Ryde

Fall 2018 Senior Project

Wen Cao, Tyler Carberry, Benny Chen, Nicholas La Sala,
Benny Liang, Michael Matthews, Tapan Soni, and John Stranahan

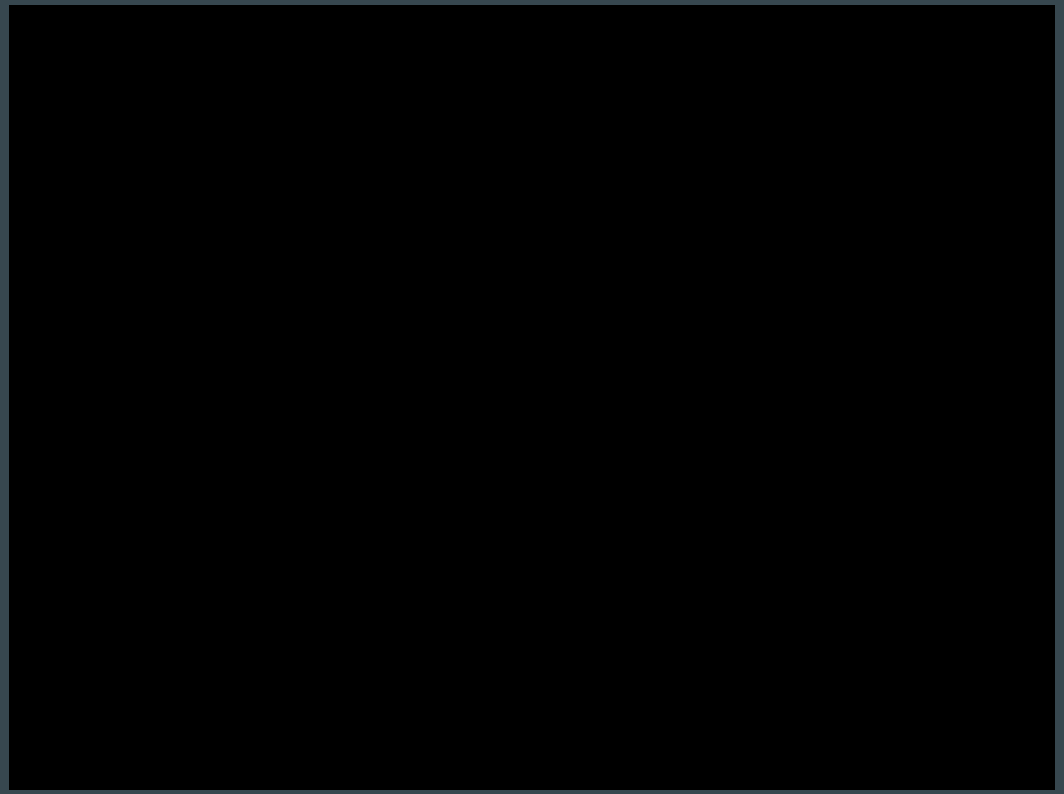
Ryde

The purpose of our app is to help Rowan students find carpoolers who live near them and share a similar class schedule

Pros of Ryde:

- **More parking spaces**
- Cuts down on traffic
- Saves commuters money

Demonstration Video



App Architecture

Flutter → Spring running on AWS → MySQL database

Finding matches

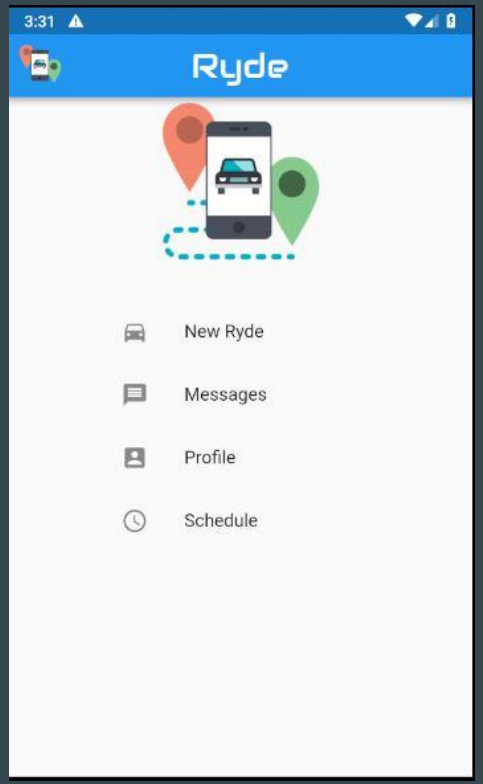
Users are matched if they live within a certain range of each other and have similar schedules.

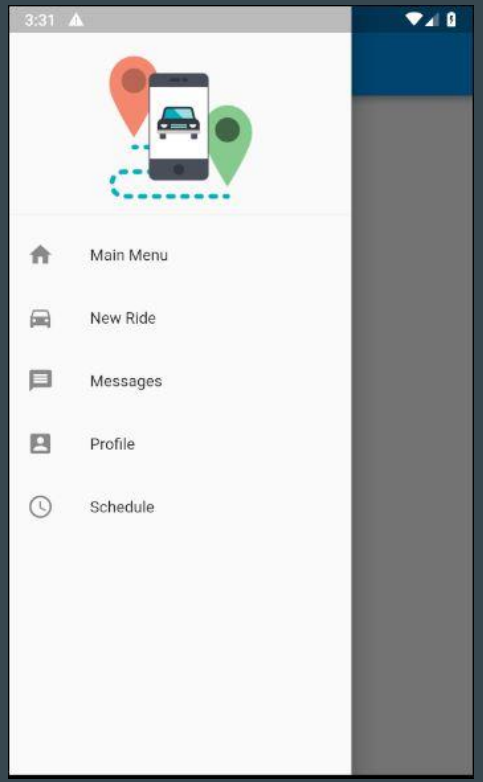
- Longitude and latitude of a user's address is stored in the database
- A SQL query is used to retrieve all other users that live within a certain range of the user
- The schedules of those users are then compared to find matches

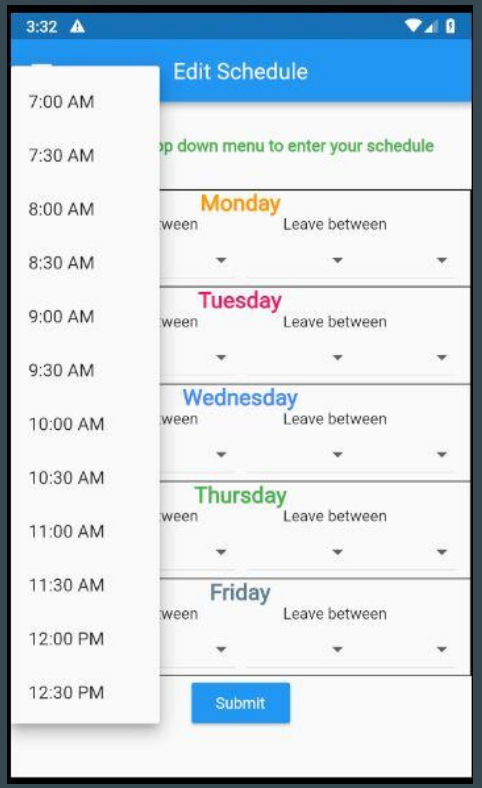
Flutter Framework -- Front End

1) Used Flutter for the front end
    a) Developed by Google for cross-platform application development using a single code base
    b) Full of rich features and widgets for developing apps
    c) Easy to install

Screenshots of Ryde

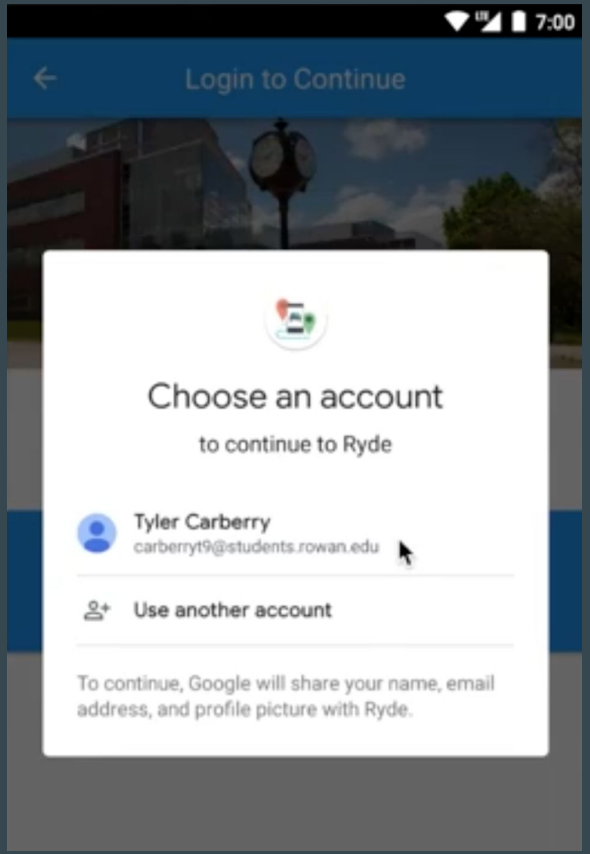






Google OAuth

- Use your Rowan credentials to log in
- Links your Ryde account with your Rowan account
- Validates that you are a Rowan students
- We are able to pull your email address and profile picture
- Improved security - don't need to handle passwords



Google Geocoding API

- Provides latitude and longitude given an address
  - Used to find the distance between users
- Also formats the user's address
- Ex. If you mistype your address as
  "201 Mullica Hill Street" Google will autocorrect it


```
],
"formatted_address" : "1600 Amphitheatre Parkway, Mountain View, CA 94043, USA",
"geometry" : {
   "location" : {
      "lat" : 37.4224764,
      "lng" : -122.0842499
   },
   "location_type" : "ROOFTOP",
   "viewport" : {
      "northeast" : {
         "lat" : 37.4238253802915,
         "lng" : -122.0829009197085
      },
      "southwest" : {
         "lat" : 37.4211274197085,
         "lng" : -122.0855988802915
      }
   }
},
```

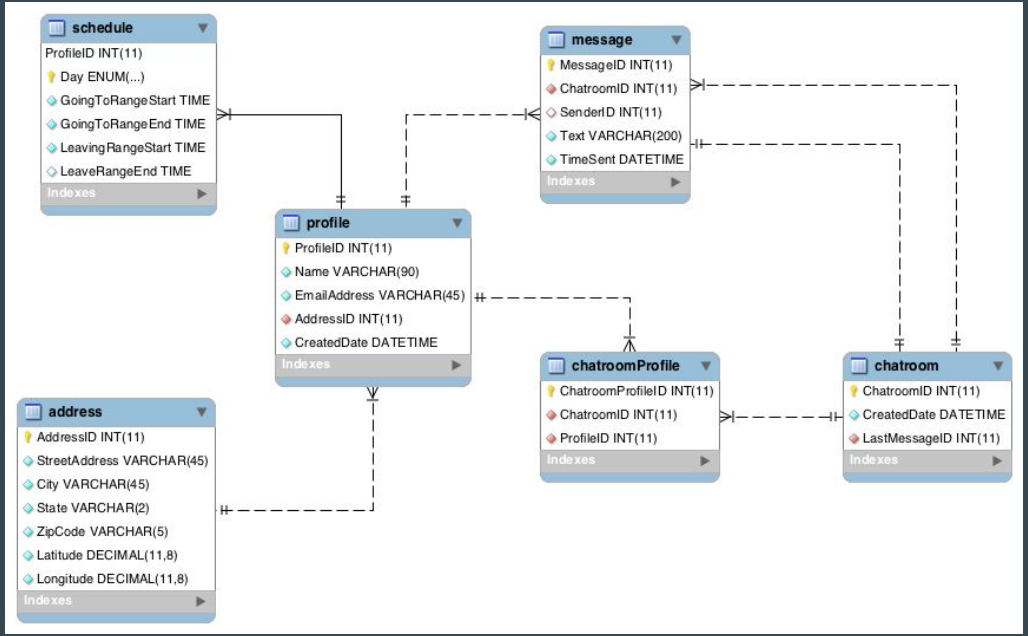
Back End

Our back end was coded in Java.

- Spring Boot: A Java framework to make web and server applications
- RESTful API to handle interactions with the front end
- Endpoints accepts and returns JSON objects for GET and POST requests

Database

- Initially wanted to use Cassandra Database for scalability (NoSQL)
- Low expected user base; scalability isn't an issue
- MySQL for simplicity



Hibernate

Hibernate allows us to work with Java classes that are mapped to database tables

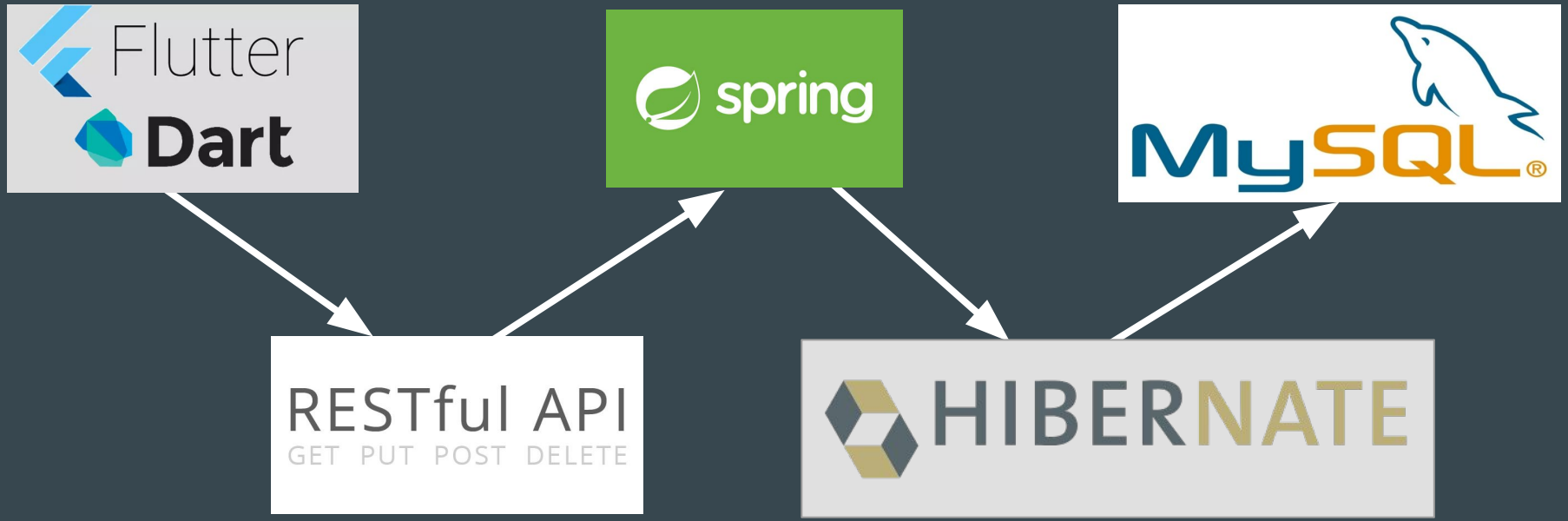
- We generally don't have to write our own queries - Hibernate provides a nice framework that auto generates simple queries
- To update a row in the profile table, we just need to modify the corresponding Profile object and push the changes to the database


```
/*  Change name of a profile */
// JSON inputs are the profile id and the new name
int profileID = Integer.parseInt(map.get("id"));
Profile profile = profileRepository.findById(profileID).get();
profile.setName(name);
profileRepository.save(profile);    // push changes to database
```

RESTful API

- **Re**presentational **S**tate **T**ransfer
- Architecture style for designing networked applications
- Relies on stateless, client-server protocol, almost always HTTP
- Treats server objects as resources that can be created or destroyed
- Uses JSON file format

App Structure



HTTP REQUEST

http://amazonaws.com:8080/rides/profile/4

JSON RESPONSE


```
{
  "id": 4,
  "name": "Ali Houshmand",
  "address": {
    "Id": 4,
    "streetAddress": "101 Hazardous Circle",
    "city": "Flavor Town",
    "zipCode": "08096",
    "state": "NJ",
    "latitude": 39.8233921,
    "longitude": -75.1192713
  },
  "createdDate": "11/17/18 21:21:29",
  "schedules": [
    {
      "id": 22,
      "day": "THURSDAY",
      "goingToStart": "06:00:00",
      "goingToEnd": "12:00:00",
      "leavingStart": "17:00:00",
      "leavingEnd": "19:00:00"
    }
  ],
  "chatrooms": [ ],
  "distance": 0,
  "distanceRounded": 0,
  "schedulesString": null,
  "email": "hotsauce@.rowan.edu"
}
```

RESTful API Get

- In Flutter making rest requests from the server is done using future builder objects and returning the Json from a url
    - This data can be accessed as a snapshot using future builder inside a container
    - An object used to represent a mapping of data is used for the the future builder to create a list

```
Container(
    child: Center(
        child: FutureBuilder<Post>(
            future: getPost(),
            builder: (context, snapshot) {
              if (snapshot.hasData)
                return Text('${snapshot.data.name}');
              else
                return CircularProgressIndicator();
            })), // FutureBuilder, Center, Container
```

```
Future<Post> getPost() async {
  int userid = await getId();
  print(userid);
  print(userid.toString());
  String postUrl = BASE_URL + '/rides/profile/$userid';
  print(postUrl);

  final response = await http.get(postUrl);
  return postFromJson(response.body);
}
```

RESTful API Post

- Posting to the server also requires a future method
  - This method will take the data being posted in the form of an object
  - The object should act as map for the different fields being posted
  - The object is json encoded when the response is created.

```
Address newAddress = Address(
    id: userId,
    streetAddress: streetNameEdit,
    city: cityNameFinal,
    zipCode: zipCodeEdit,
    state:
        stateEdit); // creating a new Post object to send it to API // Address

createAddress(newAddress).then((response) {
  if (response.statusCode > 200)
    print(response.body);
  else
    print(response.statusCode);
}).catchError((error) {
  print('error : $error');
});
```

```
Future<http.Response> createPost(Post post) async {
  String updateUrl = BASE_URL + '/rides/address/update';
  final response = await http.post('$updateUrl',
      headers: {
        HttpHeaders.contentTypeHeader: 'application/json',
        HttpHeaders.authorizationHeader: ''
      },
      body: postToJson(post));
  return response;
}
```

Challenges

1. Flutter
    a. In beta most of the time we were making this app - so not everything worked!
    b. No one in our group had ever developed apps in Flutter so it was a brand new experience for everyone
2. Running the server on AWS
    a. Needed to update AWS every time the server changed
3. Android Studio
    a. Needs a powerful computer to run

Improvements for the future

If we were to keep working on this app, what would we add?

1. Rating system
2. Multiple person carpool option
3. Group chats
4. Matches based on major
5. Support for other universities

Ryde