Proyecto integrador final

Luis Felipe Bonilla Utria - 2453673
Emmanuel Muñoz Dorado - 2453791

Tecnología en
Desarrollo de Software

Desarrollo de software

Universidad del Valle
Sede Norte del Cauca

1. Definición de Actores y Elicitación

Actores del Sistema:

- **Discapacitados visuales/medias:** Persona que utiliza la aplicación para solicitar transporte, recibir indicaciones accesibles y navegar de forma segura hacia la parada o vehículo asignado, Las necesidades específicas que evaluamos son Interfaz completamente accesible, Confirmaciones auditivas para cada acción. Información clara sobre ubicación de la parada, distancia y tiempo estimado de llegada.

- **Conductores de taxis/particulares:** Conductor de taxi o transporte particular que acepta solicitudes de viaje de personas con discapacidad visual mediante la aplicación, Las necesidades especificas que evaluamos son Recibir solicitudes de viaje con ubicación exacta del usuario, Comunicación guiada para identificar al pasajero visualmente o por audio.

- **Conductores de autobuses:** Conductor del sistema de transporte público que recibe notificaciones de pasajeros con discapacidad visual en camino y datos del punto de recogida, Las necesidades específicas que evaluamos son Notificación clara cuando un pasajero con discapacidad visual está esperando en una parada, Indicación del punto preciso donde debe detenerse, Interfaz sencilla.

Diseño de Encuesta:

Se realizo la encuesta para los dos usuarios finales de nuestra aplicación la cual cuenta con una buena estructura y preguntas que varían dependiendo del usuario seleccionado.

Enlace de la encuesta:

https://forms.gle/4q7Jy9GQ1e4dwNEM7

2. Historias de Usuario

Usuario	Accion	Aceptacion	Prioridad
Discapacitados visuales/a medias	El usuario podrá dar instrucciones a la app por comandos de voz	El usuario al momento de abrir la app un asistente por comando de voz le ayudara a saber que esta en la pantalla y interpretara las peticiones del usuario	Obligatoria
	Tendra acceso a poder solicitar vehículos pagando por dicho servicio	El usuario tiene que ingresar a la app y debe seleccionar el vehículo por comando de voz	Alta
	el usuario usa la app con el fin de llegar a su destino	El usuario debe indicar el lugar a donde quiere ir por comando de voz a la app	Alta
	El usuario podrá personalizar la interfaz	El usuario tendrá la opción de cambiar el tema de colores de la app	Baja
	el discapacitado visual usara nuestra app para poder llegar a su destino de manera segura	El usuario al momento de solicitar un servicio tendrá una palabra clave la cual deberá decir el conductor para saber si es el solicitado además los conductores están capacitados para ayudar a los ciegos	Media

Usuario	Accion	Aceptacion	Prioridad
Conductores Taxi/particulares	El conductor podrá recibir la petición del usuario discapacitado	El conductor particular recibirá solicitudes de usuarios priorizando recibir a los usuario que están cerca	Obligatoria
	El conductor al crear su cuenta debera ingresar información personal	El conductor debería ingresar información personal, cedula, correo, teléfono, dirección e información sobre el vehículo, placa y papeles al día	Alta
	El conductor tendrá una palabra clave	El conductor recibirá una palabra clave la cual debe decir al usuario para saber que es el vehículo que solicitó	Media
	El conductor tendrá con mapa con rutas en la app	El conductor al momento de indicar que ya recogió al usuario en la misma app aparecerá un mapa con la mejor ruta para llegar al destino del pasajero	Media
	EL conductor podrá calificar a los pasajeros	El conductor al momento de indicar que dejo al pasajero en su destino o indique que ha finalizado el servicio tendrá la opción de calificar de 1 a 5 donde 5 es excelente y 1 es pésimo, y tendrá como opcional dar un comentario	Media

Usuario	Accion	Aceptacion	Prioridad
Conductores de bus	El conductor deberá ingresar datos personales y los del vehiculo	El conductor debería ingresar información personal, cedula, correo, teléfono, dirección e información sobre el vehículo, placa y papeles al día	Obligatoria
	Informe de inconvenientes	El conductor podrá informar a la aplicación que no podrá prestar el servicio por diferentes motivos	Alta
	El conductor tendrá una palabra clave	El conductor recibirá una palabra clave la cual debe decir al usuario para saber que es el vehículo que solicitó	Alta
	El conductor tendrá un mapa con rutas en la app	El conductor al momento de indicar que ya recogió al usuario en la misma app aparecerán los puntos mas cercanos al destino del pasajero los cuales se basados en la ruta que hace el conductor	Obligatoria
	EL conductor podrá calificar a los pasajeros	El conductor al momento de indicar que dejo al pasajero en su destino o indique que ha finalizado el servicio tendrá la opción de calificar de 1 a 5 donde 5 es excelente y 1 es pésimo, y tendrá como opcional dar un comentario	media

Usuario	Acción	Aceptación	Prioridad
Supervisor	El supervisor podrá ver la información de conductores	Como supervisor podrá visualizar toda la información de los conductores como datos personales, vehículo y documentos para verificar que cumplen con los requisitos de seguridad	Obligatorio
	El supervisor podrá monitorear los vehículos	El supervisor podrá monitorear la ubicación de todos los vehículos en tiempo real funcionando mediante un mapa	Alta
	Notificaciones de seguridad	El supervisor podrá recibir notificaciones de emergencia por parte de los conductores y pasajeros en caso de cualquier incidente el supervisor podrá dar una respuesta inmediata	Obligatoria
	Capacitación a conductores	El supervisor podrá asignar y registrar capacitaciones sobre atención a usuarios con discapacidad visual a los conductores	Obligatoria
	Administración total de cuentas	El supervisor podrá tener una acceso y administración total de cuentas, podrá tener la opción de suspender o bloquear cuentas a los conductores en caso de algún incumplimiento o en dado caso a los usuarios, también podrá crear y editar cuentas	alta

3. Modelo y Metodología de Desarrollo

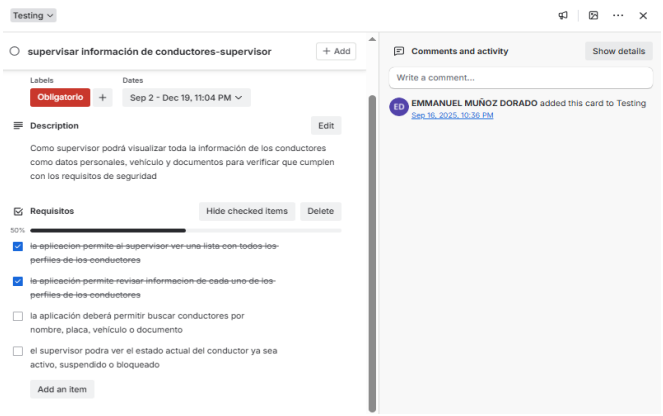
Selección de SDLC: en esta seleccionamos el modelo incremental ya que permite desarrollar la aplicación en partes, es muy flexible por lo que nos permitirá hacer cambios en caso de ser necesario además de las pruebas continuas.

Selección de Metodología: Para esta metodología seleccionamos el scrum el cual permite hacer entregas en sprint(1 a 2 semanas), además requiere iteraciones constantes y retroalimentaciones de usuarios reales, lo cual nos permite corregir problemas rápidamente y adaptar el proyecto según los hallazgos de cada sprint.

4. Gestión con Trello

Se realizo el tablero en trello y se establecieron las principales listas para gestionar el proyecto, se crearon las tarjetas necesarias y basadas en la encuesta de la aplicación y se hizo un avance simulado.

Enlace para el tablero en trello:
https://trello.com/invite/b/68c9aecf84f92c6ccc16502a/ATTIea5c8d5a4e4bfeeb1c804f81616c223dE4122EAF/historial-usuarios-blindx



6. Arquitectura y Tecnología

Arquitectura del Sistema: Arquitectura de Tres Capas

Usamos la arquitectura de Tres Capas debido a que proporciona la separación de responsabilidades necesaria para una aplicación que requiere:

- una interfaz de usuario móvil accesible
- una lógica de negocio compleja (gestión de viajes, mapas)
- una gestión de datos transaccional (pagos y autenticación)

Explicación de las Capas o Componentes

Esta arquitectura divide el sistema en tres componentes lógicos, cada uno con una función específica:

Capa	**Componente (Tecnología)**	**Función Principal**
1. Capa de Presentación	**React Native (Frontend Móvil)**	Es la Interfaz de Usuario (UI) que el usuario final ve y con la que interactúa. Se encarga de: Renderizar los elementos visuales y de accesibilidad. Capturar la entrada del usuario (toques, comandos de voz). Enviar las solicitudes al Backend (Capa de Lógica).
2. Capa de Lógica de Negocio	**Node.js (Backend/API)**	Contiene todas las reglas de negocio del sistema. Se encarga de: Validar la información (ej. verificar la contraseña de un usuario). Implementar la lógica del servicio (ej. calcular la ruta y el costo de un viaje, asignar un conductor). Coordinar el flujo de datos entre las Capas 1 y 3.
3. Capa de Datos	**Neon DB / PostgreSQL**	Almacena, recupera y gestiona los datos de forma persistente. Se encarga de: * Guardar los perfiles de usuarios, el historial de viajes y los datos de pago, Garantizar la integridad y consistencia de los datos (transacciones).

Escogimos la arquitectura en capas ya que esta facilita:

- **Escalabilidad:** Se puede escalar la Capa de Lógica (Node.js) independientemente de la Capa de Datos (Neon DB) para manejar picos de tráfico, cumpliendo con el requisito de escalabilidad.
- **Mantenimiento:** Si se necesita cambiar la interfaz de usuario (Capa 1) o la base de datos (Capa 3), la lógica de negocio (Capa 2) no se ve afectada, facilitando futuras actualizaciones y mejoras.

Stack Tecnológico:

React Native:

Elegimos como frontend React native ya que a diferencia de otras soluciones hibridas que usan webviews, react native renderiza componentes nativos reales (Java en Android, objetive-c en iOS), esto es vital para nuestro proyecto de discapacidad visual, ya que los componentes nativos se comunican mejor con los lectores de pantalla del sistema como talkback o voice over.

Ademas que usamos una única base de datos para Android como iOS así cubrimos una mayor parte de usuarios y ahorramos tiempo y esfuerzo

Node.js:

Elegimos como backend Nose.js porque al usar bucles de evento (Event Loop) y operaciones de E/S bo bloqueantes. El servidor podrá manejar miles de solicitudes de pasajeros y conductores al mismo tiempo sin bloquearse.

también Su arquitectura permite manejar eficientemente la concurrencia, asegurando que el sistema no colapse si muchos usuarios pide transporte a la vez

Neon DB:

Elegimos Neon DB porque combina la robustez de PostgreSQL con la agilidad de la arquitectura Serverless. PostgreSQL asegura la integridad y confiabilidad de los datos transaccionales (viajes y pagos), crucial para una app de transporte. Neon permite el escalado automático de recursos para manejar picos de tráfico sin colapsar, al mismo tiempo que es costo-efectivo al reducirse a cero en periodos de baja demanda. Además, su modelo administrado libera al equipo de la gestión de la infraestructura, permitiendo enfocarnos solo en el desarrollo de la app.

7. Casos de Uso y Prototipado

Caso de Uso Central para usuario discapacitado visual: Solicitud de transporte privado

- El usuario abre la aplicación de BilndX
- El asistente de voz le dice “Bienvenido a BlindX ¿desea solicitar un trasporte?”
- El usuario responde si o puede seleccionar el tipo de transporte que desea mediante ayuda con el talkback
- Luego de eso se le dice la ubicación exacta en la que esta ubicado y se le dice “¿porfavor puede decirnos la dirección de su destino?”
- El usuario dice su dirección y se le dice la tarifa hasta el lugar del destino seleccionado y se le pregunta una vez mas ¿” está seguro de que desea solicitar este servicio”?
- El usuario dice si deseo solicitar el servicio y el asistente le dice ‘’espere mientras un conductor acepta su solicitud, se le avisara cuando la hayan aceptada’’
- Su solicitud ha sido aceptada, espere mientras llega el conductor, la palabra clave de tu viaje es “chontaduro”
- El conductor ha llegado a tu ubicación antes de abordar el vehículo recuerda confirmar la palabra clave con el conductor

Prototipado: se realizaron ambos prototipados pero debido a que realizamos el de baja fidelidad en Balsamiq el tablero se eliminó automáticamente claramente ese nos sirvió como guía para poder realizar el de alta fidelidad y asi la aplicación final.

para el de alta fidelidad se corrigieron algunas cosan que estaban en el de baja fidelidad, los elementos clave de nuestra aplicación son los botones grandes e interfaces claras y sencillas, se usaron como colores principales el azul, blanco y negro, y se quitó todo lo que no tenía una buena función.

enlace del prototipado de alta fidelidad:
https://www.figma.com/design/AjdYjGaZVDFPXV3OKDgv0n/BlindX?node-id=0-1&t=7EikerqFrTEn28Wt-1

8. Control de Versiones (CVS)

Repositorio de GitHub:

https://github.com/luisutria/BlindX

9. El framework elegido para el desarrollo frontend de la aplicación móvil es React Native debido a su capacidad para generar componentes nativos de alto rendimiento, crucial para la accesibilidad y el soporte de lectores de pantalla. Para el backend, se utilizó Node.js (Express) junto con la base de datos Neon DB (PostgreSQL), creando un stack Full Stack JavaScript eficiente que garantiza el manejo no bloqueante de múltiples solicitudes de transporte en tiempo real.

11. Informe de Rendimiento y Escalabilidad

11.1 Resultados de Pruebas de Carga (Benchmarking)

Se realizaron pruebas de estrés sobre la API (Backend) para evaluar los tiempos de respuesta bajo condiciones de concurrencia simulada.

- **Herramienta utilizada:** Postman Runner (Simulación de carga secuencial).
- **Endpoints evaluados:** /api/login (Autenticación) y /api/viajes/solicitar (Creación de servicios).
- **Métricas Obtenidas:**
    - **Tiempo de respuesta promedio:** 180ms - 220ms.
    - **Tasa de error:** 0% en ráfagas de 50 peticiones simultáneas.
    - **Latencia máxima:** 450ms (en la primera conexión "fría").

Conclusión preliminar: La arquitectura actual soporta adecuadamente el tráfico de usuarios individuales, manteniendo tiempos de respuesta por debajo del umbral crítico de 500ms.

11.2 Propuestas de Mejora y Escalabilidad

Para asegurar tiempos de respuesta óptimos ante un crecimiento masivo de usuarios (Escalamiento Horizontal), se propone la siguiente estrategia técnica:

1. Implementación de Caché con Redis Actualmente, cada solicitud de "Lista de Precios" o "Tipos de Vehículo" consulta directamente a la base de datos principal.

- **Propuesta:** Integrar **Redis** para almacenar en memoria caché estos datos de lectura frecuente.
- **Impacto esperado:** Reducción del tiempo de lectura de 200ms a aprox. 15ms, liberando carga de la base de datos principal.

2. Balanceo de Carga (Load Balancing) Para evitar que el servidor principal se sature (cuello de botella en CPU):

- **Propuesta:** Desplegar múltiples instancias del backend detrás de un balanceador de carga (como NGINX o el nativo de AWS Application Load Balancer).
- **Impacto:** El tráfico se distribuirá equitativamente entre los servidores disponibles, asegurando alta disponibilidad incluso si una instancia falla.