Proyecto de Reconocimiento Facial

INDICE

Primer cuatrimestre 2024 - Universidad Nacional de General Sarmiento - Comisión 02

Nombre del proyecto: InnovaSoft

Nombre del producto: Cypher Vault

Profesores:

- Ing. Francisco Orozco De La Hoz @ forozco@campus.ungs.edu.ar
- Lic. Leandro Dikenstein @ ldikenstein@campus.ungs.edu.ar

Equipo de trabajo:

- Flavio Ybarra - flavio_712@hotmail.com - DNI: 36322712
- Alejandro Moras - spectrelonewolf@gmail.com - DNI: 31625246
- Javier Galeano - javi_b_galeano@hotmail.com - DNI: 41805228
- Melanie Ibarra - meluibarra15@gmail.com - DNI: 44301664
- Ivan Sanchez - ivansncz11@gmail.com - DNI: 42087962
- Fernando Trejo - fernandotrejo125@gmail.com - DNI: 43986607

Introducción

Bienvenidos al proyecto Cypher Vault de Autenticación Facial en Kotlin. Este documento tiene como objetivo dar a conocer el desarrollo de una aplicación capaz de autenticar a los usuarios a través del reconocimiento facial enfocado para Tablets. La misma está diseñada para funcionar sin conexión a internet ofreciendo un servicio de almacenamiento de imágenes seguro y encriptado para el usuario.

Objetivos del Proyecto

1. Desarrollar una aplicación de reconocimiento facial.
2. Implementar la aplicación en Android usando Kotlin y Android Studio.
3. Utilizar OpenCV y TensorFlow Lite para el reconocimiento facial.
4. Almacenar en el dispositivo imágenes cifradas/encriptadas.
5. Facilitar el uso de la aplicación para cualquier tipo de usuario.
6. Optimizar la aplicación para un uso eficiente de los recursos del dispositivo.

Objetivos del Documento

Este documento tiene como objetivo explicar cuáles son los pasos a seguir en el ciclo de vida del desarrollo de este software, es decir, se detallarán los requerimientos funcionales, no funcionales, armado de la WBS (funcionalidades del proyecto), definición de roles, estimaciones de

implementación y diagrama de arquitectura. Más adelante se detalla mejor el objetivo de cada uno.

Definición de Roles

- **Product Owner**: Es el individuo que representa al cliente en el proyecto.
- **Scrum Master**: Supervisa el progreso del proyecto y se asegura de que se cumplan los plazos.
- **Development team**: Encargados de la codificación, el testeo y la implementación de la aplicación.

Equipo de Trabajo y Roles

Nombre	Rol Primario	Rol Secundario
Francisco Orozco De La Hoz	Product Owner	-
Flavio Ybarra	Scrum Master	Tester
Alejandro Moras	Desarrollador	UX/UI
Fernando Trejo	Desarrollador	UX/UI
Javier Galeano	Desarrollador	UX/UI
Ivan Sanchez	Tester	Capacitador y Prueba de Usuario
Melanie Ibarra	Tester	Scrum Master

Metodología

En este proyecto, implementaremos una combinación de metodologías ágiles y Waterfall, también conocida como "Wagile" o "Agilefall". Este enfoque nos permitirá aprovechar lo mejor de ambos métodos para adaptarnos a las necesidades específicas de nuestro equipo de seis personas. A continuación les presentaremos un resumen de cómo lo haremos:

- **Comprender las metodologías:** Todo el equipo debe entender Agile y Waterfall.
- **Identificar las fases del proyecto:** Dividiremos el proyecto en fases claramente definidas.
- **Aplicar Waterfall en las fases iniciales:** Usaremos Waterfall para la planificación, análisis de requerimientos y diseño.
- **Implementar Agile en las fases de desarrollo:** Aplicaremos Agile para las fases de desarrollo y pruebas.
- **Facilitar la comunicación y colaboración:** Fomentaremos la comunicación abierta y la colaboración durante todo el proceso.
- **Realizar retrospectivas periódicas:** Programaremos reuniones regulares de retrospectiva al final de cada fase o sprint.
- **Ser flexible y adaptativo:** Mantendremos una mentalidad flexible y adaptativa a medida que evolucione el proyecto.

Nuestro enfoque Agile se enfocará en Scrum, el cuál se basa en entregar funcionalidades de forma incremental, en períodos de dos semanas. Dentro de las mismas se realizan reuniones diarias del equipo para planificación, control y revisión del trabajo realizado hasta el momento.

Con este enfoque, nuestro equipo podrá gestionar eficazmente el proyecto, adaptarse a los cambios y entregar valor de manera constante y oportuna.

Enlaces

- **Repositorio:** se decidió utilizar Github para que todos los miembros del equipo puedan acceder y trabajar con mayor comodidad. Repositorio Github
- **WBS:** Se decidio utilizar Miro que es una plataforma de colaboración digital para realizar la WBS. Por motivos de seguridad no se compartirá el link pero la misma se mostrará en la documentación.
    - **Herramientas a utilizar:** Android Studio, OpenCV, TensorFlow Lite, Visual Studio.
    - **Comunicación de equipo:** WhatsApp y Discord.
    - **User Stories:** Trello
    - **Comunicación con el Líder del Proyecto:** Telegram o Mail.
    - **Diagrama de arquitectura:** draw.io

Gestión

Misión y Visión del Negocio

Nuestra visión: Aspirar en que sea una aplicación cómoda y fácil de usar dentro de los estandares de seguridad para así brindar tranquilidad y seguridad al usuario.

Nuestra misión: Es crear una aplicación de almacenamiento de imágenes privadas las cuáles son encriptadas en el dispositivo, donde el usuario se registra e ingresa a través del reconocimiento facial mediante la utilizacion de la cámara frontal del dispositivo (tablet).

Posteriormente el ingreso del usuario se realizará comparando la foto tomada con las imágenes guardadas en el dispositivo, estas imágenes estan encriptadas y cifradas.

- **Alcance:**
    - Aplicación para dispositivos Android (Tablets).
    - Registro por reconocimiento facial.
    - Login por reconocimiento fácil.
    - Almacenamiento de imágenes en el dispositivo.
    - Registro Alternativo
- **Fuera del alcance:**
    - Aplicaciones para IOS y Computadoras.

- Varios idiomas.
- Registro biometrico en oscuridad.
- No contemplamos diseño de la aplicacion con interfaz en vertical.
- Multiples formatos de archivos (videos, audios, etc)

Poca información: sobre la implementación de la app no podemos confirmar las funcionalidades que quedan por fuera del alcance.

Plan de Comunicaciones

Para facilitar la comunicación, empleamos la plataforma WhatsApp, que nos brinda un canal de comunicación instantánea y versátil. Esto nos permite interactuar ágilmente entre los miembros del equipo, compartir actualizaciones rápidas y discutir ideas en tiempo real. Además, utilizamos Discord para llevar a cabo reuniones diarias y charlas técnicas. En cuanto a la gestión de tareas y el seguimiento del proyecto, recurrimos a Trello. Esta herramienta nos permitió crear un flujo de trabajo estructurado y asignar tareas, asegurando que cada miembro del equipo estuviera al tanto de sus responsabilidades y plazos. Adicionalmente, mantenemos reuniones presenciales con nuestro product owner para garantizar que nuestro producto final cumpliera con los requisitos del cliente. Además de la posibilidad de mantener contacto a través de Telegram

Requerimientos

En este apartado se detallarán los requerimientos del sistema, además se hará mención de la nomenclatura a utilizar para la clasificación de dichos requerimientos. Los requerimientos funcionales son aquellos que definen la funcionalidades que va a tener el software. Tales requerimientos se clasifican en estos tres tipos:

Requerimientos esenciales: Estos requerimientos hacen que el sistema tenga sentido, es decir, sin esta clases de funcionamientos no se cumplirían el objetivo que necesitan los usuarios.

Requerimientos importantes: Son aquellos que, si no están, el software funciona igual pero se limitará el funcionamiento.

Requerimientos deseables: Son componentes adicionales que pueden ser agregados al software pero su prioridad es la mínima.

Una vez explicado las clasificación de requerimientos funcionales, se hará a continuación mención de los requerimientos no funcionales:

Requerimientos No funcionales: El objetivo de estos requerimientos es explicar las limitaciones o restricciones que el sistema posee. Estos requisitos no tienen ningún impacto en la funcionalidad del software, pero garantizan que el sistema satisfaga las necesidades de los usuarios del sistema.

- Funcionales:
  - Registro:
    - El sistema debe ser capaz de capturar imágenes de la cámara frontal de la Tablet.

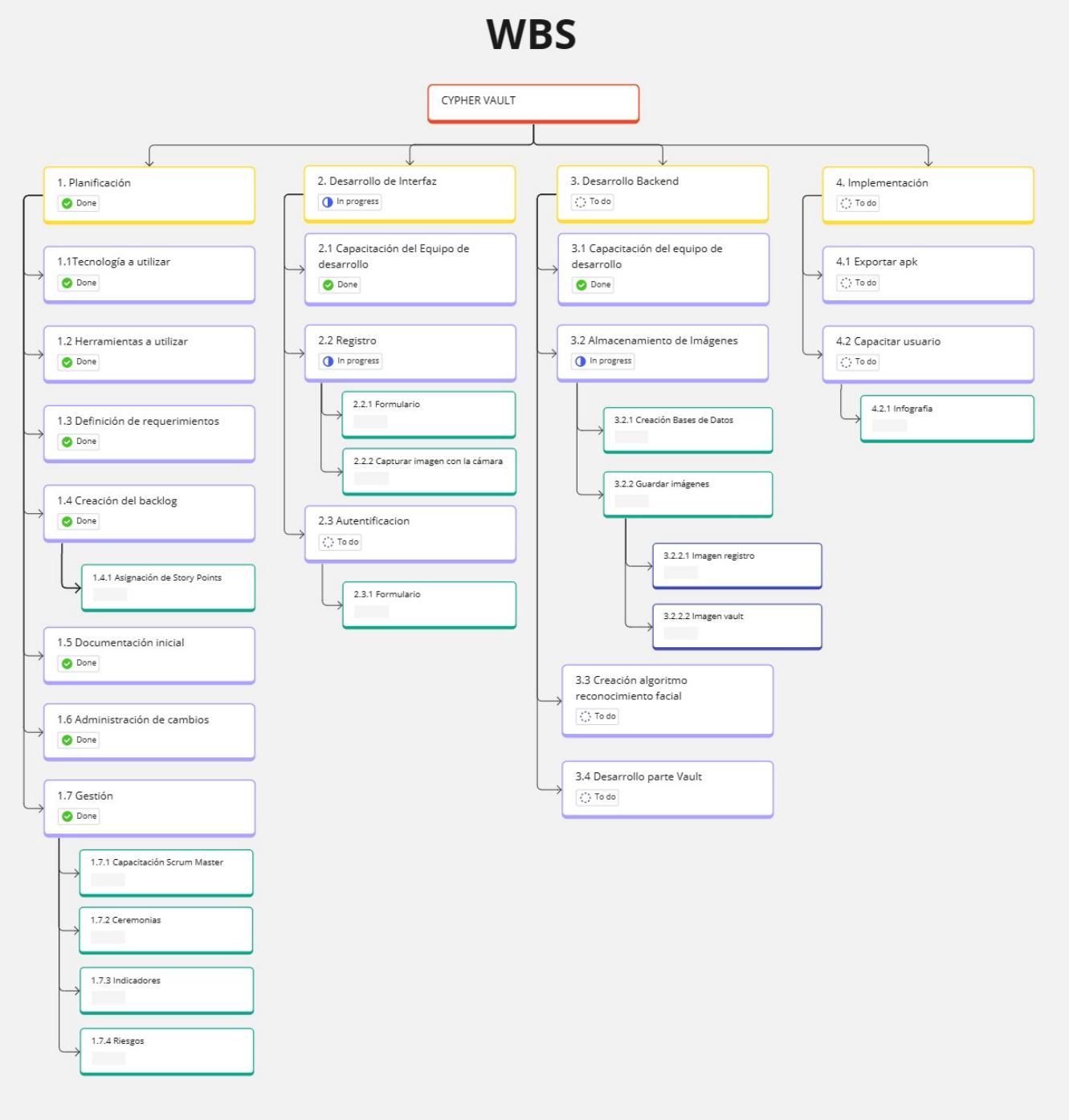
        - Se debe crear una interfaz de login donde el usuario se registre con sus datos (nombre y mail) y su rostro.
        - La interfaz debe tener un boton para capturar la imagen.
        - Tiene que solicitar los permisos necesarios para acceder a la camara
        - Se debera guardar la imagen en una base de datos almacenada en la tablet
        - El sistema debe ser capaz de detectar rostros en las imágenes capturadas.
        - El sistema debe ser capaz de identificar a las personas a partir de sus rostros.
        - Asociar las rostros de las personas a su cuenta de registro
    - Autentificación
        - Una vez registrado el usuario debe ser capaz de loguearse a su cuenta atraves de la verificaion facial
        - El sistema debe ser capaz de autenticar a las personas comparando sus rostros con una base de datos de rostros conocidos almacenada en la Tablet.
        - El sistema debe mostrar un mensaje de "Acceso Permitido" o "Acceso Denegado" en la pantalla de la Tablet en función del resultado de la autenticación.
        - El sistema deberá registrar un log con los datos de ingresos (Hora, ID de persona, etc.)
        - El sistema deberá permitir una alternativa manual de ingreso ante posibles desconexiones (sin Wifi o datos).
    - Perfil de usuario
        - El sistema deberá permitir el ALTA/MODIFICACIONES de las personas a autenticar.
        - El usuario podra modificar su informacion personal o registrar otra foto de su rostro.
    - Galeria
        - Ver imagenes de la tablet en la aplicación.
        - Agregar imagenes de la galeria de la tablet a la galeria de la aplicación.
        - Las imagenes de la aplicación no se ven dentro de la galeria de la tablet
    - Deseables
        - Re-Autentificación de usuario mientras se encuentra en la aplicación.
        - Comprobar que el usuario este frente al dispositivo cada cierto tiempo.
        - Cuando se detecta otro rostro en la captura de la cámara se debe bloquear la aplicación.
        - Capturar imagenes dentro de la aplicación
        - Implementación de la aplicación en vista horizontal
- No Funcionales:
    - Usabilidad:
        - Si el usuario desea entrar y no esta registrado se le debe mostrar un mensaje de "acceso denegado, primero necesitas registrarte"
        - Si el usuario desea ingresar a su cuenta con una foto u otro rostro (no asociado a su cuenta) se le debe mostrar un mensaje de "acceso denegado".

- Si el usuario pudo ingresar a su cuenta se le debe mostrar un mensaje de "acceso permitido".
- Por cada interfaz en la que el usuario se encuentre el sistema debe mostrar el mensaje adecuado correspondiente a la interacción del mismo con la aplicación.

◦ Rendimiento

- El sistema debe ser eficiente en el uso de la batería, la memoria y el procesador de la Tablet.

WBS



Diccionario

Los pesos se clasifican en base a: - 3: Esencial - 2: Importante - 1: Deseable

ID	NOMBRE	DESCRIPCIÓN	TAREA	RESPONSABLE	PESO
1	Planificación	Planificación sobre las tareas que la componen	Planificación, investigación	Todo el equipo	3
1.1	Tecnologías a utilizar	Investigar sobre las tecnologías a utilizar	Investigación	Todo el equipo	3
1.2	Herramientas a utilizar	Investigar sobre las herramientas a utilizar	Investigación	Todo el equipo	3
1.3	Definición de requerimientos	Definir los requerimientos del proyecto	Documentación	Todo el equipo	3
1.4	Creación del backlog	Crear backlog	Planificación	Scrum Master	3
1.4.1	Asignación de Story Points	Estimar esfuerzo de los requerimientos	Planificación, Estimación	Scrum Master	3
1.5	Documentación inicial	Crear la documentación	Planificación	Todo el equipo	3
1.6	Administración de cambio	Planificar el flujo de los cambios	Planificación	Scrum Master	2
1.7	Gestión	Gestión general de indicadores y comunicación del equipo	Planificación, Gestión	Scrum Master	3
1.7.1	Capacitación Scrum Master	Capacitación en herramientas de gestión e indicadores	Capacitación	Scrum master	3

ID	NOMBRE	DESCRIPCIÓN	TAREA	RESPONSABLE	PESO
1.7.2	Ceremonias	Planificación de las ceremonias de Scrum. Sprint planning, dailys, Sprint review, Sprint retrospective	Planificación	Scrum Master	3
1.7.3	Indicadores	Control de inidcadores generales	Planificación, Gestión	Scrum Master	3
1.7.4	Riesgos	Control e identificación de riesgos	Planificación	Scrum master	2
2	Desarrollo de Interfaz	Desarrollar la interfaz	Desarrollo	Equipo de desarrollo	2
2.1	Capacitación del Equipo de Desarrollo	Capacitar al equipo con las tecnologías a utilizar	Capacitación	Equipo de desarrollo	2
2.2	Registro	Crear interfaz registro	Desarrollo	Equipo de desarrollo	2
2.2.1	Formulario	Creación del formulario	Desarrollo	Equipo de desarrollo	2
2.2.2	Cámara	Implementar cámara en interfaz	Desarrollo	Equipo de desarrollo	2
2.3	Autentificación	Implementar interfaz de autentificación	Desarrollo	Equipo de desarrollo	2
2.3.1	Formulario	Creación de formulario de autentificación	Desarrollo	Equipo de desarrollo	2
3	Desarrollo backend	Desarrollar la lógica de la aplicación	Desarrollo	Equipo de desarrollo	3
3.1	Capacitación del equipo de desarrollo	Capacitar al equipo de desarrollo con las tecnologías a utilizar	Capacitación	Equipo de desarrollo	2

ID	NOMBRE	DESCRIPCIÓN	TAREA	RESPONSABLE	PESO
3.2	Almacenamiento de imágenes	Almacenar las imágenes en el dispositivo del usuario	Desarrollo	Equipo de desarrollo	2
3.2.2	Guardar imágenes	Guadar imágenes en la base de datos	Desarrollo	Equipo de desarrollo	3
3.2.2.1	Imagen registro	Guardar imagenes del registro facial	Desarrollo	Equipo de desarrollo	3
3.2.2.2	Imagen vault	Guardar imagenes para la galeria	Desarrollo	Equipo de desarrollo	3
3.2.1	Creación de Base de Datos	Crear base de datos	Desarrollo	Equipo de desarrollo	2
3.3	Crear algoritmo reconocimiento facial	Implementación de lógica de reconocimiento facial	Desarrollo	Equipo de desarrollo	3
3.4	Desarrollo parte Vault	Implementar aplicación de galería privada	Desarrollo	Equipo de desarrollo	3
4	Implementación	Puesta en servicio de la aplicación	Implementación	Capacitador y Prueba de Usuario, Equipo de desarrollo	2
4.1	Exportar apk	Compilación del proyecto a formato de dispositivo android	Implementación	Equipo de desarrollo	2
4.2	Capacitar usuario	Capacitar a usuario final	Capacitación	Capacitador y Prueba de Usuario	2
4.2.1	Infografía	Mostrar imagen de uso	Capacitación	Capacitador y Prueba de Usuario	2

Calendario

Entrega	Fecha	Tareas
1	(19/4)	Presentación de Plan de Proyecto
2	(26/4)	Implementacion de interfaz inicial
3	(8/5)	Implementación de algoritmo de reconocimiento facial
4	(22/5)	A definir
5	(5/6)	A definir
6	(14/6)	A definir
7	(26/6)	A definir

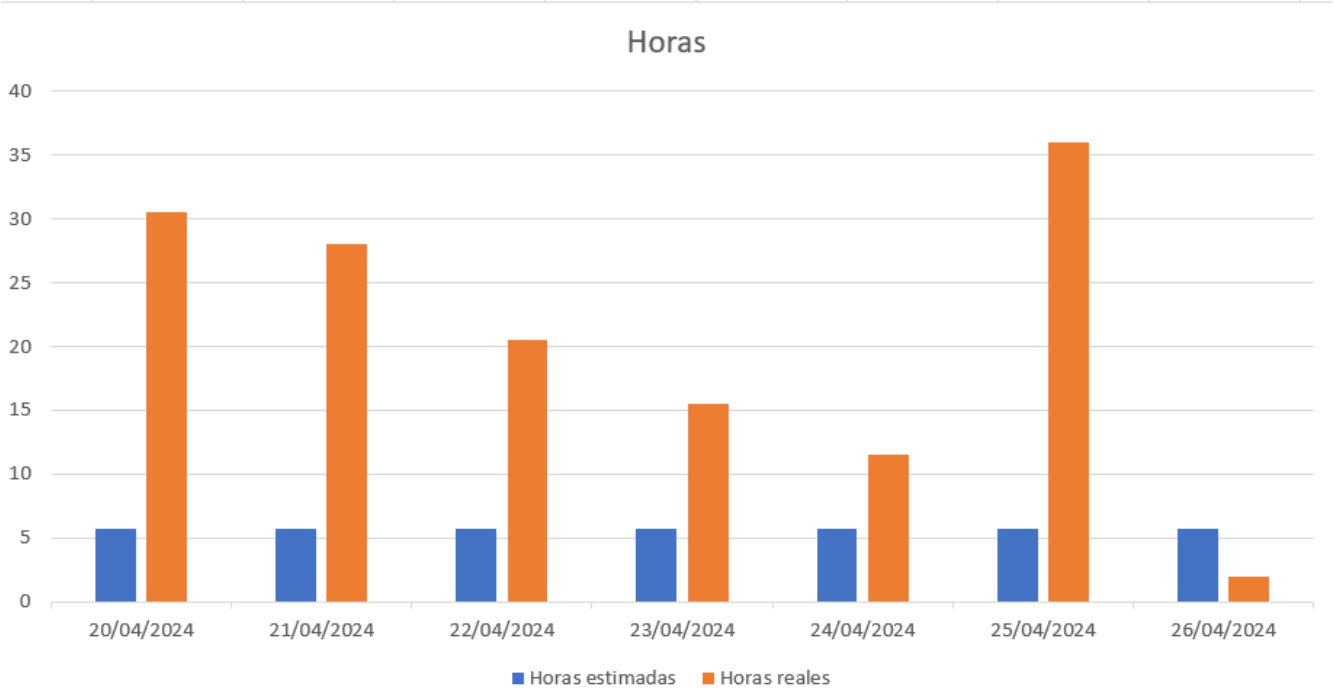
Estimaciones Iniciales

Se entregará un prototipo de la interfaz funcional para el registro, la autentificación y base de datos. Estimamos que el tiempo empleado será:

- **Capacitación del equipo en las tecnologías:** 5hs por cada desarrollador y tester.
- **Desarrollo:** 20hs por cada desarrollador.
- **Testing:** 10hs por tester.

Horas empleadas

- **Capacitación del equipo en las tecnologías:** 10hs por cada desarrollador y tester.
- **Desarrollo:** 25hs por cada desarrollador.
- **Testing:** 2hs por tester.
- **Ceremonias y reuniones técnicas:** 15hs con todo el equipo presente



Riesgos

- R1 Falta de claridad en los objetivos
- R2 Escasez de práctica en la gestión de proyectos
- R3 Constantes modificaciones en los requerimientos
- R4 Ausencia de un miembro del equipo
- R5 Tensiones comunicativas dentro del equipo
- R6 Estimación erroneas debido a la falta de experiencia
- R7 La curva de aprendizaje en nuevas tecnologias podria afectar la eficiencia de los desarrolladores
- R8 Variación en los tiempos de dedicación entre los miembros del equipo

Riesgo	Probabilidad de ocurrencia	Severidad/ Impacto	Exposición al riesgo
R1	2	3	6
R2	3	2	6
R3	2	3	6
R4	2	3	6
R5	2	3	6
R6	2	2	4
R7	3	1	3
R8	3	1	3

- Plan de mitigacion
  - R1 = Armado detallado de la WBS. Organizar reuniones para revisar y aclarar los objetivos.
  - R2 = Documentar las lecciones aprendidas durante el proyecto. Contar con miembros experimentados que brinde asistencia y orientacion al equipo

    - R3 = Investigar y comunicarse con las autoridades reguladoras pertinentes
    - R4 = Designar roles suplentes para asegurar la continuidad del trabajo en caso de ausencia de algun miembro
    - R5 = Programar reuniones periodicas y practicar la escucha activa durante las interacciones
    - R6 = Realizar estimaciones realistas teniendo en cuenta la experiencia del equipo y los recursos disponibles, utilizando enfoques de metodologias apropiadas
    - R7 = Investigar , evaluar y capacitarse en nuevas tecnologias antes de su implementacion en el proyecto
    - R8 = Elaborar un calendario que refleje los horarios disponibles de cada miembro del equipo
- Plan de contingencia
    - R1 = Definir y compartir los objetivos del proyecto de manera clara en todo el equipo
    - R2 = Ampliar conocimientos tanto mediante la teoria como consultando a profesores
    - R3 = Adaptarse a los nuevos cambios que surjan durante el proyecto.
    - R4 = Brindar apoyo a los compañeros que enfrenten dificultades personales y, de ser necesario, redistribuir tareas
    - R5 = Asignar un mediador para resolver los conflictos internos de manera efectiva
    - R6 = Establecer un margen de contingencia para hacer frente a situaciones imprevistas. Aprender de tareas realizadas previamente.
    - R7 = Falicitar la tranferencia de conocimiento mediante la colaboracion de un miembro mas experimentado
    - R8 = Adaptar las tareas según el ritmo de trabajo y conocimiento de cada miembro del equipo

Entregables

Definimos los hitos que ocurrirán en las diferentes fechas del proyecto. El primer hito se enfocará en la presentación formal del proyecto al cliente. En esta se explicará el plan de gestión que tendremos para administrar el proyecto. En los hitos restantes se presentará al cliente los avances en el producto.

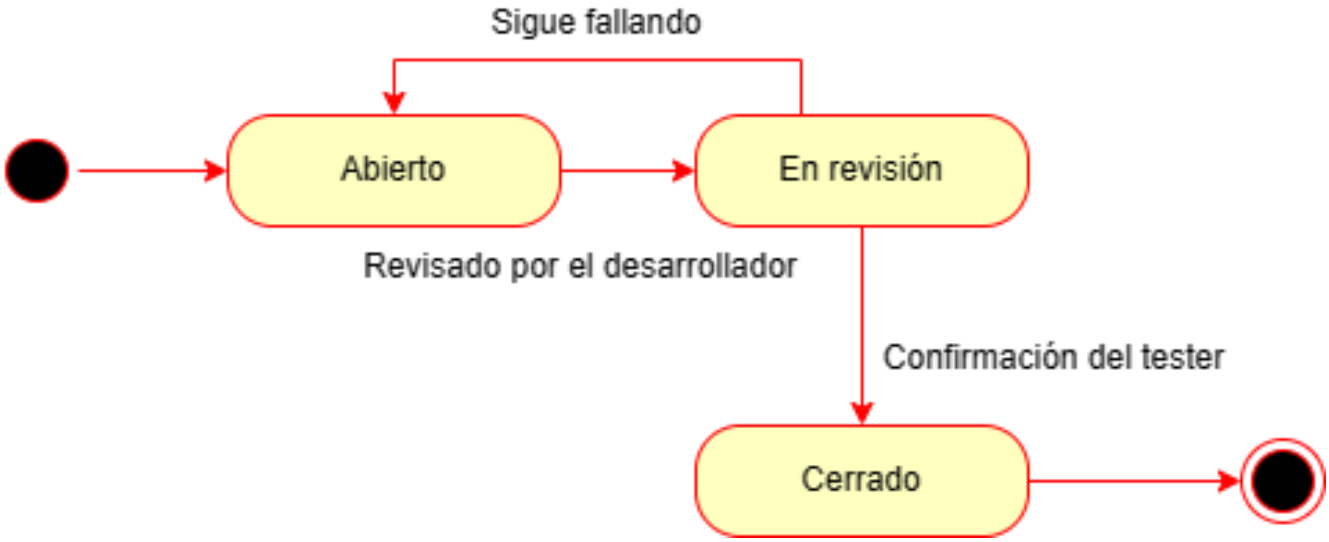
- Presentación del proyecto el día 19/04
- Reunión formal 1 el día 26/04
- Reunión formal 2 el día 8/05
- Reunión formal 3 el día 22/05
- Reunión formal 4 el día 5/06
- Reunión formal 5 el día 14/06
- Presentación final el día 26/06
    - Entregables para el proximo sprint del dia 8/05:

- Resolución de bugs del sprint anterior
- Investigación de reconocimiento facial
- Mejoras en interfaz
- Implementación y desarrollo de algoritmo de reconocimiento facial (deseable)

Administración en el Manejo de Bugs

Ejecutar una gestión eficaz de errores y pruebas es un componente esencial en un sistema de software. Estas tareas son vitales para asegurar que el sistema opere de forma fiable, eficiente y satisfaga las necesidades de los usuarios.

Haremos un seguimiento de los errores en una planilla de excel en un drive compartido con todo el equipo en el cual se detalla fecha de descubrimiento, funcionalidad afectada, tester que lo identifico, desarrollador responsable, detalle del bug, estado y fecha de cierre. Los categorizaremos en tres niveles de acuerdo a su severidad: bajo, medio o alto. Esto nos permitirá determinar cuáles son las dificultades más urgentes y cuáles son de menor prioridad.



Administración de Cambios

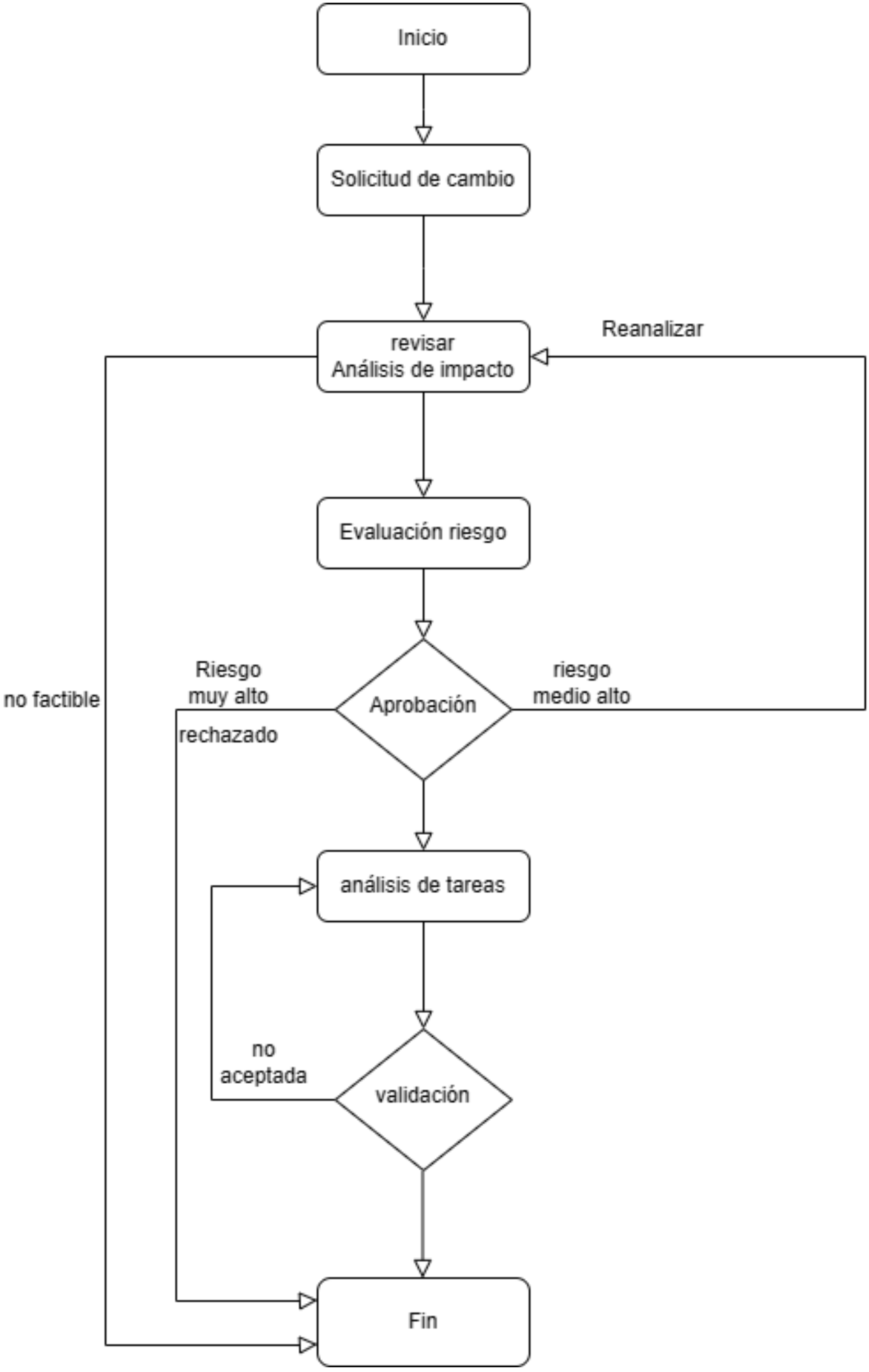
La adecuada administración de cambios es crucial para el éxito de cualquier proyecto. Los cambios pueden surgir por distintos motivos, como nuevos requerimientos del cliente, descubrimientos durante el desarrollo, o variaciones en las condiciones del mercado.

Para documentar los cambios se emplearán:

- Informe de avance: se mantendrá actualizado un informe de avance que puede incluir un registro de todos los cambios. Además, se registrarán los cambios en las minutas de las reuniones.
- Trello: como se mencionó previamente, será nuestra principal plataforma para el seguimiento y gestión de cambios.

Aprobación o rechazo de cambios: Un comité de cambios evaluará cada solicitud de cambio basándose en los siguientes factores:

- Cronograma: se considerará si el cambio afecta al cronograma del proyecto. Aquellos cambios que impacten serán evaluados en función de su urgencia y prioridad.
- Alcance: se analizará si el cambio está en línea con los objetivos y el alcance del proyecto.

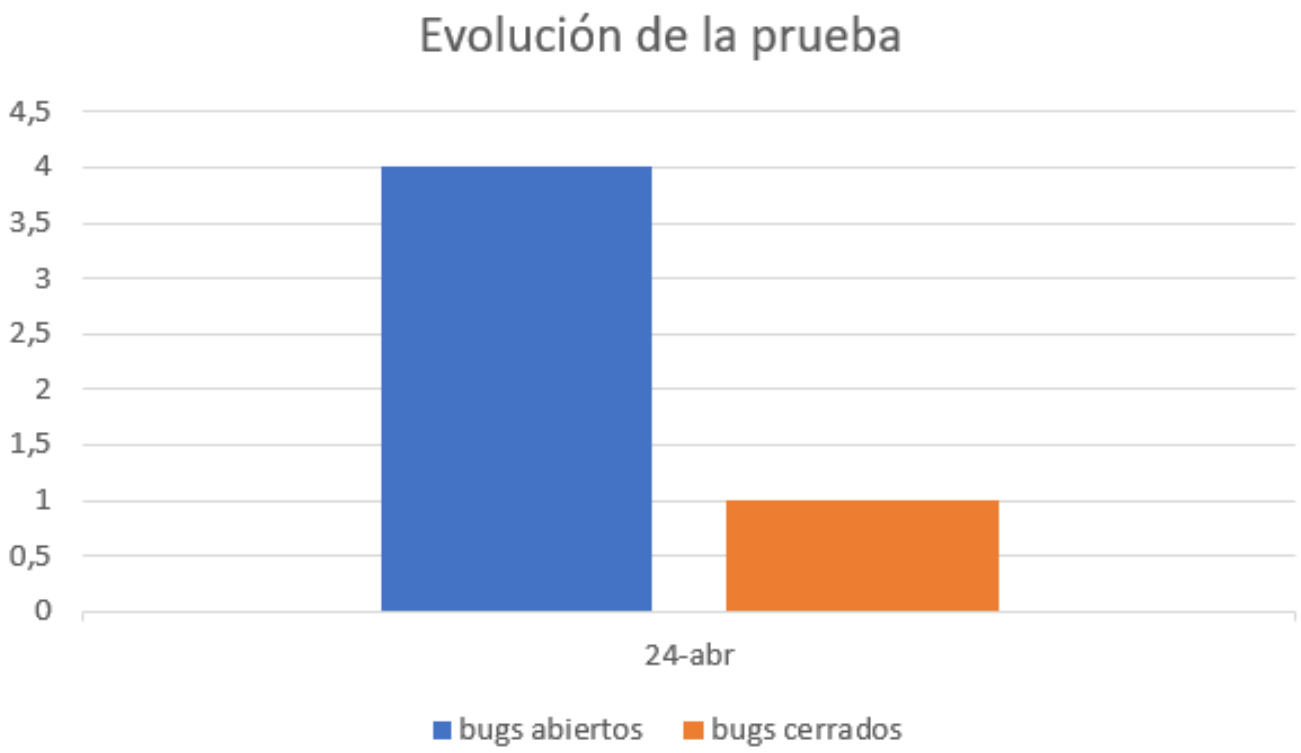

Inicio
Solicitud de cambio
revisar
Análisis de impacto
Reanalizar
Evaluación riesgo
Riesgo
muy alto
Aprobación
riesgo
medio alto
no factible
rechazado
análisis de tareas
no
aceptada
validación
Fin

Indicadores

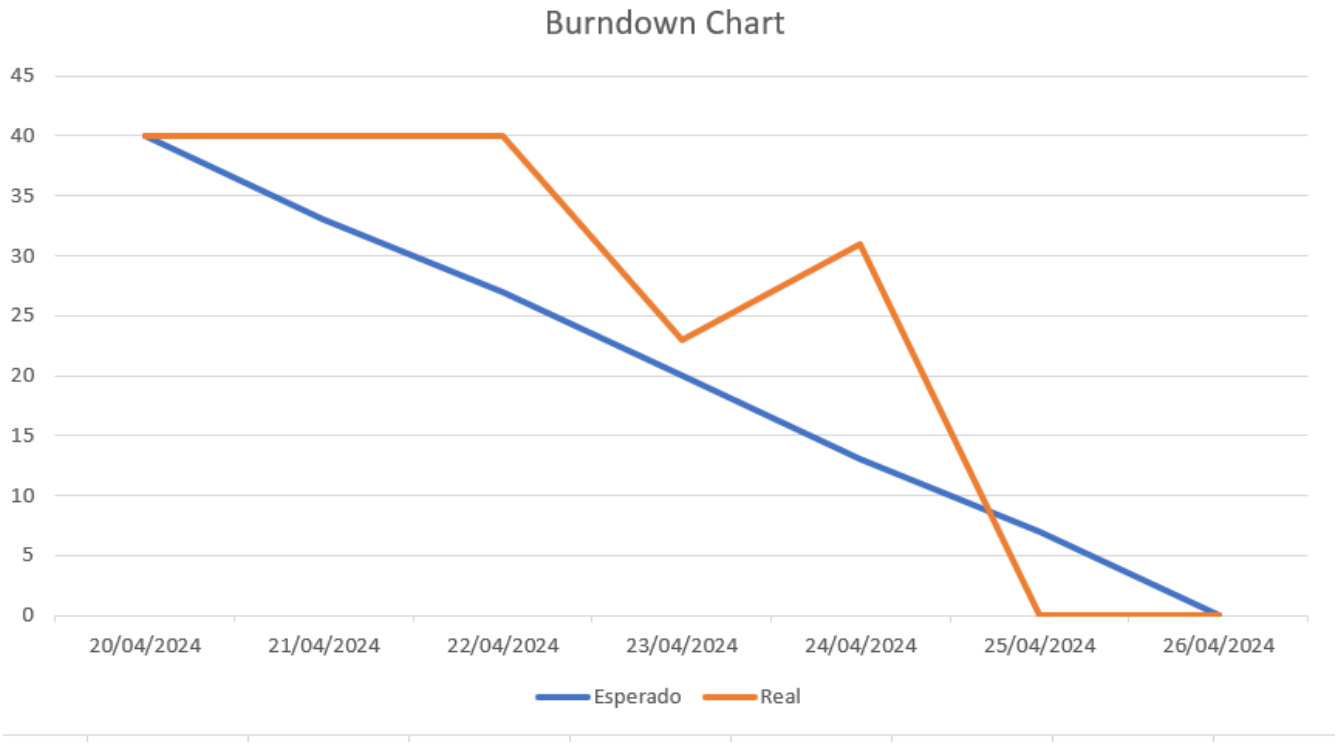
- Funcionalidad Completa

Funcionalidad	Peso	Fecha de creacion	Fecha estimada	Fecha real	Estado
Formulario Logueo	5	20-abr	25-abr	25-abr	Realizada, testeo proximo sprint
Formulario Registro	5	20-abr	25-abr	25-abr	Finalizada
Ver camara frontal en aplicación	5	20-abr	25-abr	25-abr	Finalizada
Creacion base de datos	3	20-abr	25-abr	25-abr	Realizada, testeo proximo sprint
Almacenar registro	8	23-abr	25-abr	25-abr	Realizada, testeo proximo sprint

- Nivel de Calidad
- Evolución de la Prueba



- Burndown Chart



Tecnologías

- **Android Studio:** Es un entorno de desarrollo integrado gratuito diseñado específicamente para el desarrollo de aplicaciones Android.
- **Kotlin:** Kotlin Es un lenguaje de programación de código abierto para aplicaciones Android.
- **OpenCV:** Es una biblioteca que proporciona una amplia gama de funciones y algoritmos para el procesamiento de imágenes y vídeo
- **TensorFlow Lite** TensorFlow Lite es un marco de trabajo ligero desarrollado por Google que permite ejecutar modelos de aprendizaje automático en dispositivos móviles e integrados.
- **SQLite:** SQLite es un sistema de gestión de bases de datos relacional (RDBMS) ligero, rápido, autónomo y de código abierto.

Continua: por motivos de estetica y facil acceso se implementa dentro del indice Herramientas / Implementacion.

Herramientas / Implementacion

Visual Studio Code

Fuente: Microsoft.com/VisualStudio

Dentro del proyecto su uso no es primario, pero si se utiliza para revisar las clases dentro del proyecto de Android Studio, por otro lado es una gran herramienta a la hora de confeccionar el informe / documentacion del proyecto.

Editor de código fuente independiente que se ejecuta en Windows, macOS y Linux. El IDE de Visual Studio es una plataforma de lanzamiento creativa que puede utilizar para editar, depurar y compilar código y, finalmente, publicar una aplicación. Además del editor y depurador estándar que ofrecen la mayoría de IDE, Visual Studio incluye compiladores, herramientas de completado de código, diseñadores gráficos y muchas más funciones para mejorar el proceso de desarrollo de software.

Android Studio

Fuente: android.com/developer

Es la herramienta principal del desarrollo de la aplicacion, el mismo es un IDE robusto el cual cuenta con varias funcionalidades las cuales acompañan las etapas de desarrollo, depuracion, testeo e implementacion.

Entorno de desarrollo integrado (IDE) oficial del desarrollo de apps para Android. Basado en el potente editor de código y las herramientas para desarrolladores de IntelliJ IDEA, Android Studio ofrece aún más funciones que mejoran tu productividad cuando compilas apps para Android, como las siguientes:

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Ediciones en vivo para actualizar elementos componibles en emuladores y dispositivos físicos, en tiempo real
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra Variedad de marcos de trabajo y herramientas de prueba
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine.

Aclaracion: Dentro del equipo de desarrollo y testeo utilizamos los mismos dispositivos celulares para emular y testear la implementacion/desarrollo, ya que en algunos casos es imposible por las capacidades computacionales de los equipos

(computadoras) de cada uno de los integrantes.

IMPLEMENTACION

En esta seccion se pasa a detallar cada una de las partes del desarrollo, junto a sus herramientas, ya que dentro de android studio como se detallo anteriormente se encuentran funcionalidades especificas.

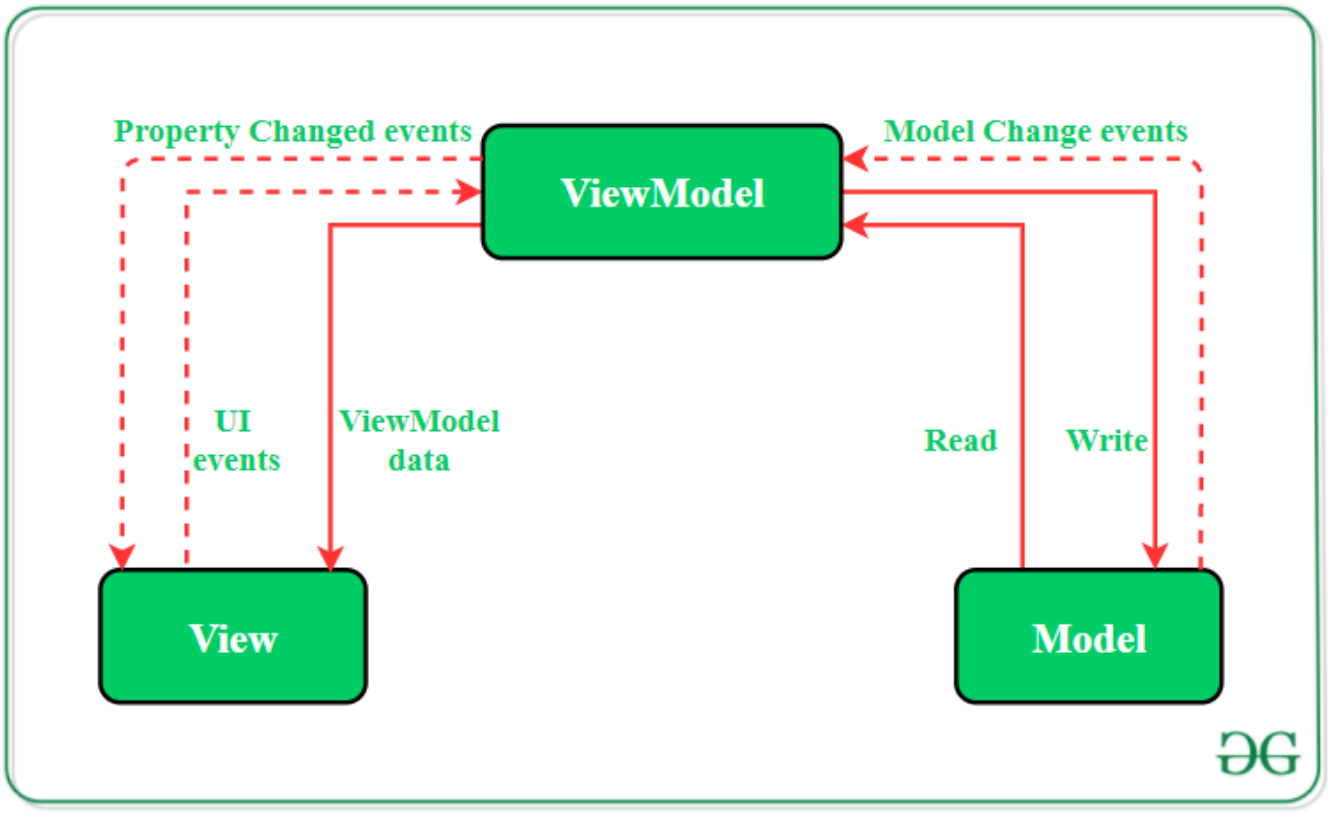
Para comenzar nos encontramos con el ultimo IDE estable lanzado por Android, el cual es la version Iguana, luego se creo un proyecto con la version minima compatible recomendada por el mismo. Teniendo esto en cuenta nuestra aplicacion es compatible desde Android 7 hasta la ultima version lanzada al dia de hoy.

¿Por que no versiones anteriores?: Porque esto genera problemas en las dependencias (funcionalidades ofrecidas por android) disponibles, lo cual provoca que la aplicacion pierda posibilidades de escalado, ya que hay versiones anteriores que resultan incompatibles con las mismas dependencias..

Modelo de diseño

Fuente: barcelonageeks.com/mvvm

- Al organizar los códigos de acuerdo con un patrón de diseño, ayuda en el mantenimiento del software. Al tener conocimiento de todas las partes lógicas cruciales de la aplicación de Android , es más fácil agregar y eliminar funciones de la aplicación. Además, los patrones de diseño también aseguran que todos los códigos se cubran en las pruebas unitarias sin la interferencia de otras clases. Model — View — ViewModel (MVVM) es el patrón de arquitectura de software reconocido en la industria que supera todos los inconvenientes de los patrones de diseño MVP y MVC . MVVM sugiere separar la lógica de presentación de datos (vistas o interfaz de usuario) de la parte lógica empresarial central de la aplicación.
- Las capas de código separadas de MVVM son:
  - **Modelo:** esta capa es responsable de la abstracción de las fuentes de datos. Model y ViewModel trabajan juntos para obtener y guardar los datos.
  - **Vista:** El propósito de esta capa es informar al ViewModel sobre la acción del usuario. Esta capa observa el ViewModel y no contiene ningún tipo de lógica de aplicación.
  - **ViewModel:** Expone esos flujos de datos que son relevantes para la Vista. Además, sirve como enlace entre el Modelo y la Vista.



- El patrón MVVM tiene algunas similitudes con el patrón de diseño MVP (Modelo, Vista, Presentador) ya que ViewModel desempeña el rol de Presentador. Sin embargo, los inconvenientes del patrón MVP han sido resueltos por MVVM de las siguientes maneras:
  - ViewModel no contiene ningún tipo de referencia a la Vista.
  - Existe una relación de muchos a 1 entre View y ViewModel.
  - No hay métodos de activación para actualizar la Vista.

INTERFAZ

Fuente: android.com/compose

Fuente: android.com/nav_controller

Fuente: android.com/composable

Fuente: android.com/patterns

El código se divide en tres paquetes que se encuentran en `app > src > main > java > com.example.cypher_vault`. Acá hay dos paquetes: uno llamado 'controller' y otro llamado 'view'.

Paquete Controller > Authentication

- AuthenticationController.kt
  - `AuthenticationController` es una clase que toma como parámetro un `NavController` y devuelve el `NavController` con la dirección a la que debe navegar. Cada dirección tiene su propia función: `fun navigateToCamera()`, `fun navigateToConfirmation()`, `fun navigateToLogin()` (esta última falta implementar).

```
fun registerUser( ①
        email: String,
        name: String,
        showDialog: MutableState<Boolean>,
        errorMessage: MutableState<String>
    )
```

① La función `registerUser` valida los campos por el momento. Más adelante deberá enviarlos al modelo para guardarlos en la base de datos. Recibe como parámetros `email`, `name`, `showDialog` y `errorMessage`. Estos parámetros son para que salga la alerta y mostrarla con sus respectivos mensajes. Si todos los campos están bien, llama a `navigateToCamera` y los manda a la cámara.

- Las funciones `validateMail()`, `validateName()` y `validateFields` verifican la validez de los campos de entrada.
  - `validateMail(email: String)`: Se fija que se cumpla android.util.Patterns.EMAIL_ADDRESS.matcher(email).
  - `validateName(name: String)`: Se fija que no tenga menos de 3 carácteres el nombre.
  - `validateFields(name: String, email: String)`: Se fija que no esten vacios.

Paquete View > Registration

NavigationHost.kt

```
fun NavigationHost() ①
```

① `NavigationHost()` es una función que se utiliza para manejar la navegación en la aplicación, cada vez que se presiona un botón cambia las pantallas.

Aclaracion: Empieza en register por predeterminado y luego va cambiando, toma como parámetro las direcciones que le pasa el AuthenticationController, .

- **Definición de pantallas**: Dentro de esta función `NavHost`, se definen varias pantallas que representan diferentes partes:
  - **register**: Esta es la pantalla inicial donde los usuarios pueden registrarse. Muestra `InitialScreen`.
  - **camera** Esta es la pantalla donde los usuarios pueden usar la cámara durante el proceso de registro. Muestra `RegistrationCameraScreen`.
  - **confirmation**: Esta es la pantalla donde los usuarios pueden confirmar su registro. Muestra `ConfirmationScreen`.
  - **login**: Esta es la pantalla donde los usuarios pueden iniciar sesión. Falta implementar.

InitialScreen.kt

```
fun RegistrationCameraScreen(authenticationController: AuthenticationController) ①
```

① Recibe como parametro authenticationController para luego poder navegar por la aplicacion

`InitialScreen` es la pantalla inicial donde los usuarios se van a registrar. Se encuentran los campos de entrada para el correo electrónico y el nombre. Al hacer clic en el botón "Registrarse", se llama al método `registerUser` del `AuthenticationController`.

RegistrationCameraScreen.kt

```
fun RegistrationCameraScreen(authenticationController: AuthenticationController) ①
```

① Recibe como parametro authenticationController para luego poder navegar por la aplicacion

Esta función Muestra la vista previa de la cámara `ProcessCameraProvider:` Esta es una clase que se utiliza para interactuar con las cámaras disponibles en el dispositivo. En este caso, se obtiene una instancia de ProcessCameraProvider y se recuerda para su uso posterior.

`CameraSelector:` Esta es una clase que se utiliza para seleccionar una cámara en el dispositivo. En este caso, se está seleccionando la cámara frontal.

```
fun CloseCameraButton(isCameraOpen: MutableState<Boolean>,
                      cameraProvider: ProcessCameraProvider,
                        authenticationController: AuthenticationController) ①
```

① Botón que se muestra para cerrar la cámara e ir a la parte de ConfirmationScreen

```
fun CameraPreview(preview: Preview) ①
```

① Muestra la vista previa de la cámara en la interfaz de usuario. Utiliza la clase AndroidView para mostrar la vista previa de la cámara en la interfaz de usuario de Compose.

ConfirmationScreen.kt

```
fun ConfirmationScreen(authenticationController: AuthenticationController) ①
```

① Recibe como parametro authenticationController para luego poder navegar por la aplicacion

`ConfirmationScreen` Es una pantalla que muestra un mensaje de que se pudo registrar y un botón para iniciar sesión

Paquete View > Login

LoginList.kt

```
fun NavigationLogin(authenticationController: AuthenticationController) ①
```

① `NavigationLogin()`: Esta función se encarga de mostrar un lista de los usuarios que ya están registrados en la aplicación. Permite a los usuarios navegar a través de sus cuentas de forma eficiente.

```
fun loginCamera(authenticationController: AuthenticationController, user: String) ①
```

① La función `loginCamera` se activa después de que el usuario ha seleccionado su cuenta. Su propósito es encender la cámara frontal para realizar una verificación biométrica,asegurándose de que la cuenta seleccionada pertenezca realmente al usuario en cuestión. Esta validación permite mantener la seguridad y la integridad de la cuenta.

```
fun CloseCameraButton(cameraProvider: ProcessCameraProvider, authenticationController:
AuthenticationController) ①
```

① La función `CloseCameraButton` permite al usuario cerrar la cámara frontal si se ha seleccionado una cuenta incorrecta. Ofrece una interfaz para regresar de manera rápida al inicio de sesión, específicamente a la pantalla de `NavigationLogin`, facilitando el desplazamiento dentro de la aplicación.

ALMACENAMIENTO

Guardar datos en una base de datos es ideal para los datos estructurados o que se repiten, como la información de contacto. En esta página, en la que se asume que estás familiarizado con las bases de datos SQL en general, encontrarás información que te ayudará a comenzar a usar bases de datos SQLite en Android. Las APIs que necesitarás para utilizar una base de datos en Android están disponibles en el paquete android.database.sqlite.

Consideramos utilizar: la libreria Room de Android Studio.

Fuente : android.com/room

Fuente : android.com/room/definir_datos

Fuente : android.com/room/accesando_datos

Fuente : android.com/room/interface

Fuente : medium.com/tutorial_room

¿Por que Room?: Si bien estas APIs son potentes, se caracterizan por ser bastante específicas y su uso requiere de mucho tiempo y esfuerzo. No hay verificación en tiempo de compilación de las consultas de SQL sin procesar. A medida que cambia tu grafo de datos, debes actualizar manualmente las consultas de SQL afectadas.

> Este proceso puede llevar mucho tiempo y causar errores. Debes usar mucho código estándar para convertir entre consultas de SQL y objetos de datos. Por estos motivos, usamos la Biblioteca de persistencias Room como una capa de abstracción para acceder a la información de las bases de datos SQLite la app.

Componentes principales

- Estos son los tres componentes principales de Room:
  - La clase de la base de datos que contiene la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes de la app.
  - Las entidades de datos que representan tablas de la base de datos de tu app.
  - Los objetos de acceso a datos (DAOs) que proporcionan métodos que tu app puede usar para consultar, actualizar, insertar y borrar datos en la base de datos.

Implementacion dentro de Android Studio

Dentro de Android Studio es necesario la implementacion de de dependencias, especificamente dentro del archivo `build.gradle`. A continuacion los agregados dentro la misma.

Gradle module app

```
plugins {
    kotlin("kapt") ①
}
dependencies {
  implementation("androidx.room:room-runtime:2.6.1") ②
  annotationProcessor("androidx.room:room-compiler:2.6.1") ③
    kapt("androidx.room:room-compiler:2.6.1") ④
}
```

① Libreria encargada de las anotaciones dentro de kotlin, se implementa para la correcta interpretacion de la anotacion 4.

② Declaracion de la dependencia Room

③ Declaracion de las anotacionesde de Room.

④ Agregado de las anotaciones dentro de Room.

Paquete database

- El mismo consta de siete archivos, como se nombro anteriormente, la base de datos en Room consta de 3 partes principales, la clase de la base de datos, las entidades de datos (tablas) y los objetos de acceso a datos (DAO) (Interfaces en las cuales estan descriptas las querys).
- Los archivos son:
  - AppDatabase.kt
  - User.kt
  - UserDao.kt

- Images.kt
- ImagesDao.kt
- ImagesRegister.kt
- ImagesRegisterDao.kt

AppDatabase.kt

```
@Database(entities = [User::class, Images::class, ImagesRegister::class], version = 2)
①
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
    abstract fun imageDao(): ImageDao
    abstract fun imageRegisterDao(): ImageRegisterDao
}
```

① Creacion/Definicion de una base de datos con tres tablas (Usuarios, imagenes y registro de imagenes).

User.kt

```
@Entity
data class User(
    @PrimaryKey val uid: Long,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "entry_date") val entryDate: Long, // Fecha de ingreso
    @ColumnInfo(name = "pin") val pin: String? // PIN del usuario
) ①
```

① Definicion de la entidad User

UserDao.kt

```
@Dao
interface UserDao { ①
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "email LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insert(user: User) // Método para insertar un solo usuario

    @Query("SELECT * FROM user WHERE email = :email LIMIT 1")
```

```
    fun findByEmail(email: String): User? // Método para buscar un usuario por su
correo electrónico

    @Delete
    fun delete(user: User)

    @Insert
    fun insertAll(vararg users: User)

    @Query("SELECT * FROM user WHERE uid = :userId")
    fun getUserById(userId: Int): User?
}
```

① Interfaz de la entidad User

Images.kt

```
@Entity(
    tableName = "images",
    foreignKeys = [ForeignKey(
        entity = User::class,
        parentColumns = ["uid"],
        childColumns = ["user_id"],
        onDelete = ForeignKey.CASCADE
    )]
)
data class Images(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val imageData: ByteArray,
    val user_id: Int
) ①
```

① Definicion de la entidad Images, la misma es para el almacenamiento de las imagenes privadas (galeria principal de la aplicacion).

ImageDao.kt

```
@Dao
interface ImageDao { ①
    @Insert
    fun insertImage(images: Images)

    @Query("SELECT * FROM images WHERE user_id = :userId")
    fun getImagesForUser(userId: Int): List<Images>

    // Otros métodos según sea necesario
}
```

① Interfaz de la entidad Images

ImagesRegister.kt

```
@Entity(
    tableName = "images_register",
    foreignKeys = [ForeignKey(
        entity = User::class,
        parentColumns = ["uid"],
        childColumns = ["user_id"],
        onDelete = ForeignKey.CASCADE
    )]
)
data class ImagesRegister(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val imageData: ByteArray,
    val user_id: Int // referencia al usuario que posee la imagen
) ①
```

① Definicion de la entidad ImagesRegister, aqui se almacenaran las imagenes de registro del usuario.

ImagesRegisterDao.kt

```
@Dao
interface ImageRegisterDao { ①
    @Insert
    fun insertImage(imagesRegister: ImagesRegister)

    @Query("SELECT * FROM images WHERE user_id = :userId")
    fun getImagesForUser(userId: Long): List<ImagesRegister>
}
```

① Interfaz de la entidad ImagesRegister.

Paquete model > dbmanager

- Se define la interfaz DataBaseManager la cual contiene las tres interfases principales UserDao, ImagesDao e ImagesRegisterDao, la misma se implementa para aislar y organizar los llamados aparte de facilitar la inicializacion de la base de datos.

DataBaseManager.kt

```
object DatabaseManager {
    private lateinit var database: AppDatabase

    fun initialize(context: Context) {
        database = Room.databaseBuilder(
            context.applicationContext,
            AppDatabase::class.java, "my_database"
        ).build()
    }
```

```
    // Métodos relacionados con la tabla de usuarios
    fun getAllUsers(): List<User> {
        return database.userDao().getAll()
    }

    fun getUserById(userId: Int): User? {
        return database.userDao().getUserById(userId)
    }

    fun insertUser(user: User) {
        database.userDao().insert(user)
    }

    fun deleteUser(user: User) {
        database.userDao().delete(user)
    }

    // Métodos relacionados con la tabla de imágenes
    fun insertImage(image: Images) {
        database.imageDao().insertImage(image)
    }

    fun getImagesForUser(userId: Int): List<Images> {
        return database.imageDao().getImagesForUser(userId)
    }

    // Métodos relacionados con la tabla de registros de imágenes
    fun insertImageRegister(imageRegister: ImagesRegister) {
        database.imageRegisterDao().insertImage(imageRegister)
    }

    fun getImageRegistersForImage(user_id: Long): List<ImagesRegister> {
        return database.imageRegisterDao().getImagesForUser(user_id)
    }

    // Otros métodos según sea necesario para otras operaciones con usuarios, imágenes
e imágenes registros
}
```