TP Cuidandonos

Alumnas	● Jimena Cori ● Melani R Lorenzo
Curso k3001 k3101	Martes - Mañana
Fecha entrega	07/05/2024

Punto 1
1)

	Solución a: Firebase	**Solución b: Proceso Propio**
Mantenibilidad	**Ventajas:** - Fácil implementación y mantenimiento con Firebase ya que proporciona una API bien documentada y herramientas para el desarrollo móvil. - Simplificación del desarrollo gracias a la infraestructura gestionada.	**Ventajas:** - Mayor control y flexibilidad en el desarrollo, más control en el código lo que facilita la adaptación a cambios - Independencia de terceros, menos limitaciones en la personalización y más control en la infraestructura y mantenimiento.
	Desventajas: - Dependencia de la infraestructura y cambios en la API de Firebase. - Limitado en términos de personalización y flexibilidad en comparación con una solución de proceso propio.	**Desventajas:** - Mayor complejidad de desarrollo. - Responsabilidad total del equipo de desarrollo, puede requerir recursos extras.

	Solución a: Firebase	Solución b: Proceso Propio
Disponibilidad	**Ventajas:** - Gestión de aspectos como la replicación de datos por parte de Firebase. - Ofrece alta disponibilidad y escalabilidad, es menos probable que experimente tiempos de inactividad debido a problemas de infraestructura.	**Ventajas:** - Mayor control sobre la disponibilidad, el equipo puede implementar estrategias específicas para garantizarla. -El sistema es menos vulnerable a problemas de disponibilidad relacionados con servicios externos.
	Desventajas: -Al depender de terceros, el sistema de mensajería podría verse afectado y el equipo de desarrollo tendría poco control directo sobre la solución. - Firebase podría realizar cambios en su API o en sus políticas que afecten la forma en que funciona el sistema y los ajustes que requiera podrían afectar la disponibilidad.	**Desventajas:** - Mantener esta solución puede requerir más recursos en términos de tiempo, personas y costos de infraestructura. - Cuenta con un mayor riesgo de errores que pueden afectar la disponibilidad.

2)

La aplicación deberá tener una capa de visualización que corra sobre el sistema operativo del smartphone y la lógica de negocio implementada en un servidor en la nube.

Como la lógica de negocio se mueve a un servidor en la nube, al utilizar servidores más potentes y escalables, se reduce la carga en el dispositivo móvil, se facilita la gestión de un mayor número de usuarios y transacciones, se garantiza la escalabilidad, mejorando así el rendimiento. Por otro lado, los usuarios pueden acceder a la aplicación desde diferentes sistemas operativos, por lo tanto ofrece portabilidad. Además, las actualizaciones y mejoras en la lógica de negocio pueden implementarse de manera eficiente en el servidor, sin necesidad de actualizar cada dispositivo individualmente.

☑ rendimiento ☑ portabilidad ☑ eficiencia

El cálculo de la distancia deberá hacerla la propia aplicación.

El cálculo de la distancia es un requerimiento crítico para el funcionamiento de la aplicación, sin la distancia no se podrá calcular el tiempo de demora, entonces no podrá notificar si algo malo sucedió. Por lo tanto, si esta parte es ejecutada por un componente de terceros, el

funcionamiento de la aplicación estaría dependiendo fuertemente de este componente. Para evitar el riesgo de interrupciones o problemas de rendimiento causados por el proveedor externo, puede ser preferible desarrollar el cálculo de la distancia internamente para tener un mayor control y mitigar el riesgo de dependencia.

- ☑ rendimiento
- ☑ funcionalidad

El dominio podría ser implementado en una base de datos no relacional

El dominio podría ser implementado en una base de datos no relacional debido a su capacidad escalable y flexible, que se adapta bien a las necesidades cambiantes del sistema y a su posible crecimiento en usuarios y datos. Además, las bases de datos no relacionales proporcionan un rendimiento eficiente en operaciones importantes como calcular la distancia entre ubicaciones y gestionar notificaciones en tiempo real.

- ☑ rendimiento
- ☑ eficiencia

Punto 2

1)

Se utilizaron adapters para calcular la distancia para priorizar la mantenibilidad y extensibilidad del código en caso que en un futuro la API para calcular distancias cambie.En el caso del notificador push, es útil para abstraer y encapsular la lógica de envío de notificaciones push a través de diferentes proveedores de servicios de mensajería, lo que haría el sistema ser compatible con múltiples plataformas de notificación push.
Para modelar las reacciones se pedía considerar que pueden surgir nuevas formas de reaccionar frente a un incidente y que el usuario puede cambiar esta configuración cuantas veces quiera. Entonces por eso optamos por un patrón strategy con interfaz para asegurar extensibilidad si se necesitan incorporar nuevas formas.

2)

```
public class Trayecto {
  ...
        public double calcularTiempoDemoraAproxIncluyendoParadas(calculador:
CalculadorDeTiempoDemora){
                double demoraEnMinutos;

                if(paradas.all(parada -> parada.seDemora())
                        demoraEnMinutos = paradas.sum(parada ->
parada.calcularTiempoDemoraAprox(calculador);
                else
                        demoraEnMinutos = this.calcularTiempoDemoraAprox(calculador);

                return demoraEnMinutos;
```

```
        }
}


public class Parada {
   …
        public double calcularTiempoDemoraAprox(calculador:
CalculadorDeTiempoDemora){
                double demoraEnMinutos;
                demoraEnMinutos = calculador
                                .calculadorDeMinutosDemora(this.origen.obtenerDireccion(),
                                this.destino.obtenerDireccion())
                                + this.demoraMinutos;

                return demoraEnMinutos;
        }
}
```