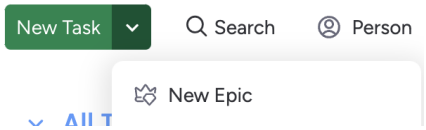
# Capstone Project

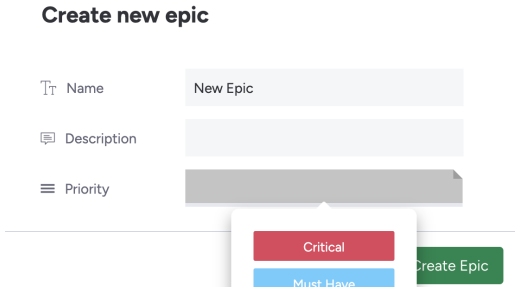
To complete this template, please refer to the "Capstone Project Guidelines." Fill in only the necessary sections according to the considerations outlined in the Capstone document.

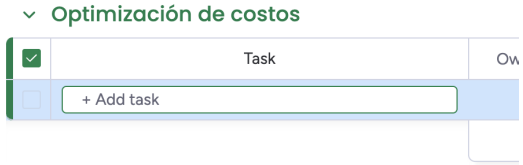
| |
|---|
| Creator's name: Grethel Bello Cagnant |
| Discipline: Security<br>Level: General |
| **Propósito**<br>Este documento describe la arquitectura de software del sistema operacional para la empresa ACME cuyo negocio es el envío de paquetería a nivel internacional. |
| **Audiencia**<br>Este documento está dirigido a las partes interesadas en el proyecto, así como a los equipos de desarrollo que están implementando el sistema. |
| **Descripción del Proyecto Final**<br>Este proyecto final se centra en mostrar y conocer los recursos y herramientas utilizadas por líderes de equipos de desarrollo de software con enfoque específico en el desarrollo de software seguro. Utilizando herramientas de organización y colaborativas con la finalidad de ampliar la capacidad de priorizar y asignar tareas específicas a los equipos de desarrollo. Herramienta de uso aplicable : JIRA/Kanban Board (monday) |
| **Actividad**<br>De acuerdo a la documentación proporcionada se harán las siguientes actividades: |

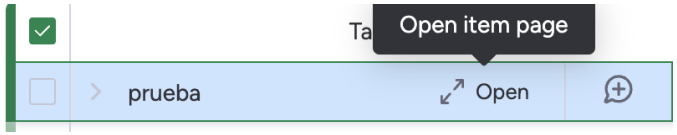
- Analizar la arquitectura propuesta de acuerdo a los pilares del Well-Architected Framework
  - Excelencia Operativa
    - ¿Cómo garantizamos que cada versión cumpla con la calidad requerida?
    - ¿Qué tan tardado es hacer un release?
    - ¿Cómo monitoreamos el estado del sistema?
    - ¿Cuál es la estrategía para dar seguimiento a los errores reportados?
    - ¿Qué tan fácil es hacer cambios de infraestructura?
  - Seguridad
    - ¿Qué mecanismos de autenticación están implementados?
    - ¿Cómo podemos trazar el flujo de la información y sus accesos?
    - ¿Qué seguridad está implementada en front end, back end y comunicación con terceros?
    - ¿Qué estándares de protección de datos debemos seguir?
    - ¿Cuáles son los riesgos identificados?
  - Fiabilidad
    - Si perdiéramos las bases de datos, ¿qué plan de recuperación tenemos?
    - ¿Cuál es la cadena de comunicación y medios en caso de errores?
    - ¿Cuáles son las expectativas de rendimiento de los sistemas?
    - ¿Cómo monitoreamos el rendimiento del sistema?
  - Eficacia del rendimiento
    - ¿Cuáles son los recursos que requiere el sistema?
    - ¿Qué disponibilidad podemos garantizar para el sistema?
    - ¿Hay partes del sistema que podrían representar menos riesgo si se consumieran como servicio?
  - Optimización de costos
    - ¿Cuál es el costo operacional del sistema?
    - ¿Cómo se monitorean los costos mensualmente?
    - ¿Hay estrategias para alertar en el incremento o decremento de los costos?
- Diagrama tu solución para complementar tu análisis
- Mostrar el costo de implementar o no implementar tu solución, así como las etapas en las que puede ser dividido
- Integrar ambiente de desarrollo y CI/CD en la solución
- Crear los epics y tareas siguiendo una estrategia ágil de los cambios que se requieren hacer para implementar la solución de tu análisis

- Para crear epics click en el + del botón verde -> New Epic
- 

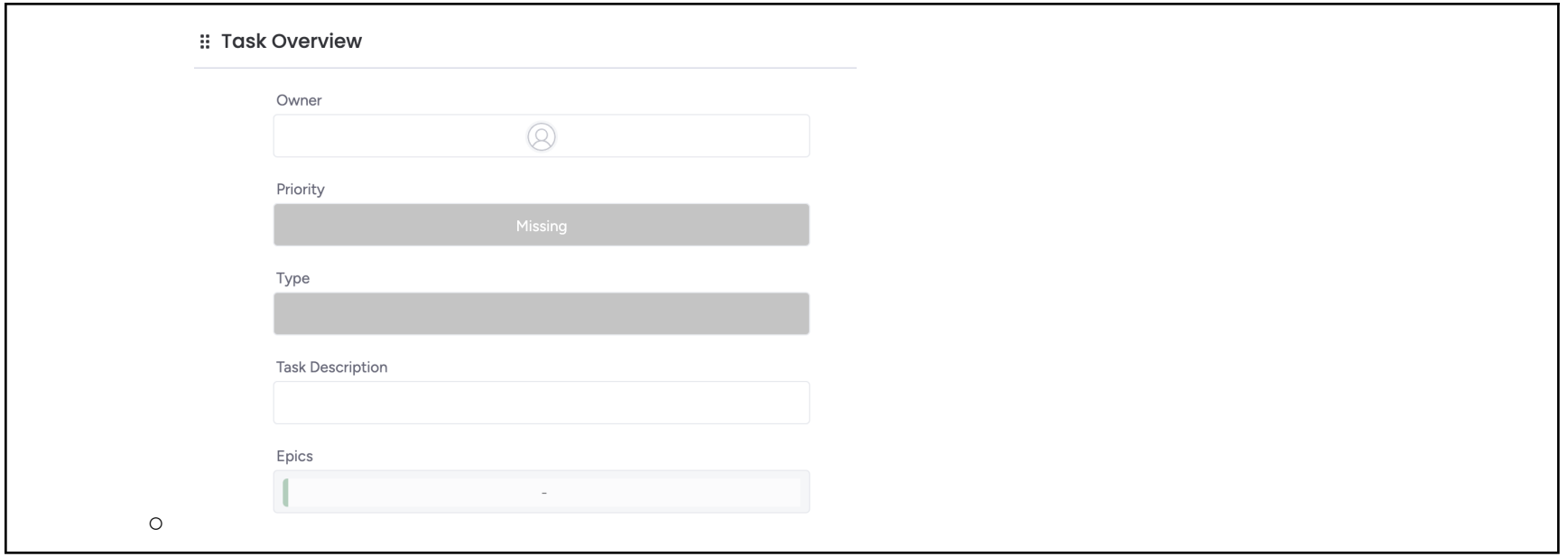
- Llenar los datos
- 

- Para crear una tarea, empieza a escribir en el campo Add task y enter
- 

- Abre la tarea para agregar más información
- 

- En el overview se puede asignar el epic al que pertenece


Task Overview
Owner
Priority
Missing
Type
Task Description
Epics
-

# Documentación



# Introducción

ACME es una aplicación de servicio de transporte bajo demanda para sus usuarios. Cualquiera que necesite moverse puede registrarse y reservar un vehículo para viajar de un origen a un destino. Cualquier persona que tenga un vehículo puede registrarse como conductor y elegir un viaje. Conductores y pasajeros se pueden comunicar a través de la aplicación en sus smartphones.

Digamos que a esta fecha el sistema tiene aproximadamente 500 millones de pasajeros y 5 millones de conductores. Asumamos las siguientes cifras:

- 20 millones de pasajeros y tres millones de conductores diarios.
- Esto sería a 20 millones de viajes diarios
- Cada 4 segundos los conductores envían una actualización de su ubicación

# Almacenamiento

| Entidad | Unidad | Unidades | Total |
|---|---|---|---|
| Pasajero | 1,000 bytes | 500 millones ($500 \times 10^6$) | 500 GB |
| Conductor | 1,000 bytes | 5 millones ($5 \times 10^6$) | 5 GB |
| Viaje | 100 bytes | 20 millones | 2 GB |

| | | | |
|---|---|---|---|
| | | $(20\times10^6)$ | |
| Nuevos pasajeros registrados | 1,000 bytes | 500,000 | 500 MB |
| Nuevos conductores registrados | 1,000 bytes | 100,000 | 100 MB |
| Ubicación de conductor | 36 bytes | 5 millones $(5\times10^6)$ | 180 MB |
| Almacenamiento Diario | | | 2.78 GB |
| Almacenamiento Anual | 2.78 GB | 365 días | 1.01 TB |

# Ancho de banda requerido

Asumamos que la transmisión de los datos del conductor y el pasajero (datos como el perfil) son muy pequeños como para ser considerados. Esto nos deja con el viaje y la ubicación del conductor:

| Entidad | Unidad | Unidades | Total |
|---|---|---|---|
| Viajes | 100 bytes | 20 millones (20 millones de viajes al día / | 23.4 kb ≈ 185 kbps |

| | | | |
|---|---|---|---|
| | | 86400 segundos en el día es igual a 232 viajes por segundo) | |
| Ubicación de conductor activo | 19 bytes | 3 millones | 57 MB ≈ 456 octetos cada 4 segundos = 114 Mbps |
| Total | | | 114.19 Mbps |

# Requerimientos

## Requerimientos Functionales

- Actualizar la ubicación del conductor: El conductor es una entidad en movimiento por lo cual su ubicación debe ser actualizada por intervalos de tiempo.
- Mostrar conductores cercanos a un pasajero
- Pedir un viaje: Un pasajero debe ser capaz de pedir un viaje, y los conductores cercanos deben ser notificados sobre el nuevo viaje.
- Sistema de pago: Al inicio del viaje, el sistema debe empezar el proceso de pago y administrar otros pagos (propinas, uso de tag, etc.).
- Mostrar al pasajero el tiempo estimado de llegada del conductor
- Confirmar viaje: Conductores deben ser capaz de confirmar que tomarán el viaje.
- Mostrar actualizaciones del viaje: Una vez que el conductor y el pasajero han aceptado el viaje, deben de poder ver actualizaciones del viaje como el tiempo estimado de llegada y la ubicación actual hasta que su viaje termine.
- Finalizar el viaje: El conductor marca que ha terminado el viaje hasta su destino y que está disponible para otro viaje.

## Requerimientos No Funcionales

- Disponibilidad: El sistema debe estar altamente disponible. Un tiempo de caída, por mínima que sea, puede provocar que el proceso

de viaje falle, ejemplo el conductor no pudiendo ubicar un pasajero, o el pasajero no pudiendo encontrar al conductor.

- Escalabilidad: El sistema debe de poder manejar un incremento de usuarios (picos de temporalidad).
- Robusto ante fallos: El sistema debe ser rápido y libre de fallos mientras provee el servicio. Peticiones de viaje y actualizaciones de ubicaciones deben de estar disponibles siempre.
- Consistencia: Los conductores y pasajeros deben tener una vista consistente del sistema.
- Detección de fraude: El sistema debe tener la capacidad de detectar actividad sospechosa de fraude en los procesos de pago.

# Arquitectura de la Aplicación ACME App

La Figura 1 muestra una descripción de la propuesta de arquitectura de la aplicación de ACME.

Figura 1. Arquitectura de la Aplicación

Los siguientes son los principales contenedores de la aplicación ACME App:

- Frontend. Se trata de la aplicación móvil usada por pasajeros y conductores para ofrecer el servicio de viajes.
- Backend. Se trata de las entidades serverless usadas para ejecutar las acciones realizadas por los usuarios y presentar información.
- Proveedores de Pago. Son entidades terciarias que procesan los pagos entre el sistema y los bancos.

Los componentes y funcionalidades son descritos con mayor detalle en las siguientes secciones del documento:

- *Frontend* de la Aplicación
- *Backend* de la Aplicación
- Proveedores de Pago

# Frontend de la Aplicación

Esta sección describe los elementos, propiedades y relaciones que soportan el contenido y el contenido estático de la aplicación ACME App.

El *frontend* de la aplicación ACME App tiene las siguientes características:

- Usa React Native para construir una aplicación utilizada en diferentes sistemas operativos
- La aplicación debe ser entregada al Store de cada plataforma para ser instalada por los usuarios
- Se tiene una guía de buenas prácticas por plataforma para la construcción e implementación de la aplicación
- La aplicación debe ser capaz de distinguir entre pasajeros y conductores
- El pago debe procesarse en la aplicación a través de diferentes formas de pago:
    - Tarjeta
    - Intermediarios de pago como Paypal
- Se requiere llevar un control de las capacidades de la aplicación en cada versión para tener una trazabilidad de errores
- A la vez se debe dar soporte a la aplicación de acuerdo a los errores reportados se deben implementar nuevas capacidades de acuerdo a la investigación de mercado

# Backend de la Aplicación

Esta sección describe los elementos, propiedades y relaciones que soportan los servicios de la aplicación ACME App, el almacenamiento de recursos y la base de datos. El *frontend* interactúa con los elementos del *backend* que se encuentran en la nube.

El *backend* incluye los siguientes servicios de AWS:

- AWS Lambda. Consiste en un servicio que ejecuta código cuando es necesario y sin proveer o administrar servidores. Ejecuta los servicios de *backend* de la aplicación.
- Amazon Keyspaces. Servicio de base de datos no relacional basado en Apache Cassandra.
- Amazon DynamoDB. Servicio de base de datos no relacional basado en documentos.
- Amazon S3. Consiste en un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento. Cuenta con características para organizar los datos y configurar controles de acceso.

Las funcionalidades de los servicios serverless son:

- Crear un viaje
- Encontrar conductor más cercano

- Calcular costo de viaje
- Mostrar información del viaje
- Actualizar estado del viaje (creado, conductor encontrado, confirmado, pick up, en progreso, finalizado)
- Calcular el tiempo estimado de llegada
- Procesar pagos
- Actualizar la ubicación

La aplicación móvil hace una petición al backend indicando que requiere iniciar un viaje indicando origen, destino e información del pasajero. Se activan los siguientes procesos

- Crear el viaje: inserta en DB el viaje con su información
- Buscar un conductor cerca: hace uso del QuadTree, el origen del viaje y las ubicaciones proporcionadas por los conductores

El resultado de este proceso es mostrar una notificación a un conductor. Si el conductor acepta entonces:

- Se actualiza el estado del viaje a conductor encontrado
- Se calcular el costo del viaje
- Se muestran los datos del conductor

Si el conductor rechaza el viaje, se hace la búsqueda descartando al conductor que rechazó el viaje. El pasajero se mantiene en espera hasta que se encuentre un conductor.

Si el pasajero acepta el costo del viaje:

- Viaje cambia a estado confirmado
- Se calcula tiempo estimado de llegada

El conductor sigue actualizando su ubicación cada 4 segundos. La ubicación del conductor se muestra al pasajero. Cuando el conductor recoge al pasajero, el viaje se actualiza como en progreso. Durante el tiempo de vida del viaje, el tiempo estimado de llegada es calculado.

Cuando el conductor finaliza el viaje, el estado es actualizado y se procesa el pago.

# Proveedores de Pago

Los proveedores de pago son intermediarios entre el backend y los bancos para procesar los cargos generados por el servicio ofrecido. Algunos ejemplos son Paypal, Discovery, etc.

Los requerimientos para la comunicación son:

- Se requiere de una conexión segura, cifrada.
- La información debe contener lo mínimo necesario para que el pago sea aprobado
- Debe existir un histórico de respuesta de los proveedores
- Se requiere de una implementación anti fraude
- Algunos proveedores requieren asegurar la comunicación usando IPs específicas