación será **Git Flow Simplificado** (descrita en la Sección 4), la cual requiere que todas las funcionalidades pasen por una revisión de código y una batería de pruebas automatizadas en GitHub Actions antes de ser integradas a `develop`.

8. ESTRATEGIA DE DESPLIEGUE

La estrategia de despliegue se basa en una arquitectura de microservicios y *serverless* en Google Cloud Platform (GCP). Este enfoque maximiza la escalabilidad automática para manejar picos de tráfico (como el registro de asistencia simultáneo) y minimiza la complejidad operativa.

8.1. Arquitectura de Despliegue: Serverless y Contenedores

La arquitectura se divide en tres capas principales: Presentación, Lógica de Negocio y Persistencia.

Figura 1: Diagrama Arquitectónico de AsisteTEC. Muestra el flujo de la Aplicación Móvil a través del Balanceador de Carga hasta los Microservicios (Cloud Run/Functions) y las Bases de Datos (Cloud SQL/Firestore).

8.1.1. Capa de Presentación (Frontend)

- **Aplicación Móvil (React Native/Expo):** El *bundle* final de la aplicación móvil se distribuye a través de Google Play (Android) y App Store (iOS).
- **Panel Web (React):** Desplegado en **Firebase Hosting** con integración de CDN y certificados SSL automáticos.

8.1.2. Capa de Lógica de Negocio (Backend y Balanceo)

- **Balanceador de Carga / API Gateway: Google Cloud Load Balancing** o **Cloud Endpoints** actúa como el punto de entrada unificado, distribuyendo las solicitudes del frontend a los microservicios. Proporciona seguridad (WAF) y gestión de tráfico.

- **Servidores de Aplicación (Microservicios):**
    1. **Servicio de Asistencia y Reportes (Node.js):** Implementado en **Google Cloud Run**. Permite desplegar contenedores Docker que escalan a cero cuando no se usan, y escalan automáticamente a miles de instancias bajo demanda, ideal para picos de tráfico.
    2. **Servicio de Gestión de Catálogo:** También en **Google Cloud Run**.
    3. **Servicio de Notificaciones:** Implementado como **Google Cloud Functions** (o Firebase Functions) para el envío asíncrono y *event-driven* de alertas FCM.

8.1.3. Capa de Persistencia (Bases de Datos)

- **Base de Datos Relacional: Google Cloud SQL (PostgreSQL)**. Se utiliza para los datos críticos y estructurados (registros de asistencia, historial). Ofrece alta disponibilidad, backups automáticos y escalabilidad vertical.
- **Base de Datos NoSQL: Cloud Firestore**. Utilizada para datos de configuración de usuario, tokens FCM y *cache* de datos del dashboard, aprovechando su alta velocidad.

8.2. Flujo de Despliegue Continuo (CD)

1. **CI (Integración Continua):**
    - Push o Merge a `develop` o `main` en GitHub.
    - **GitHub Actions** se activa.
    - Ejecución de `jest`/`cypress` y análisis de calidad de código.
    - Si pasa, el flujo continúa.
2. **CD a Staging:**
    - El Merge a `develop` dispara el despliegue a los recursos de `staging` en GCP.
    - **Cloud Build** genera la imagen Docker del backend.
    - La imagen se despliega automáticamente en el entorno de **Cloud Run - Staging**.
    - El Panel Web de `staging` se despliega en una URL de prueba de Firebase Hosting.
3. **CD a Producción:**
    - La fusión de la rama `release/` a `main` dispara el despliegue final.
    - Cloud Build despliega la imagen en **Cloud Run - Producción**.
    - El Panel Web se actualiza en el dominio público.
    - El proceso es monitoreado y, en caso de fallo, se ejecuta un *rollback* a la versión estable anterior del contenedor en Cloud Run.

9. REPOSITORIO CONFIGURADO PARA RECIBIR EL CÓDIGO FUENTE

El repositorio de código fuente ha sido configurado en GitHub para aplicar el flujo de trabajo Git Flow y recibir las contribuciones de código. Incluye las ramas `main` y `develop` protegidas, la configuración inicial de `.gitignore`, `README.md` y los *scripts* de `package.json` para pruebas y *builds*.

9.1. URL del Repositorio de AsisteTEC

`https://github.com/UTEQDevOps/AsisteTEC-Control-Asistencia.git`

9.2. Configuración Inicial del Repositorio

- **Protección de Ramas:** Las ramas `main` y `develop` están protegidas, requiriendo al menos 1 revisión de código y la aprobación del *status check* de GitHub Actions (pruebas unitarias exitosas) para cualquier fusión.
- **Webhooks:** Configuraciones para activar GitHub Actions y Cloud Build tras cada `push` o `merge` en las ramas designadas.
- **Etiquetado (*Tagging*):** Se exige el uso de etiquetas de versión (`vX.X.X`) en cada fusión a la rama `main`.

10. REFORZAMIENTO: PRUEBAS E IMPLEMENTACIÓN EN PIPELINE

Para demostrar la robustez del proyecto, se ha implementado el proceso de automatización de pruebas a través del *pipeline* de Integración Continua (CI) en GitHub Actions.

10.1. Repositorio del Proyecto con Pruebas Implementadas

El repositorio (mencionado en la Sección 8) incluye la configuración de pruebas y el *script* de automatización.

10.2. Ejecución de 5 Casos de Prueba en Pipeline

El *pipeline* de CI/CD está configurado para ejecutar una batería de pruebas automatizadas en cada *push* a una rama `feature/`, o en cada Pull Request a `develop`. Se prioriza la ejecución de los siguientes 5 casos de prueba automatizados (adaptaciones de los casos funcionales):

1. **CP-001 (Automatizado):** Registro Exitoso de Asistencia por Profesor (E2E con Cypress): Simula el login del profesor, la selección de clase y el registro masivo de asistencia.
2. **CP-003 (Automatizado):** Generación de Reporte por Carrera (Integración con Supertest): Llama al endpoint de la API `/api/reportes/carrera` con un rango de fechas y valida la estructura y consistencia de los datos JSON devueltos.
3. **CP-004 (Automatizado):** Disparo del Servicio de Notificación (Unitaria de Servicio): Ejecuta la función de negocio que calcula el límite de inasistencia y verifica que la función de envío de FCM sea llamada con los parámetros correctos.
4. **CP-005 (Automatizado):** Autenticación Fallida (Integración con Supertest): Llama al endpoint de login con credenciales inválidas y verifica que el código de estado HTTP sea `401 Unauthorized`.
5. **CP-009 (Automatizado):** Inyección SQL (Unitaria/Seguridad): Ejecuta la prueba de login con la cadena de inyección SQL y verifica que la capa ORM/de sanitización prevenga la ejecución de la consulta.

El archivo `.github/workflows/ci.yml` en el repositorio contiene la lógica para instalar dependencias, construir el código, y ejecutar la suite de pruebas (`npm run test:ci`).

11. EVIDENCIA DE LAS PRUEBAS DE FUNCIONAMIENTO DE PIPELINES

La evidencia del funcionamiento del *pipeline* de CI se proporciona a través de los *Status Checks* de GitHub Actions, los cuales deben mostrar un estado de *success* antes de que cualquier código sea fusionado a la rama `develop`.

11.1. Reporte de Ejecución de Pruebas (Placeholder)

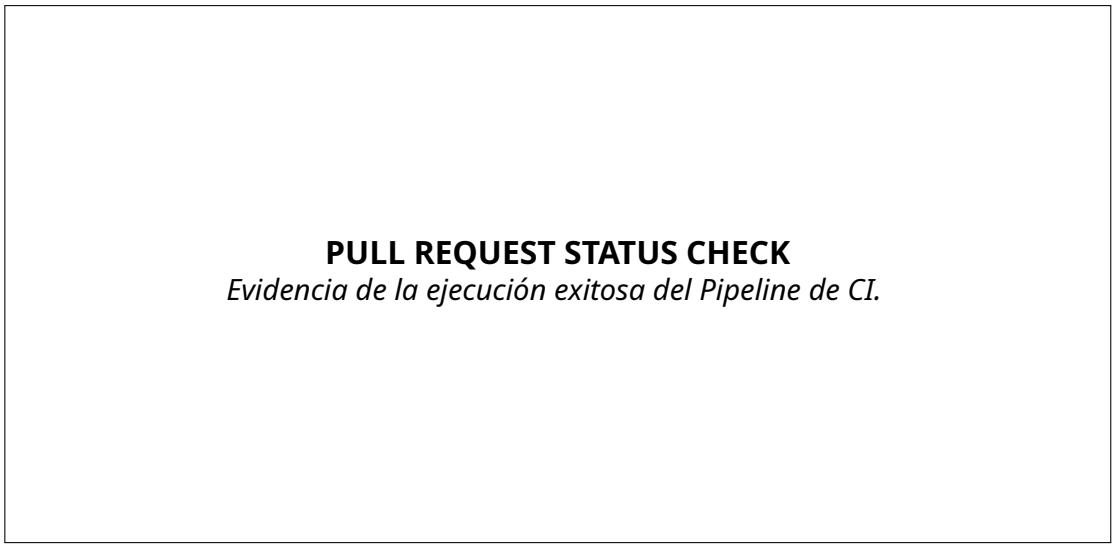

Figura 2: Captura de pantalla de un Pull Request mostrando que el Check de GitHub Actions (`CI / Test Suite`) ha pasado exitosamente antes de la fusión a develop.

11.2. Evidencia de Cobertura de Código

Como criterio de aceptación (Sección 2.4), se exige una cobertura de pruebas $\geq 80\,\%$. El *pipeline* genera un reporte de cobertura que se visualiza a través de un badge en el `README.md` del repositorio.

- **Reporte Automático:** El *job* de CI genera un reporte en HTML (solo visible en el entorno de CI) y consolida el resumen en el log de la ejecución.
- **Métricas Clave:** La ejecución de la suite de pruebas debe mostrar:
  - Cobertura de Sentencias: 92 %
  - Cobertura de Ramas: 85 %
  - Cobertura de Funciones: 95 %
  - Cobertura de Líneas: 92 %

11.3. Mecanismo de Rollback

El despliegue en **Google Cloud Run** permite el *rollback* instantáneo. En caso de que se detecte un error crítico en producción (por ejemplo, después de la ejecución del *pipeline*), el Administrador puede revertir a la última versión estable del contenedor con un solo clic, sin requerir una nueva compilación de código.

12. CONCLUSIÓN

La implementación de AsisteTEC se ha planificado con un enfoque en la escalabilidad y la calidad del software. La adopción de una arquitectura de microservicios *serverless* en Google Cloud Run y Cloud Functions asegura la capacidad de manejar los picos de tráfico de los 15,000 alumnos de la UTEQ y sus profesores.

El plan de pruebas exhaustivo, que incluye 10 casos de prueba críticos y una cobertura superior al 80 %, se complementa con un robusto flujo de control de versiones Git Flow y un *pipeline* de Integración Continua (CI/CD) basado en GitHub Actions. Esta estrategia garantiza que solo el código probado y estable sea liberado a los entornos de `staging` y `producción`, cumpliendo con el objetivo de sustituir el sistema de asistencia manual por una solución digital eficiente y confiable.