Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Grupo 11

Alejandro Arguello Muñoz
Juan Luis Vergara Novoa
Steven David Alfonso Galindo
Daniel Santiago Delgado Pinilla

Taller requerimientos y clean code para proyecto

El objetivo principal del proyecto es aplicar los conceptos vistos en los módulo de requerimientos y clean code, llevando a cabo un proceso integral para **documentar** las necesidades de un sistema de software. Este ejercicio está diseñado para fortalecer sus habilidades en la conceptualización, análisis y diseño inicial de sistemas de software.
Al finalizar el taller, ustedes deben tener avanzado el MVP del proyecto final puesto que entramos a la fase de testing en 2 semanas (primera semana de febrero).

Esta entrega se hará en dos fases, la primera es el 31 de Enero a las 23:59, esta será preliminar, es decir, aunque no habrá nota deben entregar todo lo que hayan desarrollado hasta ahora. Es obligatoria, quienes NO la hagan la nota de los dos módulos será evaluada sobre 4.0, les daré retroalimentación y tendrán 1 semana para corregirlo, el día 7 de febrero a las 23:59 será la entrega que tendrá su calificación.

La estructura de entrega es la siguiente:

La mayor parte de la información del taller debe estar centralizada en un único documento PDF llamado Proyecto_Final.pdf, que contendrá los puntos principales del proyecto (detallados más abajo). Sin embargo, algunos puntos requieren archivos adicionales que deben incluirse en carpetas específicas.

```
/Raiz..
  |-- README.md                 <<-- Archivo de presentación del proyecto.
  |-- .gitignore                <<-- Configuración para evitar subir archivos innecesarios.
  |-- Proyecto/                 <<-- Carpeta con el código fuente del proyecto.
  |    |-- (código fuente)
  |
  |-- Documentación/            <<-- Carpeta principal de la documentación.
       |-- Proyecto_Final.pdf   <<-- Documento central del proyecto.
       |-- Casos_de_Uso/        <<-- Carpeta con los casos de uso individuales.
       |     |-- Caso_de_uso_<nickname>_01.pdf
       |     |-- Caso_de_uso_<nickname>_02.pdf
       |
       |-- Historias_de_Usuario/    <<-- Carpeta con historias de usuario individuales.
       |     |-- historia_de_usuario_<nickname>_01.pdf
       |     |-- historia_de_usuario_<nickname>_02.pdf
       |
       |-- Diagramas/           <<-- Carpeta con todos los diagramas del sistema.
       |     |-- Diagrama_Arquitectura.pdf
       |     |-- Modelo_Base_Datos.pdf
       |     |-- Diagrama_Casos_de_Uso.pdf
       |
       |-- Clean_Code/          <<-- Carpeta con el informe de código limpio.
       |     |-- Informe_codigo_limpio.pdf
```

**<nickname> hace referencia a su usuario unal, por ejemplo el mio es “oalvarezr” por lo que los archivos quedan: Caso_de_uso_oalvarezr_01.pdf

Punto	Ubicación de calificable
Punto 1	/README.md
Punto 2	Documentación/Proyecto_Final.pdf
Punto 3	Documentación/Proyecto_Final.pdf
Punto 4	Documentación/Proyecto_Final.pdf
Punto 5	Documentación/Casos_de_Uso/* Documentación/Diagramas/Diagrama_casos_de_uso
Punto 6	Documentación/Historias_de_Usuario/*
Punto 7	Documentación/Clean_Code/Informe_codigo_limpio.

	pdf
Punto 8	Documentación/Proyecto_Final.pdf Documentación/Diagramas/*
Punto 9	Documentación/Proyecto_Final.pdf

Punto 1: Pre Requerimientos

SIN ESTE PUNTO NO SE REVISARÁN LOS SIGUIENTES.

En el archivo **README.md** del repositorio, incluyan lo siguiente:

1. **Título del repositorio:**
    - Ejemplo: Repositorio grupal - Ingeniería de Software 1 - 2024-2 Grupo #.
2. **Nombres de los integrantes del equipo.**
3. **Nombre del proyecto.**
4. **Icono o logo del proyecto (opcional).**
5. **Descripción del objetivo del proyecto:**
   Descripción de lo que trata el proyecto.
6. **Tecnologías a utilizadas:**
    - Lenguajes de programación.
    - Frameworks.
    - Servicios externos (bases de datos, servidores, APIs).
7. **Archivo .gitignore:**
   implementar el **.gitignore**, les recomiendo esta pagina 🙂
   https://www.toptal.com/developers/gitignore

Punto 2: Levantamiento de requerimientos

Este punto se enfoca en identificar y documentar las necesidades del proyecto desde sus fundamentos. Para estructurar este apartado, utilicen las siguientes preguntas orientadoras a para redactar su texto a manera de narración:

- **¿De dónde surge la idea?**
  Explicar cómo identificaron la necesidad del sistema. ¿Fue por observación, entrevistas, investigación o una combinación de estas? ¿Qué problemática busca resolver el software?
- **¿Cómo se pusieron de acuerdo para elegir la idea?**
  Describan el proceso de consenso dentro del equipo: reuniones, votaciones, asignación de roles, etc.
- **¿Cuáles son las problemáticas que buscan resolver?**
  Contextualicen los problemas que enfrenta el público objetivo. Justifiquen la relevancia de abordarlos con un sistema de software.
- **¿Qué expectativas tienen los usuarios potenciales del sistema?**
  Identifiquen las principales funcionalidades esperadas y los beneficios que el sistema debe proporcionar.
- **¿Qué beneficios esperan ustedes al desarrollar este proyecto?**
  Reflexionen sobre lo que cada integrante espera aprender y cómo el proyecto contribuye a su formación profesional.

El levantamiento debe hablar muy brevemente sobre las funcionalidades, identificarlas es el primer paso y estas deben estar conectadas con las problemáticas mencionadas.

Punto 3: Análisis de requerimientos

Realicen un análisis detallado de cada funcionalidad utilizando el método **MoSCoW (Must, Should, Could, Won't)**. Además, para cada funcionalidad, añadan una **estimación de esfuerzo con la secuencia de Fibonacci** (1, 2, 3, 5, 8, 13, etc.). Argumenten por qué llegaron a cada estimación considerando factores como:

- Complejidad técnica.
- Recursos disponibles.
- Impacto en la experiencia del usuario.
- Relación con los objetivos del proyecto.

NOTA: utilice tablas, gráficos, alguna técnica para organizar la información, piensen que le van a presentar esto a alguien que los quiere contratar…

Punto 4: Análisis gestión de software

Analicen la **triada de gestión de proyectos** para su proyecto. Investigar y documentar lo siguiente:

- **Tiempo:**
  ¿Cuánto tiempo estiman que tomará el desarrollo de cada funcionalidad o módulo? Detallen cómo se dividirán las etapas del desarrollo (por ejemplo, diseño, desarrollo, pruebas).
- **Costo:**
  Investigar cuánto cuesta desarrollar software considerando factores como:
  - Sueldos de desarrolladores (junior, senior, arquitectos, ux, testers).
  - Licencias, servicios en la nube, bases de datos.
  - Herramientas externas (APIs, frameworks, software de desarrollo).
  - Infraestructura (servidores, almacenamiento).

  Saquen un total en una tabla organizada partiendo en que solo mostrarán lo que necesiten dentro de los costos (por ejemplo solo ocupamos un desarrollador junior)

- **Alcance:**
  Definan los límites del sistema. ¿Qué características o funcionalidades están incluidas y cuáles quedan fuera del alcance del MVP? Sean claros para evitar ambigüedades en futuras iteraciones del proyecto.

NOTA: Aunque esto no lo vimos en clase, lo que quiero que intenten es que investiguen en internet cuánto vale realmente su trabajo, esto lo deben siempre en el mundo laboral real, es la intención del ejercicio, no usen IA, o terminarán regalandose por 50k a la semana (?).

Punto 5: Casos de uso del sistema

Elaboren casos de uso exclusivamente para las funcionalidades clasificadas como **MUST** en el análisis MoSCoW. Cada estudiante deberá realizar un mínimo de **tres casos de uso** es decir, si no son suficientes los MUST, utilicen los SHOULD.

Documenten los casos de uso con los siguientes elementos como mínimo:

- Nombre del caso de uso.
- Descripción breve.
- Actor(es) involucrado(s).
- Flujo principal.
- Flujos alternativos (si aplica).
- Precondiciones y postcondiciones (si aplica).

Además, complementan los casos de uso con su diagrama (ES SÓLO UNO POR EL SISTEMA), para ambos grupos su diagrama tiene que verse parecido a esto… URL aquellos que se equivocaron en el taller, espero lo corrijan, los revisaré con detalle 🧐🧐

NOTA: Sé que anteriormente había dicho que todas las funcionalidades (MoSCoW) pero hubo grupos que han hecho demasiados casos de uso, la idea es que aprendan a como hacerlos, no ahogarlos en repetir el proceso, a los que hayan avanzado y hayan hecho más de los que les pido, me excuso desde ya.

Punto 6: Historias de usuario

Para cada caso de uso generado en el punto anterior, cada estudiante debe crear su respectiva **historia de usuario**, con un mínimo de **tres historias por persona**. Utilicen la plantilla proporcionada. Hasta ahora, todos utilizan correctamente la plantilla.

Gran parte del texto escrito durante el taller fue mejorado con ayuda de IA (Principalmente con DeepSeek y modelos de Open IA)

Punto 2

Inicialmente, no teníamos ni idea de qué proyecto realizar. Para ser sinceros, no queríamos hacer algo muy complejo, principalmente por experiencias previas en otras asignaturas, donde por cuestiones de tiempo siempre terminamos realizando las cosas sobre el tiempo. Para contextualizar cómo surgió esta idea, al principio formamos un grupo de tres integrantes que ya nos conocíamos por algunas asignaturas anteriores y en general, porque habíamos ingresado a la carrera al mismo tiempo. Como ya mencionamos, ninguno de nuestros proyectos anteriores nos entusiasmaba y buscábamos en general queríamos realizar algo diferente, algo que nos pudiera motivar sin complicarnos mucho la vida. En realidad, no exploramos muchas posibilidades, simplemente terminamos evaluando dos opciones.

La primera idea consistía en desarrollar una aplicación para el manejo y control de parqueaderos. Su origen se remonta a que uno de los miembros, un día sin nada que hacer, se puso a vaguear por Google Maps y, al leer reseñas de distintos centros comerciales, notó que una queja recurrente se refería al manejo de los parqueaderos (aunque, como el autor de esta idea y persona que está escribiendo esto, no recuerdo con exactitud qué decían). La otra idea en la que se fundamenta nuestro proyecto fue propuesta por otro integrante del grupo; un día, mientras se disponía a botar la basura, se percató de que no sabía en qué contenedor debía hacerlo, dejando finalmente dicha decisión en manos del azar. Por otro lado, el tercer miembro nunca quiso aportar una idea >:(.

La elección de cuál idea desarrollar se basó en lograr un consenso entre los tres. Aunque la idea del parqueadero no nos disgustaba, consideramos que la aplicación relacionada con la basura nos permitiría adaptarla más a nuestro estilo. Además, se nos ocurrían más funcionalidades para esa propuesta que para la otra, probablemente debido a nuestro desconocimiento en el tema, y quizá porque trabajar en la idea del parqueadero podría terminar afectando más nuestra salud mental.

Probablemente, a estas alturas de la historia, se estará preguntando porque no hemos mencionado al cuarto integrante, o tal vez no. En todo caso, se trata de una persona que conocimos este semestre, en un acto casi irresponsable de nuestra parte, jamás socializamos de manera amplia la idea del proyecto entre los cuatro. Sin embargo, a él no le pareció mala la idea 👍.

Aunque no hemos investigado estudios o estadísticas, basándonos en nuestra experiencia empírica, desde el colegio, se nos enseña a ser responsables con lo que desechamos. Sin embargo, en la práctica, muchas personas desconocen dónde depositar correctamente la basura. Hemos observado que, para concientizar e informar, se utilizan centros educativos, talleres y medios de comunicación, sin embargo, jamás hemos visto (o escuchado, probablemente por nuestra nula investigación) estrategias que hagan un gran uso de la tecnología. En pocas palabras, si tuviéramos que especificar la problemática que queremos abordar, se trataría de facilitar el acceso a la información sobre el desecho de residuos al público en general, mediante una aplicación que, además de simplificar este proceso, motive a las personas a ser más responsables con sus desechos, como por ejemplo con un sistema de rachas como Duolingo (Si no es muy difícil de hacer xd).

Queremos que la aplicación sea sencilla de usar para todo tipo de público, de modo que desde un niño hasta una persona mayor pueda manejarla sin complicaciones. Además, esperamos que los usuarios sientan que aprenden algo nuevo cada vez que la utilizan.

Entre las funcionalidades, algo que teníamos claro desde el principio es que queríamos usar ia, por lo que el escáner debe ser capaz de identificar la categoría de basura a la que pertenece haciendo uso de la inteligencia artificial. No solo queremos que los usuarios sepan dónde depositar la basura, sino también que, al escanear un objeto, reciban información básica sobre el tipo de residuo y el motivo por el que debe ir en ese contenedor.

Otro aspecto importante es que la información no se pierda al cerrar la aplicación. Por ello, consideramos esencial contar con un registro de los últimos desechos escaneados, de forma que se pueda acceder a esta información en cualquier momento o por lo menos durante cierto periodo de tiempo.

Finalmente, teniendo en cuenta que no todas las personas disponen de conexión a internet en todo momento, nuestra intención es que la aplicación funcione también sin conexión, aunque esto implique la pérdida de algunas funcionalidades.

Probablemente, a estas alturas, la aplicación, pese a contar con una premisa muy bonita, lo cierto es que probablemente muy poca gente la descargaría (ni siquiera nosotros jajaja). Por ello, hemos pensado en incorporar funcionalidades adicionales que la hagan más llamativa. Nos gustaría incluir un sistema de rachas, desafíos y estadísticas que incentiven a los usuarios a ser más responsables con sus desechos. Aunque esto sería algo mucho más opcional. Si bien el objetivo principal de la aplicación es ofrecer un acceso sencillo a la información sobre la clasificación de residuos, contribuyendo así al cuidado del medio ambiente, también nos interesa hacerlo de una forma entretenida. De esta manera, la app no solo informará, sino que también motivará y divertirá a sus usuarios.

Hablando de lo que esperamos obtener al elaborar este proyecto, como grupo consideramos que esta asignatura es fundamental para nuestra carrera. Aunque en cursos anteriores hemos trabajado en distintos proyectos, este es, sin duda, el que posee un mayor detalle técnico en el que ha participado cualquiera de los cuatro. Por ello, más allá de aprender a utilizar las tecnologías que aplicaremos, lo que más nos interesa es comprender en profundidad la estructura que debe tener un proyecto de este tipo y todo lo que ello implica. Además claro, esperamos ampliar nuestros conocimientos sobre la clasificación de residuos :).

(El texto en general fue mejorado con IA)

Requerimientos Funcionales

A continuación, se detallan los requerimientos funcionales identificados para EcoScan, organizados por prioridad y asociados a procesos específicos dentro del software:

Escáner Inteligente

- **Descripción:** Permite a los usuarios tomar una foto de un objeto para que la aplicación lo clasifique automáticamente en la categoría correcta (reciclaje, orgánico, basura general, etc.).
- **Proceso Asociado:** Procesamiento de imágenes y clasificación automática.
- **Especificaciones Técnicas:**
    - Integración de tecnología de visión por computadora.
    - Algoritmos de aprendizaje automático para mejorar la precisión del reconocimiento.

Historial de Escaneos

- **Descripción:** Permite a los usuarios registrados guardar un historial de los objetos escaneados para evaluar sus hábitos de reciclaje y recibir consejos personalizados.
- **Proceso Asociado:** Almacenamiento y gestión de datos de usuario.
- **Especificaciones Técnicas:**
    - Interfaz de usuario para visualizar el historial.

Registro Opcional de Usuarios (Opcional)

- **Descripción:** Los usuarios pueden registrarse para acceder a funcionalidades adicionales como el historial de escaneos.
- **Proceso Asociado:** Gestión de autenticación y perfiles de usuario.
- **Especificaciones Técnicas:**
    - Integración con servicios de autenticación (correo electrónico, redes sociales).
    - Seguridad en el almacenamiento de credenciales.

Recordatorios y Desafíos Ambientales (Opcional)

- **Descripción:** Envía recordatorios de reciclaje y propone desafíos ambientales, como "Recicla 10 botellas esta semana".
- **Proceso Asociado:** Gestión de notificaciones y motivación del usuario.
- **Especificaciones Técnicas:**
    - Sistema de notificaciones push.
    - Módulo para crear y gestionar desafíos.

Reconocimiento de Materiales con Consejos Específicos

- **Descripción:** Además de clasificar un objeto, detalla el material del que está hecho y ofrece consejos específicos para su limpieza o preparación para reciclaje.
- **Proceso Asociado:** Análisis detallado de materiales.
- **Especificaciones Técnicas:**
    - Base de datos de materiales y sus características.
    - Algoritmos para identificar y asociar materiales con consejos específicos.

Racha de Usuario (Opcional)

- **Descripción:** Implementa un sistema de puntos basado en la cantidad de objetos escaneados correctamente y se acumula cada día pero se pierde tras varios días sin uso.
- **Proceso Asociado:** Gamificación y motivación del usuario.
- **Especificaciones Técnicas:**
    - Interfaz para visualizar la racha.

Notificaciones Educativas

- **Descripción:** Envía notificaciones con tips ambientales, datos curiosos o recordatorios de hábitos sostenibles.
- **Proceso Asociado:** Distribución de contenido educativo mediante notificaciones.
- **Especificaciones Técnicas:**
    - Módulo de gestión de notificaciones.

Detección de Contaminación Cruzada(opcional)

- **Descripción:** Advierte sobre materiales que contaminan el reciclaje y sugiere cómo limpiarlos antes de desecharlos.
- **Proceso Asociado:** Análisis de calidad de residuos.
- **Especificaciones Técnicas:**
    - Algoritmos para detectar residuos contaminados.
    - Sistema de recomendaciones para limpieza.

Personalización del Perfil (para Usuarios Registrados)

- **Descripción:** Permite a los usuarios establecer objetivos ambientales personalizados, como reducir la producción de plástico.
- **Proceso Asociado:** Gestión de perfiles y objetivos de usuario.
- **Especificaciones Técnicas:**
    - Interfaz de usuario para establecer y modificar objetivos.
    - Almacenamiento seguro de preferencias de usuario.

Estadísticas Ambientales (Opcional)

- **Descripción:** Muestra estadísticas sobre el impacto de las acciones de reciclaje de los usuarios, así como estadísticas globales.
- **Proceso Asociado:** Análisis y visualización de datos.
- **Especificaciones Técnicas:**
    - Módulo de recopilación y procesamiento de datos.
    - Herramientas de visualización gráfica para estadísticas.

Reconocimiento Sin Conexión

- **Descripción:** Permite realizar escaneos básicos sin necesidad de conexión a internet, almacenando los datos para sincronizarse posteriormente.
- **Proceso Asociado:** Operación offline y sincronización de datos.
- **Especificaciones Técnicas:**
    - Módulo de reconocimiento local con capacidad limitada.
    - Sistema de almacenamiento temporal y sincronización en la nube.

Integración con redes sociales (Opcional)

- **Descripción:** Permite a los usuarios compartir sus logros de reciclaje, escaneos y estadísticas ambientales en redes sociales (Facebook, Twitter, Instagram, etc.), fomentando la creación de una comunidad en torno a prácticas sostenibles.
- **Proceso Asociado:** Social Sharing y difusión de información.
- **Especificaciones Técnicas:**
    - Integración con las APIs de las principales redes sociales para compartir contenido.
    - Interfaz de usuario para personalizar mensajes y seleccionar imágenes o datos a compartir.
    - Opciones para compartir de manera directa o programada (por ejemplo, al alcanzar un hito de reciclaje).

Requerimientos No Funcionales

Accesibilidad y Usabilidad

- **Descripción:** La plataforma debe ser accesible desde dispositivos móviles, con una interfaz intuitiva y adaptada para usuarios con poca experiencia tecnológica.
- **Especificaciones Técnicas:**
    - Diseño responsivo compatible con Android.

Disponibilidad y Rendimiento

- **Descripción:** La aplicación debe funcionar predominantemente online, con opciones básicas disponibles offline y tiempos de carga razonables para una experiencia fluida.
- **Especificaciones Técnicas:**
    - Infraestructura en la nube para alta disponibilidad.
    - Optimización de recursos para minimizar tiempos de carga.

Personalización y Diseño

- **Descripción:** Diseño visual atractivo que refleje una identidad ambiental, utilizando una paleta de colores naturales y evitando elementos molestos para el usuario.
- **Especificaciones Técnicas:**
    - Uso de colores y símbolos universales para la clasificación de residuos.
    - Diseño UI/UX centrado en la simplicidad y la facilidad de uso.

Punto 3 Análisis de requerimientos

Funcionalidad	Prioridad	Estimación (Fibonacci)	Justificación
Escáner Inteligente	Must Have	13	Es la funcionalidad principal que permite el reconocimiento automático de residuos. Su desarrollo implica integrar modelos de machine learning para el procesamiento de imágenes.
Historial de Escaneos	Must Have	5	Permite a los usuarios llevar un registro de sus escaneos y evaluar sus hábitos de reciclaje. Requiere una interfaz de visualización y una conexión estable con la base de datos.
Registro de Usuarios	Could Have	5	Facilita el acceso a funcionalidades adicionales (como historial y personalización), pero no es indispensable para la operatividad básica. Se apoya en formularios e integración con servicios de autenticación.
Recordatorios y Desafíos Ambientales	Could Have	8	Incentiva al usuario a mantener hábitos de reciclaje mediante notificaciones y retos. Su implementación requiere integrar notificaciones push y un módulo para gestionar desafíos, aumentando el compromiso.
Reconocimiento de Materiales	Must Have	8	Complementa el escáner identificando el material del objeto y proporcionando consejos específicos. Esto mejora la utilidad de la app al ofrecer información detallada para reciclar.

Rachas de Usuarios	Could Have	8	Funcionalidad de gamificación que premia el uso continuo de la aplicación. Aunque añade valor y motivación, no es esencial para la operatividad central del sistema.
Notificaciones Educativas	Should Have	3	Envía tips y datos ambientales que fomentan hábitos sostenibles. Es de baja complejidad, ya que se basa en contenido predefinido y notificaciones push, pero mejora el engagement del usuario.
Detección de Contaminación Cruzada	Should Have	8	Permite identificar residuos que puedan contaminar el reciclaje y recomienda acciones de limpieza. Requiere algoritmos adicionales para el análisis, aportando precisión a la clasificación. Si bien puede hacer la aplicación mucho más útil, puede requerir bastante complejidad adicional.
Personalización del Perfil	Should Have	5	Brinda la posibilidad de configurar objetivos y preferencias personales. Su implementación implica una interfaz adicional y el almacenamiento seguro de datos, mejorando la experiencia personalizada.
Estadísticas Ambientales	Could Have	8	Muestra el impacto de las acciones de reciclaje a nivel individual y global. Requiere el procesamiento y visualización de datos, lo que aporta valor motivacional pero puede en otras versiones.

Reconocimiento Sin Conexión	Could Have	8	Permite realizar escaneos sin conexión a Internet, almacenando los datos localmente y sincronizarlos posteriormente. Es útil para áreas con conectividad limitada,pero puede incrementar la complejidad de la app.
Integración con redes sociales	Won't Have	-	Aunque puede favorecer la difusión y el engagement, no es prioritaria en el alcance actual del proyecto y se podría considerar para futuras versiones.

Punto 4: Análisis gestión de software

Estimación de Recursos

Equipo

- **Desarrollador Backend:**
  Encargado de implementar la lógica de procesamiento de imágenes, integración de tecnologías, comunicación con la base de datos y servicios externos.
- **Desarrollador Frontend:**
  Responsable del desarrollo de la aplicación móvil, integración de las distintas interfaces y manejo de notificaciones.
- **Product Manager / Analista (rol dual):**
  Supervisa la integración de requerimientos, la calidad del código y la comunicación con el cliente. Este rol puede ser compartido o dual, colaborando estrechamente con los desarrolladores.

Duración Total del Desarrollo

Se estima que el proyecto se desarrollará en **aproximadamente 30 días**, distribuidos de la siguiente forma:

- **Investigación y Diseño (1 semana):**
  Definición de la arquitectura, elaboración de prototipos y validación de requerimientos.
- **Desarrollo e Integración (3 semanas):**
  Implementación de las funcionalidades principales identificadas para contar con un MVP.
- **Pruebas y Ajustes (1 semana):**
  Ejecución de pruebas funcionales, ajustes en la interfaz y optimización del rendimiento.

Estimación de Costos

Basado en información actual de plataformas laborales en Colombia y considerando un equipo de 4 personas trabajando durante 30 días, la estimación es la siguiente:

Recurso	Cantidad	Costo Unitario (COP)	Total Estimado
Desarrollador Backend	1	COP 150,000/día	COP 4,500,000
Desarrollador Frontend	1	COP 150,000/día	COP 4,500,000
Product Manager / Analista / QA	2	COP 150,000/día	COP 4,500,000

Alternativas de Reducción de Costos

- **Reutilización de Código y Componentes Open Source:**
  - Uso de plantillas y módulos ya desarrollados para Python, TensorFlow y React, lo que reduce el tiempo de desarrollo.
  - Aplicación de técnicas de Transfer Learning.
- **Priorización de Funcionalidades (MVP):**
  - Implementación inicial únicamente de las funcionalidades **Must Have**.
  - Las funcionalidades opcionales se desarrollarán en iteraciones futuras.
- **Roles Dualizados:**
  - Aprovechar la versatilidad del equipo (por ejemplo, un desarrollador Fullstack que pueda asumir tareas tanto de backend como de frontend) para optimizar recursos.

Tecnologías Propuestas

Backend

- **Lenguaje y Frameworks:**
  Python con TensorFlow y OpenCV para el procesamiento de imágenes y el reconocimiento de materiales.
- **APIs y Servicios:**
  Integración con Firebase Cloud Messaging para notificaciones y Firebase Auth para autenticación.

Frontend

- **Framework:**
  Flutter para el desarrollo de una aplicación móvil multiplataforma (Android, y opcionalmente iOS).
- **UI/UX:**
  Diseño responsivo e intuitivo, basado en las últimas tendencias para aplicaciones móviles.

Base de Datos y Servicios en la Nube

- **Firebase:**
  Para autenticación, almacenamiento de datos (historial de escaneos, perfiles de usuario) y mensajería.
- **Infraestructura:**
  Servidores en la nube (por ejemplo, Google Cloud Platform o AWS) que aseguren alta disponibilidad y escalabilidad.

Alcance

Incluido en el MVP

- **Escáner Inteligente:**
  Permite capturar una imagen y clasificar automáticamente en categorías (reciclaje, orgánico, basura general, etc.).
- **Historial de Escaneos:**
  Registro de actividades que permite a los usuarios evaluar sus hábitos de reciclaje.
- **Registro de Usuarios:**
  Permite la creación de perfiles mediante autenticación con correo electrónico o redes sociales.

Fuera del alcance y para versiones futuras

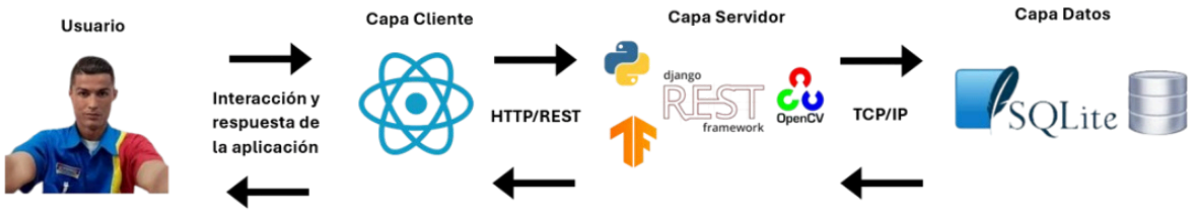
- Funcionalidades opcionales como:
  - Recordatorios y Desafíos Ambientales
  - Racha de Usuarios
  - Notificaciones Educativas Avanzadas
  - Detección de Contaminación Cruzada
  - Personalización del Perfil
  - Estadísticas Ambientales

Consideraciones Adicionales

- **Arquitectura Escalable y Modular:**
  Se debe diseñar la arquitectura de forma que permita la integración de nuevas funcionalidades en el futuro.
- **Experiencia de Usuario Óptima:**
  Priorizar tiempos de carga reducidos y un diseño intuitivo que ofrezca una experiencia de usuario de alta calidad.

Punto 8: Diseño y arquitectura

Para el desarrollo del proyecto adoptamos una arquitectura cliente-servidor, distribuyendo sus componentes en tres capas: el cliente (frontend), el servidor (backend) y la base de datos.



Capa de Cliente

En esta capa se desarrollara la aplicación móvil con React Native, ofreciendo al usuario una interfaz amigable para capturar imágenes y visualizar la información en torno a las imágenes escaneadas. Las peticiones se realizarán a través de HTTP/REST hacia el servidor, lo que nos permitirá mantener la lógica de la aplicación independiente de su diseño.

Capa de Servidor

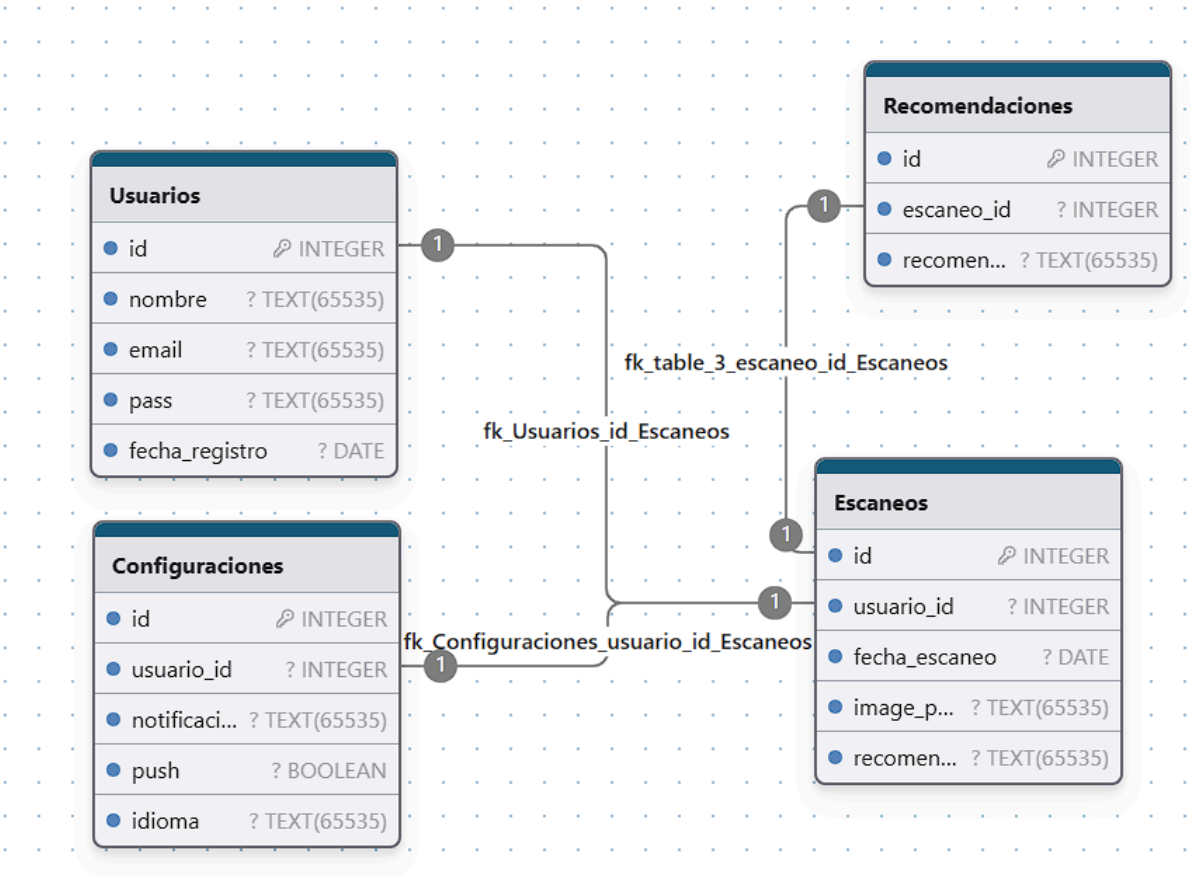
El backend se implementará con Django y Django REST Framework, integrando la lógica de la aplicación y algún modelo de reconocimiento usando TensorFlow u OpenCV. Este servidor recibirá las solicitudes del cliente, procesando la información, por ejemplo, identificando el tipo de residuo y gestionando la comunicación con la base de datos, asegurando la coherencia de los datos y aplicando reglas de autenticación cuando sea necesario.

Capa de Datos

Se empleara SQLite como la base de datos, almacenando registros de usuarios, historial de escaneos y configuraciones. La interacción con la capa de datos se realiza exclusivamente desde el servidor, reforzando la seguridad y la integridad de la información. Sin embargo, en caso de continuar con el desarrollo de la aplicación en un futuro no descartamos usar una base de datos que permita una mejor escalabilidad y concurrencia.

En general esta arquitectura de tres capas brinda flexibilidad para futuras mejoras, como la migración a un motor de base de datos más robusto o la ampliación de los servicios de inteligencia artificial. Además nos asegura una clara división de tareas entre la creación de la interfaz de usuario, la lógica de clasificación y el almacenamiento de la información, facilitando la distribución de tareas entre los miembros del equipo y un desarrollo continuo de la aplicación.

Diseño de base de datos



La base de datos cuenta con 5 tablas principales, donde cada una tiene una llave primaria como identificador. Cada que se registre un nuevo elemento en la tabla, Django asigna una llave automáticamente.

Las tablas son:

1. Usuarios: Almacena los datos de los usuarios registrados en la aplicación.

```
CREATE TABLE Usuarios (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nombre TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        pass TEXT NOT NULL
        fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Se conecta con Escaneos y Configuraciones a través de usuario_id.

2. Escaneos: Guarda los resultados de cada escaneo realizado por un usuario.

```
CREATE TABLE Escaneos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        usuario_id INTEGER NOT NULL,
        fecha_escaneo TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        imagen_path TEXT NOT NULL,
        recomendacion TEXT NOT NULL,
        FOREIGN KEY (usuario_id) REFERENCES Usuarios(id) ON DELETE CASCADE
);
```

Relacionado con Usuarios (cada usuario tiene múltiples escaneos).
Relacionado con Materiales (cada escaneo puede identificar varios materiales).

3. Configuraciones: Permite que cada usuario personalice su experiencia en la aplicación.

```
CREATE TABLE Configuraciones (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        usuario_id INTEGER NOT NULL,
        notificaciones TEXT DEFAULT,
        push BOOLEAN DEFAULT,
        idioma TEXT DEFAULT,
        FOREIGN KEY (usuario_id) REFERENCES Usuarios(id) ON DELETE CASCADE
);
```

Relacionado con Usuarios (cada usuario tiene una única configuración).

4. Recomendaciones: Ofrece sugerencias personalizadas según el análisis del escaneo.

```
CREATE TABLE Recomendaciones (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        escaneo_id INTEGER NOT NULL,
        recomendacion TEXT NOT NULL,
        FOREIGN KEY (escaneo_id) REFERENCES Escaneos(id) ON DELETE CASCADE
);
```

Relacionado con Escaneos (cada escaneo genera una o más recomendaciones).

Se escogió SQLite debido a que es ligero y eficiente, fácil de integrar con Django (es la base de datos por defecto de Django), tiene soporte para relaciones y llaves foráneas lo que permite una estructura bien organizada y también permite el modo offline para escanear cuando no haya conexión.

Punto 9: Patrones de diseño:

1. Arquitectura General: Separación de Capas

El proyecto sigue una arquitectura cliente-servidor con una clara separación entre:

Frontend (React Native): Encargado de la interfaz de usuario, lógica de presentación y gestión del estado.

Backend (Django): Gestiona la lógica de negocio, acceso a datos y exposición de APIs REST.

2. Patrón en Django: MTV (Model-Template-View) Adaptado a API REST

Aunque Django usa tradicionalmente MTV, en este proyecto se prioriza su rol como backend API, eliminando la capa de "Template".

Model (Modelo): Define la estructura de la base de datos (ej: models.py).

Ejemplo: class Reciclaje(models.Model): ....

View (Vista): Maneja solicitudes HTTP y devuelve respuestas JSON (usando Django REST Framework).

Ejemplo: class EscaneoView(APIView): ....

URLs (Enrutador): Define los endpoints de la API (ej: path('api/escaneos/', EscaneoView.as_view())).

Beneficios:

Separación clara entre lógica de negocio y datos.

Reusabilidad: Las APIs pueden ser consumidas por múltiples clientes (móvil, web, etc.).

3. Patrón en React Native: Componentes + Gestión de Estado (Flux/Redux)

React Native no sigue un patrón estricto, pero combina:

Componentes (Vista):

Encargados de renderizar la interfaz (ej: <EscaneoList />).

Usan props para recibir datos y state para manejar cambios internos.

Gestión de Estado Global (Redux - Patrón Flux):

Store (Almacén): Fuente única de verdad para el estado de la app (ej: historial de escaneos).

Actions (Acciones): Describen eventos (ej: { type: 'ADD_ESCANEO', payload: data }).

Reducers (Reductores): Actualizan el estado basado en las acciones.

Beneficios:

Reactividad: Los componentes se actualizan automáticamente ante cambios en el estado.

Centralización: Facilita el acceso a datos globales (ej: perfil del usuario).

4. Comunicación Frontend-Backend: API REST

Django (Backend):

Expone endpoints como GET /api/escaneos o POST /api/desafios.

Usa serializadores (serializers.py) para convertir datos entre JSON y modelos de Django.

React Native (Frontend):

Consume APIs usando fetch o librerías como axios.

Ejemplo:

javascript

```javascript
const response = await axios.get('http://localhost:8080/api/escaneos');
dispatch({ type: 'LOAD_ESCANEOS', payload: response.data });
```

Beneficios:

Independencia: El frontend y backend pueden evolucionar por separado.

Escalabilidad: Fácil integración con servicios externos (ej: Firebase para notificaciones).

5. Patrones Adicionales

a. Inversión de Dependencias (Dependency Injection):

React Native: Librerías como react-navigation inyectan dependencias (ej: navigation como prop).

Django: Uso de settings.py para configurar conexiones a bases de datos o servicios externos.

b. Singleton (Acceso Único):

Django:Conexión a la base de datos (ej: una única instancia de django.db.connection).

React Native: Instancia global de Redux Store o cliente de API.

c. Observer (Suscripción a Eventos):

React Native: Suscripción a cambios de estado en Redux (store.subscribe()).

Django: Señales (signals.py) para ejecutar acciones tras eventos (ej: guardar un escaneo).

1. Backend (Django - Patrón MTV Adaptado)

principales clases

```
@startuml



'------------------ Modelo ------------------

class Model {

  + fields: CharField, IntegerField, etc.

  + save()

  + delete()

}



class EscaneoModel {

  + usuario: ForeignKey(User)

  + material: CharField

  + fecha: DateTimeField

}
```

```
'------------------ Vista (API) ------------------

class View {

  + handle_request()

}



class EscaneoAPIView {

  + get(): Lista de escaneos (JSON)

  + post(): Crear escaneo (JSON)

}



'------------------ Serializador ------------------

class Serializer {

  + to_representation()

  + to_internal_value()

}



class EscaneoSerializer {

  + model = EscaneoModel

  + fields = ['usuario', 'material', 'fecha']

}



'------------------ URLs ------------------

class URLPatterns {

  + path('api/escaneos/', EscaneoAPIView.as_view())

}
```

```
'------------------ Relaciones ------------------

EscaneoModel --|> Model

EscaneoAPIView --|> View

EscaneoAPIView --> EscaneoSerializer

EscaneoSerializer --> EscaneoModel

URLPatterns --> EscaneoAPIView


@enduml
```

2. Frontend (React Native - Patrón Flux/Redux)

```
@startuml


'------------------ Componente ------------------

class EscaneoList {

  + state: { escaneos: Array }

  + useEffect()

  + render()

}


'------------------ Acciones ------------------

class Actions {

  + LOAD_ESCANEOS: "LOAD_ESCANEOS"

  + ADD_ESCANEO: "ADD_ESCANEO"

}
```

```
'------------------ Reducer ------------------
class EscaneoReducer {
  + state: { escaneos: Array }
  + action: { type, payload }
  + reduce()
}

'------------------ Store ------------------
class Store {
  + getState()
  + dispatch()
  + subscribe()
}

'------------------ API Client ------------------
class APIClient {
  + get(url): Promise
  + post(url, data): Promise
}

'------------------ Relaciones ------------------
EscaneoList --> APIClient : "Realiza solicitudes"
EscaneoList --> Store : "dispatch(LOAD_ESCANEOS)"
Store --> EscaneoReducer : "Actualiza estado"
```

```
APIClient --> DjangoBackend : "Consume API REST"



@enduml
```