CS 490 Project: NexusBird

A cross-platform mobile application for long-distance

By Omid Rooholfada
The Problem
Real-world Problem

Travelling between cities is expensive, uncomfortable, and logistically difficult.

People who travel between cities (students travelling to and from the airport, long-distance commuters, people visiting family in distant cities), are faced with a wide set of less-than-ideal choices. None of these choices nail all three relevant categories: cost, comfort, and convenience.

Carpooling handles cost and comfort, at the expense of convenience due to lack of availability. In particular, travellers don't know which rides are headed in their direction at their desired time, so many potential matches are missed.
Our Solution

NexusBird is an app that matches intercity drivers with riders headed in the same direction.

The riders split the gas cost of the driver while getting a comfortable ride to their destination.

Turn an $80 UberX into a $20 carpool ride.
User Journeys
Rider - Signing up / Logging in

Standard Login / Signup with Facebook and Google SSO
Rider - booking a ride

Riders specify their starting and ending locations, departure time, and number of passengers. They’ll get pricing information and be prompted to enter their payment information if they haven’t already.
Driver - Signing up for a drive

Drivers have similar flow to riders — except they select the “Drive” tab on the home page.
Riders + Drivers - In-app Chat

Once placed in a carpool group, drivers and riders can chat to coordinate any last-minute logistics that come up (in most cases this won’t be necessary).
View and Modify Settings

User can view and edit profile as well as payment info. Can also view ToS and privacy policy.
Rider - payments

User can store their credit card info so that they don’t have to repeat it for every ride. Drivers can also withdraw their payments.
Code Architecture + Technology
High-level Components of Codebase

- Mobile App Frontend (./frontend)
  - Mobile application (ios/Android)
- Server (.server)
  - Hosted server running a authenticated API
- Database (.server/models.py, ./alembic)
Frontend

- React Native mobile application
  - This allows us to create iOS and Android apps with a single codebase
- React MobX
  - A “store” of data that allows us to organize information on the frontend and limit the number of requests we make to the server while reducing latency
- Apollo
  - A GraphQL client
- React Native elements
  - A UI framework for React Native
Backend

- Python Flask Server
- Hosted on Google Cloud Platform using a VM instance
- Graphene
  - A library for integrating GraphQL into a Flask server
- Pipenv
  - Package and virtual environment management
- Firebase for managed Authentication
Database

- Postgres database (see schema defined in models.py)
- Indices added, including indices for location coordinates
- SQLAlchemy ORM
  - A Flask / SQL ORM with out-of-the-box functionality for DB connectivity and queries
- User auth information was stored in Firebase, not our DB
  - For both security and development speed
- Alembic
  - Database migration library that integrates with SQLAlchemy
Third-Party APIs

- Expo
  - A managed React Native solution with a large set of APIs to integrate other third party services with React Native

- Stream Chat
  - Out-of-the-box chat API — they provide a backend messaging system as well as a React Native library for displaying a chat view

- Stripe
  - Handling payments between riders and drivers

- SendGrid
  - Send users email notifications for transaction receipts, reminders, etc.

- Many other NPM modules (see package.json)
Note on GraphQL

- GraphQL is a query language for making HTTP requests, intended as a replacement for REST
- A GraphQL API has only one endpoint, and the client can always ask for exactly the information it needs
- Benefits:
  - Allows us to make data-driven changes to client queries without having to change backend
  - Allows us to ask for exactly the information we need, not extras
Software Development Practices
Mobile development

Using the Expo CLI and the iOS simulator, I could view my mobile app on my desktop while developing, with helpful features like a debugger and hot reload.
Version Control and task-tracking

- Used BitBucket for version control
  - BitBucket-slack integration to see my repo updates in a Slack channel
- Used Sourcetree as a version control GUI
- Tracked tasks in Google Sheets relatively informally (complexity was low with just one person)
Deployment Process

- Expo
  - Made it easy to create iOS and Android builds that we could launch to the Apple and Google Play consoles (wrote scripts in the ./scripts folder to streamline this process further)
  - Over-the-Air updates, allowing me to post updates to the app that users can get without having to manually download them from App Store

- TestFlight
  - Allowed me to distribute the mobile app for beta testing while bypassing App Store approval process
Connection to CS education (by specific classes)
OOP - relevant concepts

- **Separation of concerns**
  - Splitting React Native UI into reusable, maintainable components (see the folders in ./frontend)
  - Creating high-level code folders based on functionality (ex. Alembic, frontend, server, keys)

- **Defining classes**
  - Many of my React Native components are defined as classes
  - The backend schema is defined using classes (in models.py) using SQLAlchemy

- **Mixins, polymorphism**
  - In models.py, I defined a RequestMixin that RideRequest and DriveRequest are based off of

- **Inheritance**
  - In the frontend/store folder, I define my MobX store, which heavily uses inheritance (see the DomainStore class)
Databases

- Defining normalized schemas
  - I wrote a schema with 10 tables
  - The tables satisfy BCNF and 3NF

- Using foreign keys
  - I make extensive use of foreign keys in my schema (see the user_id attribute on Driver and the start_location_id on Route, for example)
Networks

- Client-server model
  - Communication over HTTP
- Knowledge of HTTP
  - Helped to know about HTTP headers when switching to GraphQL and integrating authentication
- Webhooks
  - Useful for integrating Stream Chat API (See server/receive_webhooks.py)
- Event-driven programming
  - In using MobX, I made heavy use of RxJS, a library for observable data types in JavaScript
  - Push-based rather than pull-based data flow
CS 323 (Systems Programming)

- Creating long technical specs and executing them
  - Had to talk about business needs with teammates, translating them into technical specs and dividing up the work into chunks so it became manageable
- Reading technical specs
  - While learning about new web softwares, I had to read lots of documentation (which I got experience in while reading C documentation for the more detail-specific parts of 323 assignments)