ChatGPT

Технически и UX план за мобилно приложение за пътна помощ

Основна логика

- **Единна база данни**: Всички потребители (клиенти и шофьори) се съхраняват в Firestore. Всеки потребител има роля (клиент или шофьор) и съответни атрибути (напр. за шофьор – фирма, регистрационен номер на автомобила, снимка на колата и др.).
- **Статус на шофьора**: Всеки шофьор може да сменя своя статус между **„Активен“** и **„Неактивен“**. При статус „Активен“ шофьорът получава запитвания за помощ; при „Неактивен“ – не.
- **Подаване на заявка от клиент**: Клиентът попълва форма със следните данни: описание на проблема, една или няколко снимки (качени в Firebase Storage) и GPS локация (получена чрез `expo-location`). След потвърждение заявката се записва в базата данни (Firestore, колекция `requests`) с начален статус (напр. „Очаква шофьор“) и отметка за време.
- **Намиране на свободни шофьори**: Сървърна логика (Firebase Function) търси активни шофьори в радиус 5 км около адреса на заявката, използвайки GeoFirestore за геозапитвания във Firestore. Ако няма намерени шофьори, радиусът се разширява на 10 км, 15 км и т.н., докато се намери активен шофьор или се достигне лимит.
- **Изпращане на заявка към шофьорите**: Всички активни шофьори в обхвата получават push нотификация (чрез Firebase Cloud Messaging). В приложението им се появява екран с данните от заявката: описание на проблема, качени снимки и адрес/карта на локацията на клиента.
- **Приемане на заявката от шофьора**: Първият шофьор, който натисне бутона **„Приемам“**, придобива заявката. На неговия екран се появява поле за въвеждане на ориентировъчна цена. Шофьорът въвежда сума (в местна валута) и потвърждава офертата.
- **Показване на офертата на клиента**: Клиентът получава оферта, съдържаща име на фирмата, име на шофьора, регистрационен номер на автомобила и предложената цена. Тази информация се показва на нов екран или изкачащ прозорец. Клиентът вижда два бутона: **„Приемам“** и **„Отказвам“**.
- **Приемане или отказ на офертата**: Ако клиентът откаже офертата („Отказвам“), системата я препраща автоматично към други активни шофьори. Ако клиентът приеме („Приемам“), стартира процес на плащане.
- **Плащане на 30%**: При приемане на офертата системата таксува клиента 30% от предложената цена чрез Stripe. Тези 30% се превеждат към фирмената сметка. Ако плащането е успешно, статусът на заявката се обновява (напр. „Платена“).
- **Потвърждение за шофьора**: След успешно плащане шофьорът получава известие („Заявката е потвърдена“) и вижда адреса на клиента. Може да натисне бутон **„Отвори в Google Maps“**, който стартира Google Maps с навигация към посочените координати.
- **Приключване на поръчката**: Клиентът доплаща останалите 70% директно на шофьора на място (в брой или чрез POS терминал). След приключване приложението маркира поръчката като завършена и предлага опция за оценка или обратна връзка.

Екрани и UX

Екраните за клиента

1. **Регистрация/Вход**: Екранът съдържа полета за имейл и парола (с евентуално допълнителни данни като име и телефон) и бутони „Регистрация“ и „Вход“. След успешна регистрация или вход клиентът попада на главния екран.
2. **Главен екран**: Клиентът вижда своята текуща локация (например на карта чрез `react-native-maps`) и има голям бутон **„Заяви помощ“** (например в долната част на екрана). При натискане на този бутон се отваря формулярът за заявка. Тук може да има и меню или навигация към история на поръчките и текущ статус.
3. **Форма за заявка**: Екранът за заявка съдържа текстово поле „Описание на проблема“, бутон „Добави снимка“ (използва `expo-image-picker` за камера/галерия), и поле или бутон за локация (напр. „Моята локация“ използва `expo-location`). В долната част има бутон **„Изпрати заявка“**. При натискане на „Изпрати заявка“ формулярът се изпраща и клиентът се пренасочва към статус-екрана.
4. **Очакване на оферта**: След изпращане клиентът вижда екран със съобщение „Търсим най-близки шофьори“ и индикатор за зареждане. Предлага се бутон „Отказ на заявката“, ако желае да отмени. Когато някой шофьор приеме и изпрати цена, клиентът получава push известие и приложението автоматично показва екрана с офертата.
5. **Екран за оферта**: Показва се лого/име на фирмата, име на шофьора, регистрационен номер на автомобила и предложената цена. Тези детайли са подредени като карта или списък. Под тях има два бутона: **„Приемам офертата“** (зелен) и **„Отказвам“** (червен). Клиентът прави избор с един клик.
6. **Плащане**: Ако клиентът приеме офертата, приложението преминава към плащане. Показва се сумата и се изчисляват 30% (например „Плати 30% от 100 лв = 30 лв“). Клиентът вижда Stripe форма за въвеждане на данни на карта и бутон **„Плати 30%“**. След успешното плащане екранът показва „Плащането е успешно“.
7. **Статус на поръчката**: Главният екран се обновява според статуса на поръчката – например с надпис „Шофьорът е на път“ и ориентировъчно време на пристигане. Когато шофьорът пристигне, статусът се сменя на „Шофьорът е пристигнал“. Клиентът получава push известия при важни промени (напр. „Офертата е потвърдена“, „Шофьорът пристигна“). След приключване на услугата екранът може да предложи оценка на шофьора/услугата.

Екраните за шофьора

1. **Регистрация**: Шофьорът въвежда данни като име, телефон, фирма, регистрационен номер на автомобила, снимка на колата (и евентуално други документи). Екранът съдържа бутон **„Изпрати за одобрение“**. След попълване на данните акаунтът може да е в статус „Изчаква одобрение“, докато администратор го активира.
2. **Главен екран (статус Активен/Неактивен)**: След одобрение шофьорът влиза в главния екран, където има превключвател или бутон **„Активен/Неактивен“** (например ключ и/или бутон в горния десен ъгъл). По подразбиране шофьорът е „Неактивен“. Когато стане „Активен“, той започва да получава нови заявки. На екрана може да се показва карта с текущата му позиция.
3. **Нова заявка**: При пристигане на нова поръчка шофьорът получава push нотификация. Екранът с заявката показва кратко описание на проблема, карта/адрес на клиента (чрез

`react-native-maps` или текст) и снимки. Има два бутона: **„Приемам“** (голям и зелен) и **„Пропускам“** (сив/син), за да откаже.

4. **Въвеждане на цена**: Ако шофьорът натисне „Приемам“, се показва поле за въвеждане на ориентировъчна цена и бутон **„Изпрати оферта“**. Той въвежда сумата и изпраща офертата. След това екранът може да покаже съобщение „Офертата е изпратена, чакаме потвърждение“.
5. **Очакване на отговор**: След изпращане шофьорът остава с екран, показващ че чака решение от клиента. Ако клиентът откаже офертата, шофьорът получава известие „Вашата оферта беше отказана“ и се връща към статуса си „Активен“, готов за следващи заявки.
6. **Потвърждение и навигация**: При успешно плащане от клиента шофьорът получава известие „Заявката е потвърдена“. В приложението се появява адрес/маршрут на клиента. Има бутон **„Отвори в Google Maps“**, който стартира навигацията.
7. **Завършване на поръчката**: След изпълнена услуга шофьорът може да натисне бутон **„Завърших услугата“**. Това маркира поръчката като приключена в системата. (Може да има и опция за добавяне на снимка или коментар за приключване.)

Технологии

- **React Native + Expo + TypeScript**: Избрани за бърза крос-платформена разработка (iOS и Android) с единен код. TypeScript осигурява типова безопасност и по-добра поддръжка. Expo предоставя готови модули за GPS (`expo-location`), камери (`expo-image-picker`) и други хардуерни функционалности.
- **Firebase Firestore**: NoSQL база в реално време. Примерни колекции: `users` (потребители с роля клиент/шофьор), `requests` (заявки), `offers` (оферти на шофьори), `transactions` (плащания) и др. Firestore позволява слушатели, които автоматично обновяват UI при промяна на данните (напр. статус на заявка).
- **Firebase Storage**: Съхранява снимките. Когато клиент или шофьор качи снимка, получава URL адрес. Този URL се записва във Firestore (напр. в документа на заявката или в профила на потребителя).
- **Геолокация и карти**: За получаване на координати се използва `expo-location`. За показване на карта/маршрут – `react-native-maps`. За търсене на най-близки шофьори ползваме GeoFirestore или библиотека за геозапитвания по радиус във Firestore.
- **Firebase Cloud Functions**: Сървърната логика (обработка на заявки, търсене на шофьори, препращане при откази, изпращане на нотификации) се пише като облачни функции. Например функция, задействана при нов запис в `requests`, прави геозапитване и изпраща FCM съобщения до шофьори.
- **Stripe (плащания)**: Използва се Stripe API за обработка на плащания. Например сървърно се създава `PaymentIntent` за 30% от сумата, а в приложението клиентът въвежда данни за картата през Stripe SDK и потвърждава плащане.
- **Google Maps Deep Link**: Навигацията към адреса на клиента се осъществява чрез URL-схема за Google Maps. Например: `https://www.google.com/maps/dir/?api=1&destination=LAT,LNG`. Натискането на „Отвори в Google Maps“ отваря навигацията към тази дестинация.
- **Push Notifications (FCM)**: Използва се Firebase Cloud Messaging за изпращане на push известия. Изпращаме до шофьори (нови заявки) и до клиента (промени в офертата, статуси на поръчката). Cloud Functions задейства FCM при нужда.
- **Версионен контрол (GitHub)**: Целият код (клиентското приложение, Cloud Functions, конфигурации) се съхранява в GitHub. Използва се контрол на версиите (всички промени, pull

request-и и т.н.). Може да се настрои CI/CD (напр. GitHub Actions) за автоматизирани билдове и тестове при всеки комит.

Бизнес логика

- **Комисионна 30%**: Клиентът заплаща 30% от окончателната цена през приложението (Stripe), които отиват във фирмената сметка. Останалите 70% се доплащат на шофьора извън системата (в брой или чрез POS).
- **Еднократно плащане**: Приложението таксува клиента само веднъж – при приемане на офертата (30%). След това не се правят допълнителни транзакции през приложението. След успешно плащане заявката се маркира като „Потвърдена“.
- **Рециклиране на заявки**: Ако клиентът откаже офертата на даден шофьор, системата автоматично препраща заявката към други активни шофьори. Процесът продължава, докато някой приеме офертата или не останат налични шофьори.
- **Разширяване на радиуса**: Системата започва търсене с радиус 5 км и при нужда го увеличава (10 км, 15 км и т.н.), за да осигури бързо намеряне на шофьор.
- **Таймаут за отговор**: Шофьорите имат ограничено време (напр. 1-2 минути) да отговорят на заявка. Ако не реагират, заявката се освобождава и отива при друг шофьор. Тези правила са конфигурирани в бекенда (Cloud Functions).
- **Статистика и отчети**: Системата записва всички поръчки, оферти и плащания. Администраторът може да преглежда статистики – брой заявки, приходи (30% от комисионата), активни шофьори и др.

Публикуване в Google Play и App Store

- **Expo EAS Build & Submit**: Използваме Expo Application Services за генериране и публикуване на билдовете. Потребителят инсталира EAS CLI (`npm install -g eas-cli`) и се логва (`eas login`). В конфигурацията (`eas.json`) се задават профили за билд (например `production`). След това стартираме `eas build --platform android` и `eas build --platform ios`, за да получим подписани пакети (AAB и IPA). След успешното билдване изпълняваме `eas submit --platform android --latest` (и аналогично за iOS), като подаваме нужните ключове (Google Service Account JSON за Android и App Store API ключ за iOS).
- **Настройки на акаунти**: Създава се Google Play Developer акаунт и Apple Developer акаунт. В `app.json` се попълват `android.package` (идентификатор на приложението) и `ios.bundleIdentifier`, както и други метаданни (икона, име на приложението, описание и др.).
- **AI помощ (ChatGPT/Cursor)**: Потребител без дълбоки технически познания може да ползва AI за напътствия. ChatGPT или Cursor може да генерира примерен `eas.json`, да обясни как да създадем Google Service Account или да подскаже как да решим грешка при изпращане. Expo документацията е ясна, а AI може да води стъпка по стъпка и да обяснява непознати термини.
- **Автоматизации и поддръжка**: За дългосрочна поддръжка може да се настрои CI/CD. Всеки път при нова версия в GitHub репозитория може автоматично да се стартира EAS build & submit (например чрез GitHub Actions). ChatGPT може да помогне при писането на тези скриптове. Така целият процес на билд и публикуване остава лесен и достъпен дори за неспециалисти.
