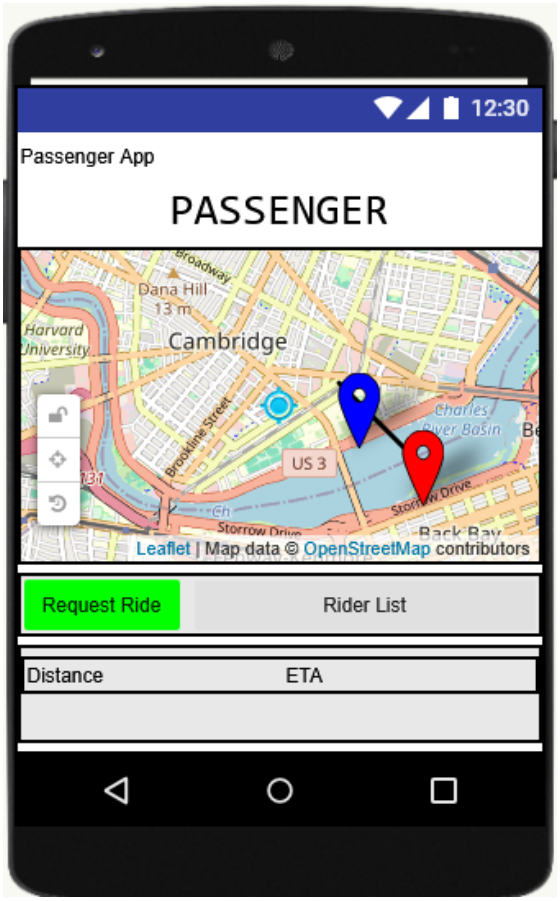

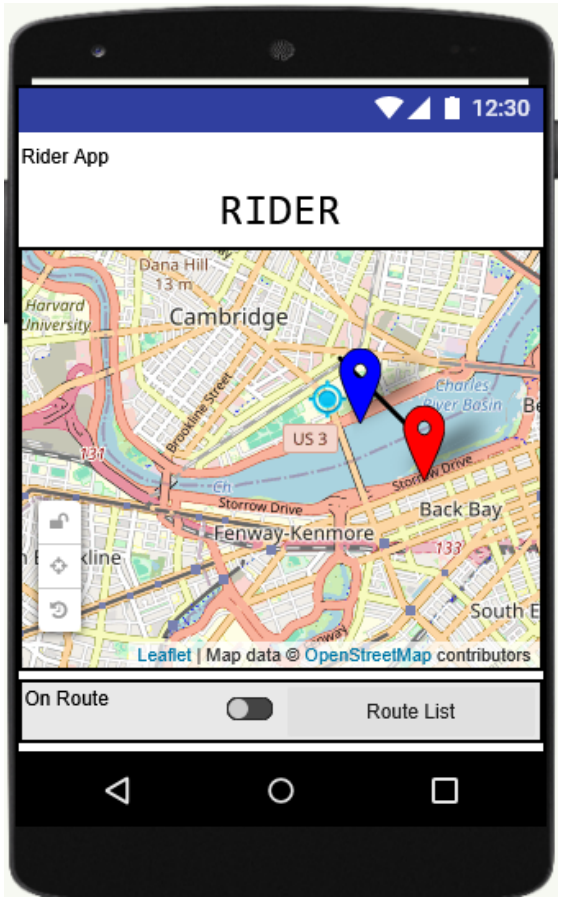

---

1. Frontend: MIT App Inventor

This application is built entirely using **MIT App Inventor**, a block-based visual programming environment. The primary components used are:

- **User Interface:** This includes a **Map Component** to display the user's and jeepney's locations, and standard UI elements like **buttons** (for login, on-route toggle), **text boxes** (for user input), and **labels** to display information.
- **Sensors:** The app relies on the **Location Sensor** to access the device's GPS and network-based location. It uses the LocationSensor.Distance block to perform a simple, straight-line calculation between two sets of coordinates to determine the ETA.
- **Built-in Functionality:** The **Notifier** component is used to generate the pop-up alerts for passengers. A **Clock** component is also used to trigger location updates at regular intervals.

- **Extensions:**
    - **FirebaseDB Extension:** This is the bridge between the app's frontend and the backend database.
    - **SignInWithEmailandPassword Extension:** This handles the user authentication process.

---

2. Backend: Firebase

The backend is entirely managed by **Google's Firebase** platform, eliminating the need for a separate server.

- **Firebase Realtime Database:** This is the primary data storage. It's a NoSQL, JSON-based database used to store and synchronize real-time data, specifically the location coordinates of active users and the "on-route" status of drivers.
- **Firebase Authentication:** This service manages all user sign-in methods, including the creation and management of persistent email/password accounts and temporary, non-persistent anonymous accounts.

---

3. External Services & Physical Hardware

- **OpenRouteService API:** While the app's ETA is a simple calculation, this API is likely integrated for more complex navigation or to display accurate road networks on the map.
- **Android Device:** The app is designed to be compiled into an **APK file** and run on any Android device with a working GPS unit and an internet connection.
- **BLE Beacons:** This is an intentional and notable component for a future version of the app. The plan is to use **BLE beacons for assisted GPS tracking**, which would improve location accuracy, especially in areas with poor GPS signals.

1. Core Functionality & Purpose

- **Alert System:** The user can dismiss the 5-minute ETA alert by closing the notification. A known bug for the next developer to fix is the re-triggering of the alert if the ETA increases beyond the 5-minute threshold and then decreases again.
- **Map with Markers:** A passenger can select a specific jeepney marker to track. If no selection is made, the app will **default to tracking the closest jeepney**. The jeepney's driver is not made aware of this selection, as their app only serves to display their own location.
- **ETA/Distance Calculation:** The ETA is calculated using a **simple distance formula** based on the device's GPS coordinates. This relies on a built-in MIT App Inventor function (specifically, the LocationSensor.Distance block). This method does not account for traffic or deviations from the route, which are **known bugs** for the next developer to address.

2. User & Data

- **User Journey:**
    - **Passenger:** The passenger's journey is streamlined. They log in, view the map, and can either select a specific jeepney marker or allow the app to default to the closest one. The app's purpose is to simply provide a notification when the jeepney is within a 5-minute ETA. The trip **does not have an "end state"** within the app, and the driver is never notified of the passenger's boarding.
    - **Driver:** The driver's app is a minimal viable product (MVP) focused on one function: **broadcasting their location**. They toggle an "on-route" button to make their marker visible to passengers. The driver has no way to see passenger locations or requests.
- **Data Structure & Handling:**
    - **Data Storage:** The app uses Firebase Realtime Database for storing real-time coordinates. These coordinates are only updated when the device's GPS location changes. When the user closes the app, their location data is no longer updated.
    - **Data Persistence:** Data for anonymous users is **not stored** and is lost upon app closure. For users who sign in with an email and password, their data can be retrieved if they re-open the app, as long as they have not explicitly logged out.
    - **Local-Only Data:** The app **does not store trip or request data** on the Firebase database. The connection between a passenger's "request" (their selection of a jeepney marker) and the jeepney is handled entirely within the app's local memory. The driver app is not involved in this process. This means that if a passenger's app crashes or is closed, the link to the selected jeepney is lost.
    - **Offline State:** A driver indicates they are "offline" or "no longer on a route" by toggling the "on-route" button. This removes their marker from the map.
- **Authentication:** Anonymous user data, including location history and any "trip" information, is completely lost when the app is closed. This provides a temporary, no-commitment option for new passengers to test the app. Drivers are required to use email/password authentication as they are considered constant users.

Functionality List

For Passengers

1. **Map Display:** Shows the real-time location of both the passenger (self) and nearby jeepneys on a map.
2. **Jeepney Selection:** Allows the passenger to select a specific jeepney marker on the map to track.
3. **Automatic Tracking:** If the passenger does not select a specific jeepney, the app will automatically default to tracking the closest jeepney.
4. **Proximity Alert:** Triggers a notification alert when the ETA of the selected jeepney is within a fixed 5-minute window.
5. **Alert Dismissal:** Allows the user to close the alert notification box.
6. **Anonymous Access:** Provides the option for new passengers to use the app anonymously, without creating an account.
7. **Account Management:** Enables passengers to create an account with an email and password for data persistence.

For Drivers

1. **Location Broadcasting:** Displays the driver's real-time location on the map for all passengers to see.
2. **Status Toggle:** A simple button allows the driver to toggle their "on-route" status, which determines whether their location is visible to passengers.

Cross-Cutting Functionality

1. **Real-time Data:** Uses Firebase Realtime Database to store and update user and jeepney locations.
2. **GPS Integration:** Utilizes the device's GPS to calculate distances and update locations.
3. **Simple ETA Calculation:** Calculates the ETA based on a simple distance formula, without accounting for traffic or real-world routing.
4. **Authentication:** Offers two authentication methods:
    - **Anonymous:** For temporary passenger use (data is not saved).
    - **Email/Password:** For both drivers and persistent passenger users.
5. **Offline State:** Allows both users to go "offline" by closing the app. For drivers, this can also be done by toggling the "on-route" button.

Components List

1. Frontend (MIT App Inventor)

- **User Interface Components:**
    - **Map Component:** Displays the map and markers for users and jeepneys. This likely uses a built-in Map component.
    - **Location Sensor:** Retrieves the device's real-time GPS coordinates.
    - **User Interface Elements:** Buttons (On-Route toggle), text boxes (for email/password), and labels.
- **Built-in Functions & Extensions:**
    - **Location Sensor:** Provides the LocationSensor.Distance function for basic distance and ETA calculations.
    - **Notifier:** Used to display pop-up alerts for passengers (e.g., the 5-minute ETA notification).
    - **Navigation:** Likely a built-in or custom extension to handle the routing aspect, though we've established it's a simple, non-API-driven calculation.
    - **FirebaseDB Extension:** The core component for interacting with the Firebase Realtime Database.
    - **SignInWithEmailandPassword Extension:** Handles user authentication with Firebase Authentication.
- **Other Components:**
    - **Clock:** Used for timing and potentially for refreshing location data at regular intervals.

2. Backend (Firebase)

- **Firebase Realtime Database:**
    - Main database for storing real-time, unstructured data.
    - Stores user locations (for both passengers and drivers).
    - Manages the "on-route" status of jeepneys.
- **Firebase Authentication:**
    - Manages user accounts.
    - Handles sign-in with email and password.
    - Manages anonymous user sessions.

3. External Services

- **OpenRouteService API:**
    - Used for navigation purposes, specifically for finding the route between two points. While the ETA is a simple calculation, the actual path on the map likely uses this API.
- **BLE Beacons:**
    - A planned component for future development (Assisted GPS tracking).

4. Physical Components

- **Android Device:** The platform on which the compiled APK will run.
- **GPS Unit:** The device's built-in GPS functionality is a critical component for the app's core functionality.

Using GitHub for MIT App Inventor Projects

Since MIT App Inventor is a block-based visual tool, it does not have a native integration with Git. However, you can still leverage GitHub for version control by manually exporting and importing your project files (.aia files).

This workflow is useful for:

- **Version History:** Saving milestones of your project so you can easily revert to an older version.
- **Backup:** Keeping a secure, remote copy of your project.
- **Collaboration:** Sharing your project with others to allow them to download and work on the same file.

Step 1: Export Your MIT App Inventor Project

To get your project file, you need to export it from the MIT App Inventor web interface.

1. Open the MIT App Inventor website and go to your "My Projects" page.
2. Select the project you want to save.
3. Go to the **Projects** menu in the top toolbar.
4. Select **Export selected project (.aia) to my computer**.

This will download a single .aia file (e.g., Jeepney_Finder.aia) to your computer. This file is a compressed archive that contains all of your project's design and block code.

Step 2: Create a GitHub Repository

If you don't already have one, create a new repository on GitHub to store your project files.

1. Go to github.com and log in to your account.
2. Click the **+** icon in the top-right corner and select **New repository**.
3. Give your repository a descriptive name (e.g., Jeepney-Finder-App).
4. Add a brief description and set the visibility to public or private.
5. Click **Create repository**.

Step 3: Upload the .aia File to GitHub

Now you can upload the .aia file you just exported to your new repository.

1. On your new GitHub repository page, click the **Add file** button.
2. Select **Upload files**.
3. Drag and drop your .aia file into the upload box.

4. Add a clear **commit message** that describes the changes. For the first upload, a good message would be "Initial commit for Jeepney Finder App." For future updates, you might write "Added Firebase integration" or "Fixed ETA bug."
5. Click **Commit changes**.

Step 4: Import the Project Back into MIT App Inventor

When you want to work on a new version of your project or if someone has sent you an updated .aia file, you can easily import it.

1. On the MIT App Inventor website, go to "My Projects."
2. Go to the **Projects** menu and select **Import project (.aia) from my computer**.
3. Choose the .aia file from your computer that you want to import.

Important Note: When you import a project, it will appear as a new project in your list. If you want to continue working on the same project without creating a new copy, you should export it, make your changes, and then re-import it, overwriting the old version.