GitHub Username: raheel

Vehicle Maintenance Reminder

Description

This app shows the user the maintenance schedule for their vehicle and allows them to be set notification. It talks to the Edmunds Vehicle APIs to get the list of vehicle by year, make, and model, and allows the user to set reminders on each of the maintenance item.

Intended User

Car owners who like to stay organized and/or have a habit of forgetting about when to take their car for maintenance.

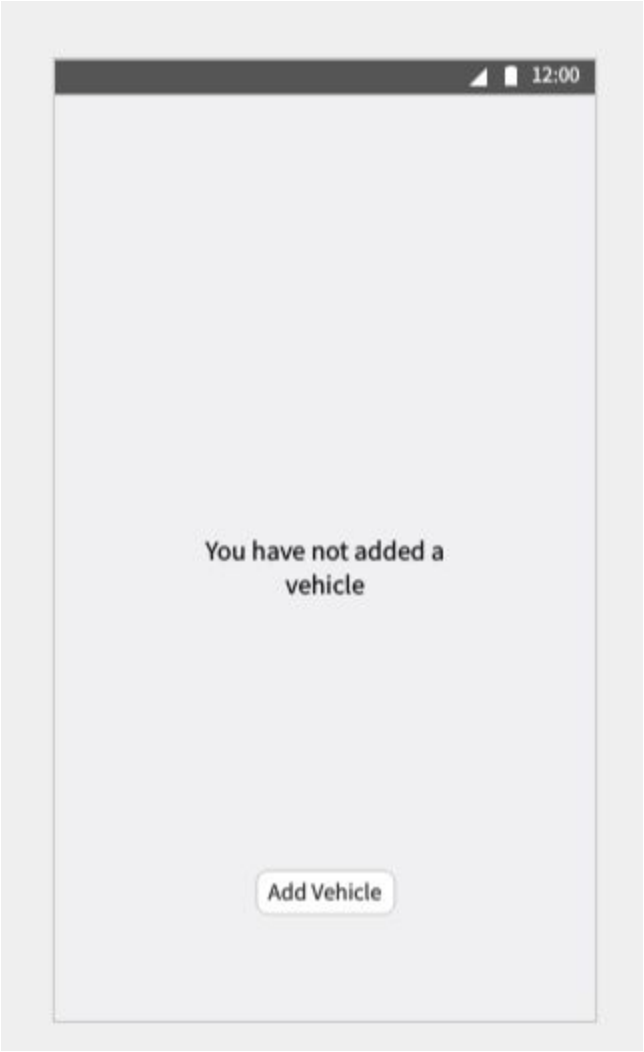
Features

- Saves Vehicle and Maintenance Schedule Information
- User is able to select which maintenance they want to set a scheduled notification for

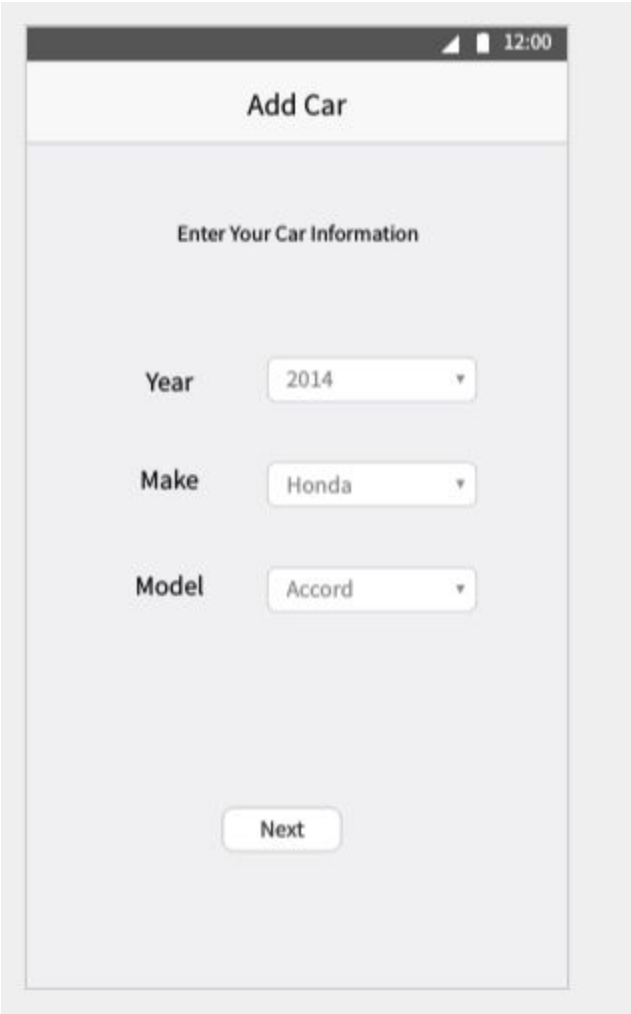
User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Photoshop or Balsamiq.

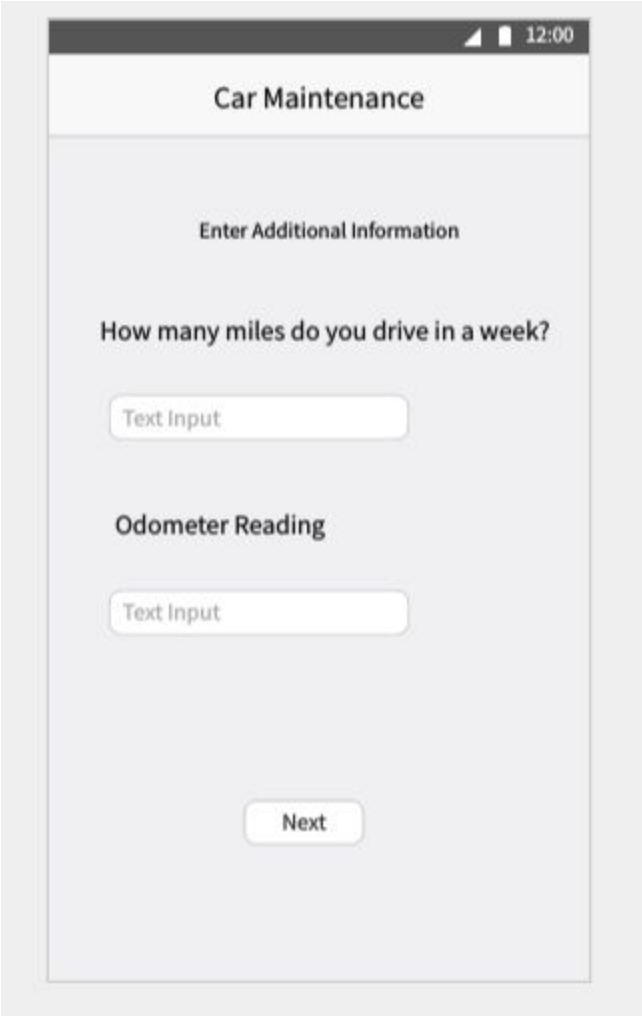
Home Screen (no vehicles)



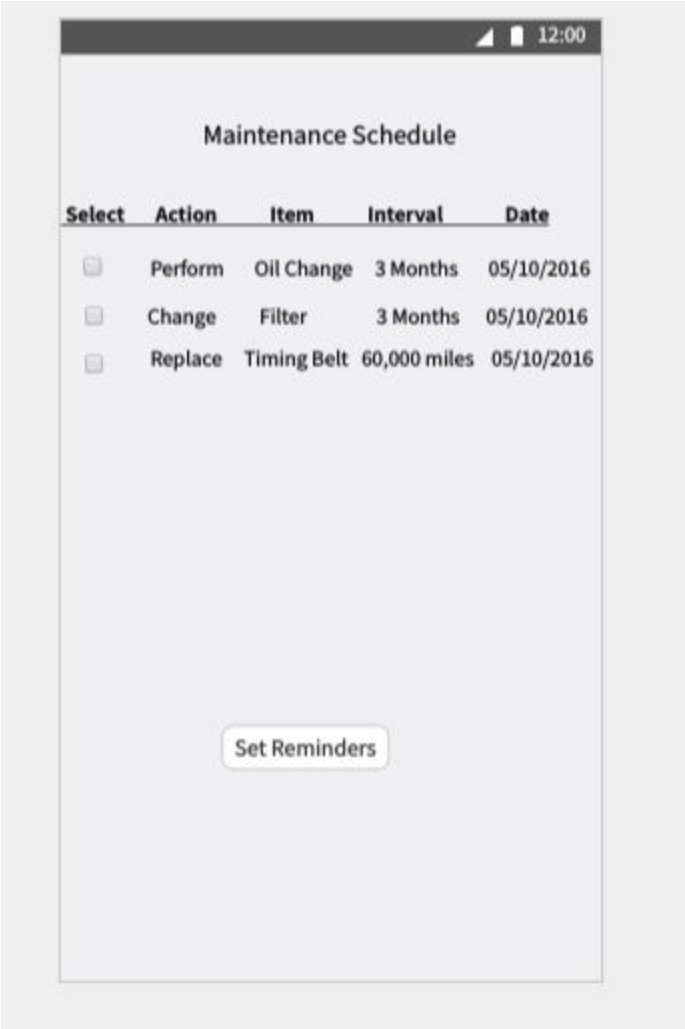
Add Vehicle Screen



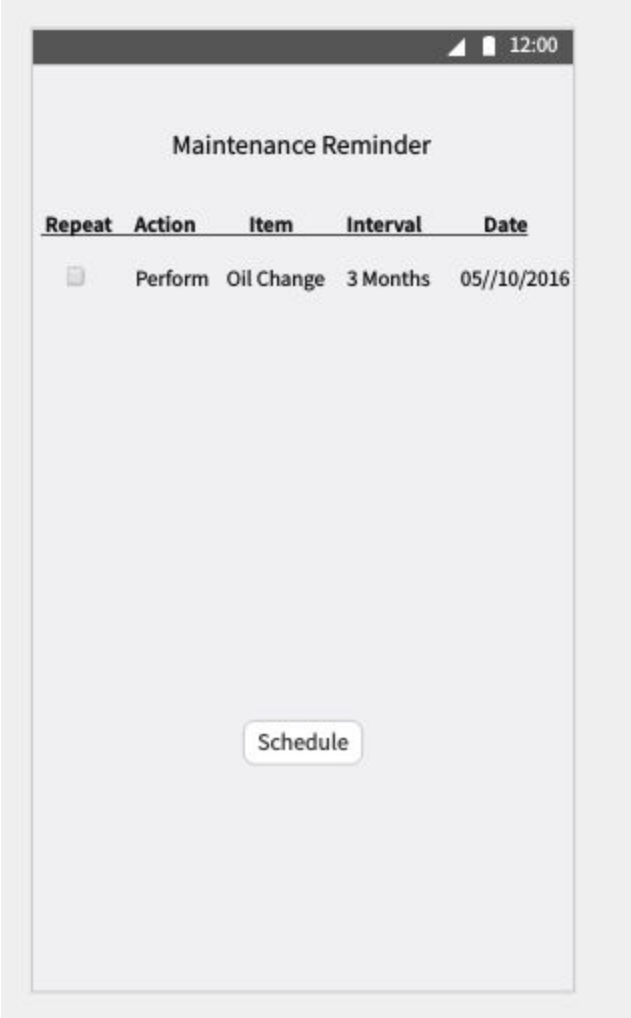
Add Additional Information Screen - This will be used to calculate on when the next maintenance is due.



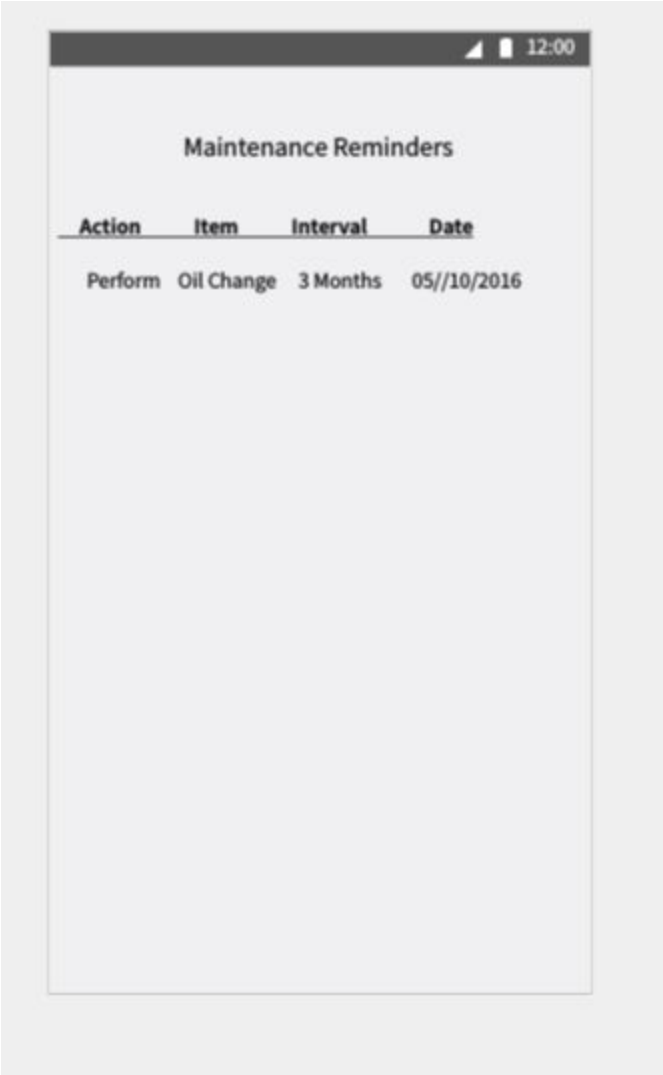
Display All Maintenance Screen



Setting Maintenance Reminders - also allows user to select if they would like to repeat the reminder



Home Page (with vehicle selected)



Key Considerations

How will your app handle data persistence?

Data will be stored in an SQLite database

Describe any corner cases in the UX.

When a user first uses the app, they will be shown a message saying that they will need to save a vehicle. There will be a button that goes to the enter your vehicle screen, once they sets his

vehicle, they will be shown their vehicle's recommended maintenance schedule, and they will be given the choice to set reminders.

If a vehicle has been set and the user starts the app, they will be shown the "My Scheduled Reminder" page.

Describe any libraries you'll be using and share your reasoning for including them.

For example, Picasso or Glide to handle the loading and caching of images.

- Picasso - Loading/caching images
- Retrofit - API calls
- RxJava - working with Observables
- Dagger - Dependency Injection
- Buterknife - View injection
- Android Job Library - Job scheduling

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

Write out the steps you will take to setup and/or configure this project. See previous implementation guides for an example.

You may want to list the subtasks. For example:

- Configure libraries
- Something else

If it helps, imagine you are describing these tasks to a friend who wants to follow along and build this app with you.

Configure libraries

- Setup all the required libraries in the app's build.gradle
- Create Dagger components, modules, and integrate it with the Application
- Create API Service for making calls to Edmund's Vehicle Maintenance API using Retrofit
- Setup project hierarchy

Task 2: Implement UI for Each Activity and Fragment

- Build UI for MainActivity. This will contain fragments for
    - Add Vehicle and mileage
    - List Recommended Vehicle Maintenance
    - Setup schedule page
    - Confirmation Screen
    - Delete Vehicle
- Build SettingsActivity

Task 3: Setup SQLite Database

- Create ContentProvider
- Setup tables
- Create models for each table

Task 4: Setup Job Scheduler Framework

- Add ability to add/delete jobs

Task 5: Create Layout

- Create layout and focus on Material Design concepts such as CoordinatorLayout, Transitions, AppBarLayout etc.

Task 6: Testing

- Create unit tests
- Manual testing