**Comparación de tecnologías**

**Lenguaje de Programación**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lenguaje** | **Dart** | **JavaScript** | **Kotlin** | **Elección Final** |
| **Pros** | Optimizado para Flutter, rápido, fácil de depurar, soporte sólido para aplicaciones multiplataforma. | Amplia comunidad, versátil, soporte extensivo para bibliotecas y frameworks. | Específico para Android, gran rendimiento y herramientas avanzadas para Android. | **Dart** |
| **Contras** | Comunidad más pequeña, enfoque limitado fuera de Flutter. | Puede ser complicado para aplicaciones móviles, gestión de asincronía compleja. | Solo aplicable para Android, no es multiplataforma. | Comunidad en crecimiento. |
| **Uso Ideal** | Aplicaciones móviles multiplataforma con alto rendimiento. | Aplicaciones web, híbridas, o con interacción ligera. | Apps Android nativas. | Apps modernas multiplataforma. |

**IDE/Editor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IDE/Editor** | **Visual Studio Code** | **Android Studio** | **Xcode** | **Elección Final** |
| **Pros** | Ligero, extensible, multiplataforma, ideal para desarrollo rápido. | Herramientas avanzadas para pruebas y depuración de Android. | Optimizado para desarrollo de apps iOS. | **Visual Studio Code** |
| **Contras** | Depuración limitada en simuladores complejos. | Requiere hardware potente, puede ser pesado. | Exclusivo para macOS, curva de aprendizaje inicial para usuarios no iOS. | Algunas funciones requieren extensiones adicionales. |
| **Uso Ideal** | Desarrollo ágil y multiplataforma. | Desarrollo de aplicaciones Android nativas. | Apps nativas para iOS. | Desarrollo de apps multiplataforma con Flutter. |

**Framework para la App Móvil**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Framework** | **Flutter** | **React Native** | **Ionic** | **Elección Final** |
| **Pros** | UI nativa, único código base, excelente rendimiento. | Amplia comunidad, buena compatibilidad con JS, compatible con librerías de terceros. | Compatible con web y móvil, fácil de implementar. | **Flutter** |
| **Contras** | Tamaño inicial grande, curva de aprendizaje inicial. | Depende de librerías externas para optimización de rendimiento. | Rendimiento inferior al nativo. | Curva de aprendizaje inicial. |
| **Uso Ideal** | Apps móviles de alto rendimiento con experiencia nativa. | Apps híbridas con funcionalidad estándar. | Apps ligeras, proyectos pequeños. | Apps móviles profesionales que requieran buen diseño y rendimiento. |

**Framework para la App Web**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Framework** | **React** | **Angular** | **Vue.js** | **Elección Final** |
| **Pros** | Popularidad global, fácil integración con bibliotecas de terceros, componentes reutilizables. | Escalabilidad, buenas prácticas integradas, soporte sólido de Google. | Ligero, fácil de aprender, curva de aprendizaje amigable. | **Angular** |
| **Contras** | Requiere configuración extra para backend. | Curva de aprendizaje empinada, requiere mayor conocimiento inicial. | Ecosistema más pequeño comparado con React y Angular. | Mayor tiempo para dominar características avanzadas. |
| **Uso Ideal** | Proyectos pequeños y medianos con componentes dinámicos. | Aplicaciones grandes y complejas con altos requisitos de escalabilidad. | Proyectos ligeros o con plazos de desarrollo cortos. | Apps escalables con gran cantidad de usuarios. |

**Base de Datos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Base de Datos** | **Firebase (Firestore)** | **Cassandra** | **MongoDB** | **Elección Final** |
| **Pros** | Integración nativa con Flutter, escalabilidad en tiempo real, solución sin servidor. | Diseñada para ofrecer disponibilidad continua sin puntos únicos de falla. Esto la hace ideal para sistemas que necesitan operar 24/7. | Escalable, flexible, fácil de usar para datos no estructurados. | **Firebase (Firestore)** |
| **Contras** | Limitado para consultas relacionales complejas. | Requiere experiencia técnica para configurar y mantener correctamente. Las configuraciones incorrectas pueden afectar el rendimiento y la disponibilidad. | No ideal para datos altamente estructurados. | Limitaciones en migración de datos. |
| **Uso Ideal** | Aplicaciones con datos en tiempo real y necesidades de sincronización instantánea. | Servicios de registro de actividades, aplicaciones IoT, y sistemas de monitoreo que generan datos continuamente. | Aplicaciones con alto volumen de datos y necesidades de escalabilidad horizontal. | Aplicaciones móviles modernas y multiplataforma. |

**Razones para la elección final**

**Dart:** Diseñado específicamente para Flutter, lo que asegura la mejor compatibilidad y rendimiento para aplicaciones móviles multiplataforma.

**Flutter:** Proporciona un desarrollo rápido con un diseño visual atractivo y un excelente rendimiento nativo.

**Angular:** Es especialmente útil para proyectos empresariales o aplicaciones que requieren una arquitectura robusta, una gran cantidad de funcionalidades integradas y un enfoque modular y limpio.

**Firebase:** Ofrece sincronización en tiempo real, escalabilidad, y una fácil integración con Flutter, perfecto para aplicaciones de transporte como la tuya.

**Visual Studio Code:** Ligero, multiplataforma, y extensible, ideal para desarrolladores que trabajan con Flutter.