Proyecto de Desarrollo - Lineamientos

Juan David Ocampo - 38402
David Rincón Toro - 20992
Kevin López Salazar - 28855
Sebastián Salazar Güiza - 906
Bryan Cartagena Hincapié - 5928

Presentado a:
Sandra Hurtado Gil
Ingeniería de Software 3

Universidad de Caldas
Facultad de Ingenierías e Inteligencia Artificial
Manizales, Colombia
2025

Tabla de Contenidos

1. Nombre del equipo:

ComiTeam

2. Integrantes del equipo:

Nombre de la persona	E-mail	Roles
Juan David Ocampo Gonzales	juan.ocampo38402@ucaldas.edu.co	Product Owner
Sebastián Salazar Güiza	sebastian.1701811396@ucaldas.edu.co	Arquitecto / Líder técnico
David Rincon Toro	david.rincon20992@ucaldas.edu.co	Lider de Calidad
Kevin López Salazar	Kevin.lopez28855@ucaldas.edu.co	Líder DevOps
Bryan Cartagena Hincapie	bryan.1701720431@ucaldas.edu.co	Gestor de Proyectos

3. Horarios de reunión:

a) Reunión de seguimiento

- *Martes de 5:00pm a 5:15pm*
- *Miércoles de 4:15pm a 4:30pm*
- *Viernes de 10:15am a 10:30am .*

b) Espacios adicionales para trabajo colaborativo

Horario	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
7-8						
8-9						
9-10						
10-11					X	X
11-12					X	X
12-1						
1-2						
2-3	X					
3-4	X					
4-5			X			X
5-6			X			X
6-7						X

4. Mecanismos de comunicación:

- Chat grupal por medio de Whatsapp para definir horarios disponibles para los miembros del grupo
- Reuniones por Google Meet para realizar y verificar las tareas que se tengan pendientes
- Tablero Team Planning en Github para tomar nota de las tareas y gestionar sus horarios y cuando se iniciaron y finalizaron

5. Estándares:

metodología GitFlow

El proyecto implementa la **metodología GitFlow** como estándar para la gestión de versiones y flujos de trabajo en Git.
Este modelo define una estructura de ramas que facilita el desarrollo colaborativo, la integración continua y el control de versiones estables.

Las ramas principales son:

- **main**: Contiene el código en producción, siempre en estado estable.
- **develop**: Rama base para la integración de nuevas funcionalidades.

Ramas de soporte:

- **feature/**: Se crean desde *develop* para desarrollar nuevas funcionalidades.
- **release/**: Preparan versiones estables antes del despliegue, derivadas de develop.
- **hotfix/**: Corrigen errores críticos directamente desde main.

Cada cambio sigue el flujo:
feature → develop → release → main
y, en caso de emergencia:
hotfix → main → develop.

Este estándar garantiza un proceso ordenado de desarrollo, reduce conflictos en la integración y asegura la trazabilidad de cada versión liberada.

Mensajes de commit: Conventional Commits

Se adopta el estándar **Conventional Commits** para mensajes de commit claros, consistentes y trazables. Cada commit debe seguir el formato:

<tipo>(<alcance>): <descripción breve en inglés>

[cuerpo opcional]

[footer opcional]

Tipos de commit permitidos:

Tipo	Descripción	Ejemplo

feat	Nueva funcionalidad	feat(auth): add user login with email
fix	Corrección de bug	fix(notifications): resolve push notification crash
docs	Cambios en documentación	docs(readme): update installation instructions
refactor	Refactorización sin cambiar funcionalidad	refactor(api): simplify food donation service
test	Adición o corrección de tests	test(auth): add unit tests for login service
build	Cambios en sistema de build o dependencias	build(deps): update react-native to 0.72

Reglas:

- Descripción en **inglés**, imperativo, minúsculas, sin punto final
- Máximo 72 caracteres en la primera línea
- El alcance (scope) es opcional pero recomendado
- Para cambios importantes (breaking changes), agregar BREAKING CHANGE: en el footer

Ejemplos válidos:

feat(donations): implement food listing with geolocation
fix(ui): resolve crash on empty food list
docs(api): add endpoint documentation for reservations
refactor(auth): migrate to firebase authentication v2

Nomenclatura de ramas

Ramas de features:

feature/descripcion-corta-en-ingles

Ejemplos: feature/user-registration, feature/push-notifications, feature/food-search

Ramas de releases:

release/vX.Y.Z

Ejemplos: release/v1.0.0, release/v1.1.0

Ramas de hotfixes:

hotfix/descripcion-del-problema

Ejemplos: hotfix/login-crash, hotfix/notification-error

Estándares de código

Lenguaje y formato:

- **JavaScript/TypeScript**: Seguir guía de estilo de Airbnb JavaScript Style Guide
- **Identación**: 2 espacios (no tabs)
- **Comillas**: Simples (') para strings
- **Punto y coma**: Obligatorio al final de statements
- **Herramienta de formato**: ESLint + Prettier configurados en el proyecto

Nomenclatura de variables y funciones:

- Variables y funciones: camelCase (ej: userName, getFoodList())
- Constantes globales: UPPER_SNAKE_CASE (ej: MAX_FOOD_ITEMS, API_BASE_URL)
- Componentes React: PascalCase (ej: FoodListScreen, DonationCard)
- Archivos de componentes: PascalCase.tsx (ej: LoginScreen.tsx)
- Archivos de servicios/utils: camelCase.ts (ej: authService.ts, dateUtils.ts)

Nombres en inglés:

- Todo el código (variables, funciones, clases, comentarios) debe estar en **inglés**
- Comentarios descriptivos en inglés para lógica compleja
- Documentación técnica en inglés; documentación de usuario en español

Testing

Cobertura mínima:

- 60% de cobertura de código en funciones críticas (autenticación, donaciones, notificaciones)

Nomenclatura de tests:

```
JavaScript
[nombre-archivo].test.js
Ejemplos: authService.test.js, FoodList.test.js
```

Estructura de tests:

```
JavaScript
javascript
describe('AuthService', () => {
  describe('login', () => {
    it('should authenticate user with valid credentials', () => {
      // test implementation
    });

    it('should throw error with invalid credentials', () => {
      // test implementation
    });
  });
});
```

Reglas de revisión:

- Todo PR debe ser revisado por al menos 1 compañero de equipo
- Los tests deben pasar antes de hacer merge
- No hacer merge de PRs con conflictos sin resolver

Gestión de dependencias

- Usar **npm** como gestor de paquetes
- Todas las dependencias deben especificarse en package.json
- Versiones fijadas en package-lock.json (commitearlo al repositorio)
- Revisar vulnerabilidades con npm audit antes de cada release
- Actualizar dependencias críticas de seguridad inmediatamente

Estructura de proyecto

Backend (Node.js + Express - Patrón MVC)

```
Shell
/backend
  /src
    /config            # Configuraciones (DB, Firebase, variables de entorno)
    /controllers       # Controladores - Lógica de negocio HTTP
    /models            # Modelos de datos y esquemas de BD
    /routes            # Definición de endpoints y rutas
    /middlewares       # Middlewares (autenticación, validación, errores)
    /services          # Lógica de negocio compleja y reutilizable
    /validators        # Validaciones de entrada de datos
    /utils             # Funciones utilitarias y helpers
    /constants         # Constantes globales del proyecto
  /tests               # Tests unitarios e integración
  /docs                # Documentación de API (Swagger)
  /scripts             # Scripts de utilidad (seeds, migrations)
  server.js            # Punto de entrada de la aplicación
  package.json
  .env.example
  README.md
```

Flujo MVC: Request → Routes → Middlewares → Controllers → Services → Models → Database

Frontend (React Native - Arquitectura por características)

```
Shell
/mobile
  /src
    /components        # Componentes reutilizables UI
      /common          # Botones, inputs, cards, modals
      /forms           # Componentes de formularios
      /layout          # Headers, containers, wrappers
      /feedback        # Alerts, toasts, loaders
    /screens           # Pantallas de la aplicación
      /auth            # Login, registro, recuperar contraseña
      /food            # Lista, detalle, agregar alimentos
      /donations       # Historial y detalles de donaciones
      /reservations    # Mis reservas
      /profile         # Perfil y configuración
      /map             # Mapa y geolocalización
      /notifications   # Notificaciones
    /navigation        # Configuración de navegación (React Navigation)
    /services          # Servicios de API y comunicación backend
    /store             # Estado global (Context API / Redux)
      /context
      /reducers
```

```
  /actions
 /hooks            # Custom React Hooks reutilizables
 /utils           # Funciones utilitarias
 /constants         # Constantes (colores, textos, configuraciones)
 /assets          # Imágenes, iconos, fuentes
 /config          # Configuración de la app (API URLs, Firebase)
/tests            # Tests de componentes y pantallas
App.jsx           # Punto de entrada
package.json
README.md
```

Principios de organización:

- Componentes genéricos y reutilizables en /components
- Pantallas específicas en /screens organizadas por feature
- Lógica de negocio separada en /services y /hooks
- Estado global centralizado en /store