**Activity: Ride Details**

The Ride Details activity displays all the information about a ride. This displays the origin and destination campuses, the addresses, departure time, a route map, the driver’s profile picture, the passengers’ pictures, the vehicle used, the number of remaining seats, and it has a button that changes depending on the user.

It can be accessed, either by clicking a marker’s InfoWindow within the Finding a Ride Activty, or clicking a Ridelist item in the Main/Ridelist Activity.

Coding wise, the most difficult part of this application is displaying the routing information on the Maps Fragment. The tutorial I followed can be found here: <https://www.youtube.com/watch?v=VNDfD6YnJE8> She has several tutorials on how to integrate Google Maps in an Android project.

**Manifest Details**

RideDetails.java implements this activity in the Android Manifest. The screenOrientation property is restricted to portrait mode, and the activity inherits the app’s theme.

Since this activity uses Google Maps, it also makes use of the <meta-data> tag referencing the Google Maps API Key stored in google\_maps\_key.xml. The <meta-data> is located over the .FindRide’s <activity> entry.

The intent-filter provides standard access when referenced by name.

**Layout Files**

Activity\_ride\_details.xml

* The main layout file for the activity. The two major layout groups is the panel for storing the activity’s button, and a ScollView containing the rest of the fields. Both are children to a ConstraintLayout.
* The Three View of note here are the rideMapView Fragment, the passengerRecyler, and the rideDetailsButton.
* rideMapView (Fragment)
  + Displays a Mini map that displays markers between two campuses, and a route between them. This view is set up in the setMapDirections() method in the code behind, by accessing Google’s API for directions.
* passengerRecycler(RecyclerView)
  + Probably the view that makes the least sense in the entire app. This View displays multiple item\_passenger layouts in a horizontally scrolling bar, a feature that isn’t supposed by Android Studio’s ListView by default. Although it looks like a ListView in the Design window, it will appear very differently when the app is running. The code for handling the RecylerView is in an internal class in the code-behind.
* rideDetailsButton(Button)
  + A button that is set in the code behind. Its functionality and text are set depending on who is accessing the RideDetails activity.

Item\_passenger.xml

* A single passenger layout that is displayed in the Activity’s RecyclerView. It contains a single ImageView that holds the passenger’s profile picture.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Class File**

* RideDetails.java
  + Implements the RideDetails Activity, and is almost complex as the CreateRide.java file. This is in part due to drawing a route on the Maps Activity, and to get the Passengers to appear on a horizontal, scrolling bar. It was terribly tricky to implement both, and I’m not sure if there is a way to make the code more efficient…
  + Variables
    - -map: GoogleMap – Displays Google Maps in a fragment
    - -queue:RequestQueue – Sends JSON requests to the server
    - -prefs: SharedPreferences – Gets the user’s access token and userId from the phone
    - -originCityTextView: TextView
    - -originDetailTextView: TextView – Display’s the origin’s address
    - -originTimeTextView:TextView – Displays the departure time.
    - -meetingLocationTextView: TextView
    - -destinationCityTextView: TextView
    - -destinationDetailTextView: TextView – Displays the destination’s address
    - -vehicleNameTextView: TextView
    - -vehicleYearTextView: TextView
    - -driverImageView: ImageView
    - -driverId: String – Used to request the driver’s profile picture, and to determine if the current user is this trip’s driver.
    - -tripId: int – The id used to pull of the trip’s information
    - -seatNumberTextView: TextView
    - -rideDetailsButton: Button – The only button of the activity, that changes function depending on the user’s current state.
    - -destLat, destLng: String – Sent as paramaters to the LocationTrackingService when the trip begins.
    - rideComplete: Boolean – determines if the rideDetailsButton should be enabled.
  + Methods and Internal Classes
    - \*onCreate(savedInstanceState): void
      * Displays the activity\_ride\_details layout, and initializes “queue” and prefs”
      * “tripId” is set by an IntExtra sent by the previous activity’s Intent.
      * Creates a back button in the toolbar
      * Displays Google Maps in “mapFragment”
      * Sends method calls to set up the Views, and to fetch the current ride’s details.
    - +onOptionItemSelected(item): Boolean
      * Makes the back button in the toolbar return to the previous activity when tapped.
    - +onMapReady(googleMap): void
      * Initializes the “map” variable to the googleMap, changes the map’s display type to show the terrain, and disables scrolling on the map fragment.
    - -setupViews():void
      * Locates and initializes the TextView’s, the driver’s ImageView, and the rideDetailsButton.
    - -getRideDetails(): void
      * Creates a url String to get the current trip based on the tripId
      * Prepares a JSON Object GET request to pull a ride’s’ information from the server, and sets the textViews to their appropriate values.
      * The JSON Object returned as “response” contains sub-objects and arrays with their own properties, and are stored in objects created for the response. Some of them are passed as parameters in other methods.
      * There are three method calls made during the request.
        + The first is made to setMapDirections, which draws the route on the map. First, the request creates a new HashMap to store the coordinates for the origin and destination, and passes it as a parameter.
        + The next call is showPassengers, which displays all of the passengers on the trip. A JSON Array called “passengers” is created, and is passed as a parameter.
        + The last call is setUpButton, which takes the server’s response, the driver object, and the passengers object and uses this to determine how the rideDetailsButton will function, in response to the current user.
      * Lastly, the request is added to the queue.
    - -setMapDirections(mapDirections): void
      * Takes in a HashMap<String, Sting> from getRideDetails to show the route between the origin and destinations campuses on the map. It’s highly recommended that you watch the tutorial video at the start of this document, if you have not done so already.
      * It creates two LatLng objects using the origin and destination coordinates, and places them on the map as Markers.
      * Next, the URL for the Google Maps Directions API is created using the origin and destination coordinates, followed by the Google Maps key.
      * Next, a JSON Object is returned containing the directions, and a JSON Array called “steps” is stored into a JSON Array variable. A String array called paths is set to the same length as the steps array, and is looped through to get “point” on a polyline. Polylines are the series of lines drawn to show the route.
      * Once the paths array is initialized, another loop adds the polylines onto the map, and the path is drawn.
      * The Markers are then added on both ends of the route, and the camera is set to enclose the path.
      * My explanation is terrible, but a simplified breakdown of the route is so:
        + Two points make up a polyline,
        + A polyline makes up a step (such as, travel 20 miles, then turn right)
        + Multiple steps make up a leg of a journey (For example, if you had to make several stops to gas stations, the journey from the school to the station is a leg, then the journey from the station to the school is another leg)
        + And all the legs together make the entire route (even if there is only one leg of the journey)
    - -showPassengers(passArray): void
      * Displays all of the passengers horizontally in the RecyclerView
      * It receives a JSONArray of passengers from the getRideDetails() method.
      * A loop moves through the array, uses a HashMap to store each passenger’s id and profile picture. The HashMap is stored in an ArrayList.
      * Afterward, the Activity finds the RecyclerView in the layout, sets its orientation to horizontal, and then displays a series of passenger items using a RecylcerView Adapter.
    - -setUpButton(r, d, p): void
      * Takes the Object response, driver, and passengers array from GetRideDetails to determine how the rideDetailsButton can be used.
      * The userId is retrieved from the phone, and Boolean flags for if the user is a driver or passenger is set to false. If the driverId matchs the userId, then this user is a driver. If the userId matches a passenger’s id, this user is a passenger.
      * If the user is a driver, the button can allow the driver to start the ride (starting the location service), cancel the ride (sending a DELETE request), or they won’t be able to click it at all, if the ride has been completed.
      * If the user is a passenger, the button only allows them to leave the ride.
      * If the user is neither a driver nor passenger, if there is enough seats available, the button allows the passenger to join the trip. If the seats are full, the button is disabled. If the user is already part of an active ride (the location service is active) then they will be considered as part of an active ride, and won’t be able to join the trip at all.
    - -joinLeaveRide(method): void
      * This sends a request to post the user as part of join or leave requests. Since data returned from the server isn’t handled on the Activity, these were placed in the same method.
      * The method parameter is set by the button’s onClick event. If the method is set to POST, it will create a HashMap of the passenger’s data and send it to the server to add the passenger to that trip. If the method is set to DELETE, then a delete request is sent to remove the passenger from the trip.
      * The service for location tracking is either started or stopped.
    - -cancelRide(): void
      * This method can only be called by the driver. It sends a JSON Object DELETE request to the server, stops the location service, and returns the user to the previous screen.
    - -startLocationService(): void
      * Prepares an Intent for the TripLocationService.java class, passing in the destination, the destination’s coordinates, and the trip and driver id.
    - -PassengerAdapter
      * An internal class to populate the RecyclerView with passengers’ profile pictures. Each item\_passenger layout is also given a click event that starts the Alternate Profile activity, displaying a user’s contact information and average rating.
    - -MyViewHolder
      * As of writing this, I cannot remember why this class is here, only that it was necessary for the RecyclerView.

**Other Files**

* Google\_maps\_api.xml
  + Holds the API key which allows the app to send request to Google Maps.