UW Carpool Application

11/21/2018

Presented by:

Jiatong He, Shuting Lian, Zhilun Chang, Zizheng Jiang

UNIVERSITY OF WATERLOO

Outline

- Introduction
- Architecture Style
- Functional Properties
- Non Functional Properties
- Design Patterns
- Technical Challenges & Future Improvement

Introduction

- **UW Carpool** Application – Android app

  - Organizes carpooling information for both **drivers** and **passengers.**

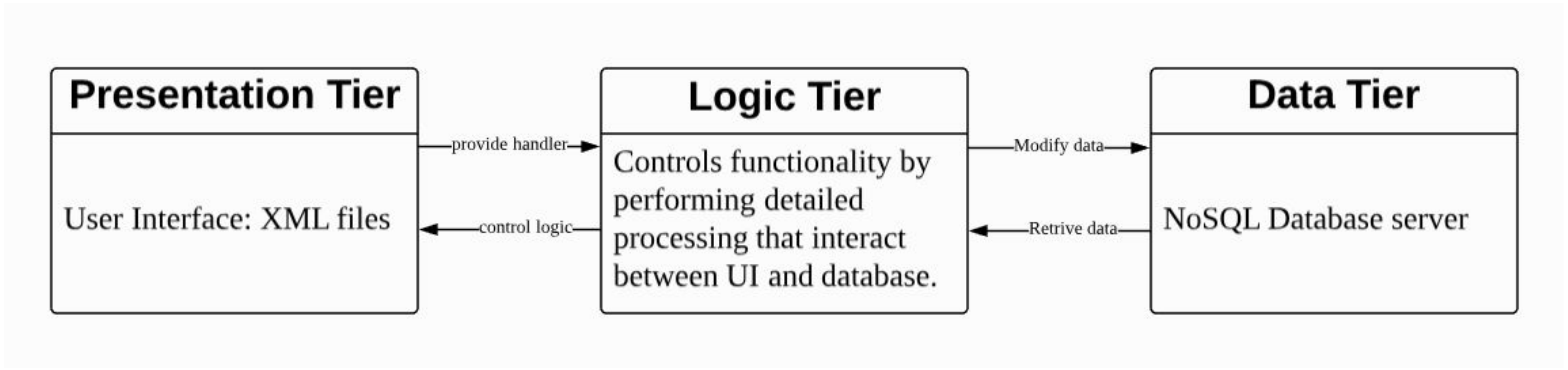
    - **Drivers** provide travel information, which includes *Departure City*, *Arrival City*, *Departure Address*, *Arrival Address*, *Departure Date & Time*, *Phone Number*, *Vacancies* and *Price*.

    - **Passengers** enter *Departure City*, *Arrival City,* and *Date* and then search for available information.

UNIVERSITY OF WATERLOO

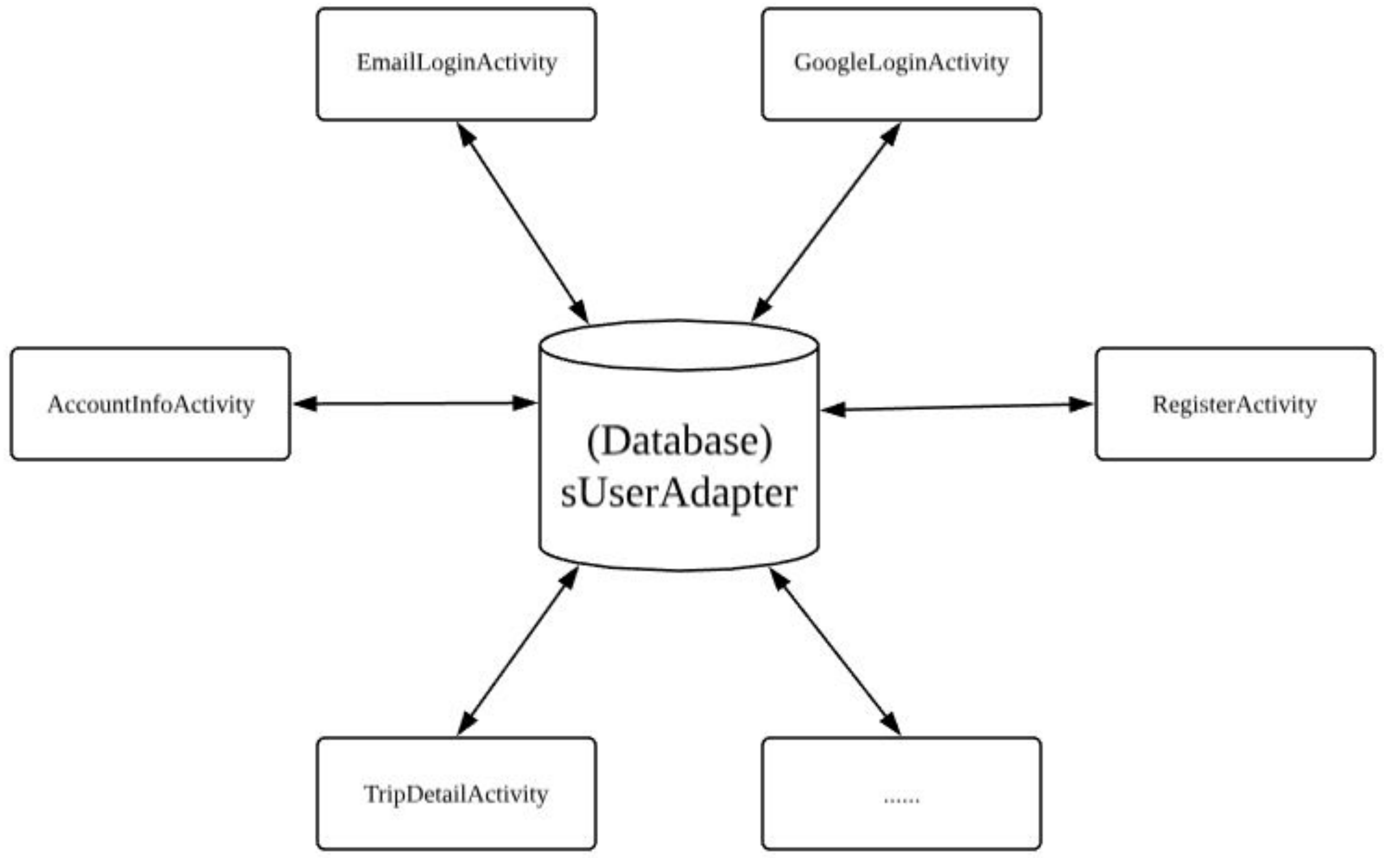
Architecture - Client-Server Style

- **Presentation Tier**: User Interface
- **Logic Tier**: Logic between UI and NoSQL
- **Data Tier**: NoSQL Database


Presentation Tier
User Interface: XML files
provide handler
control logic
Logic Tier
Controls functionality by performing detailed processing that interact between UI and database.
Modify data
Retrive data
Data Tier
NoSQL Database server

Architecture - Blackboard Style

Various activities can access data from **NoSQL Database** or **sUserAdapter** which contains data retrieved from database.



UNIVERSITY OF WATERLOO

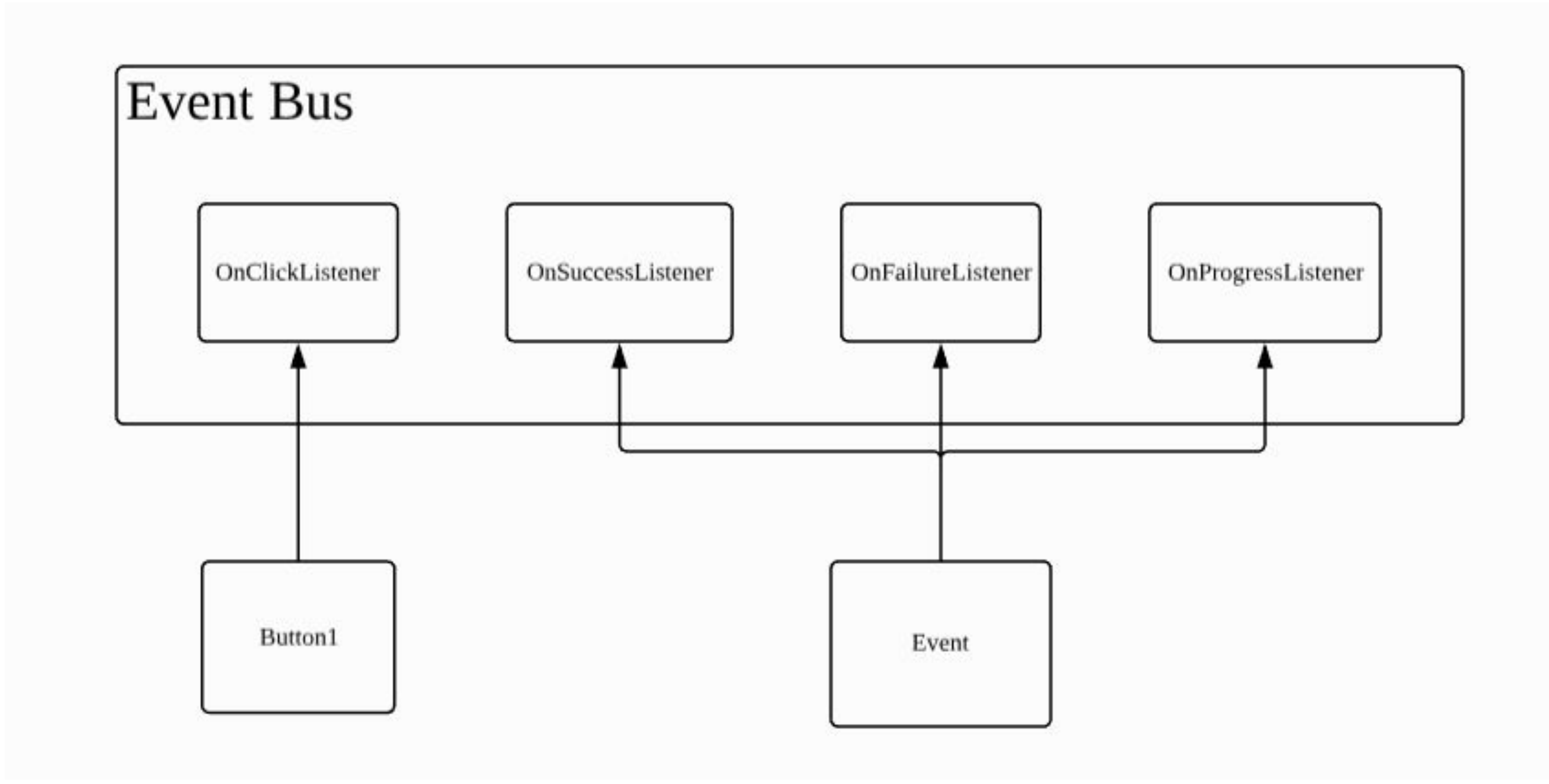
Architecture - Event-based Style

An **event-based architecture** consists of event ***creators*** and event ***consumers***.

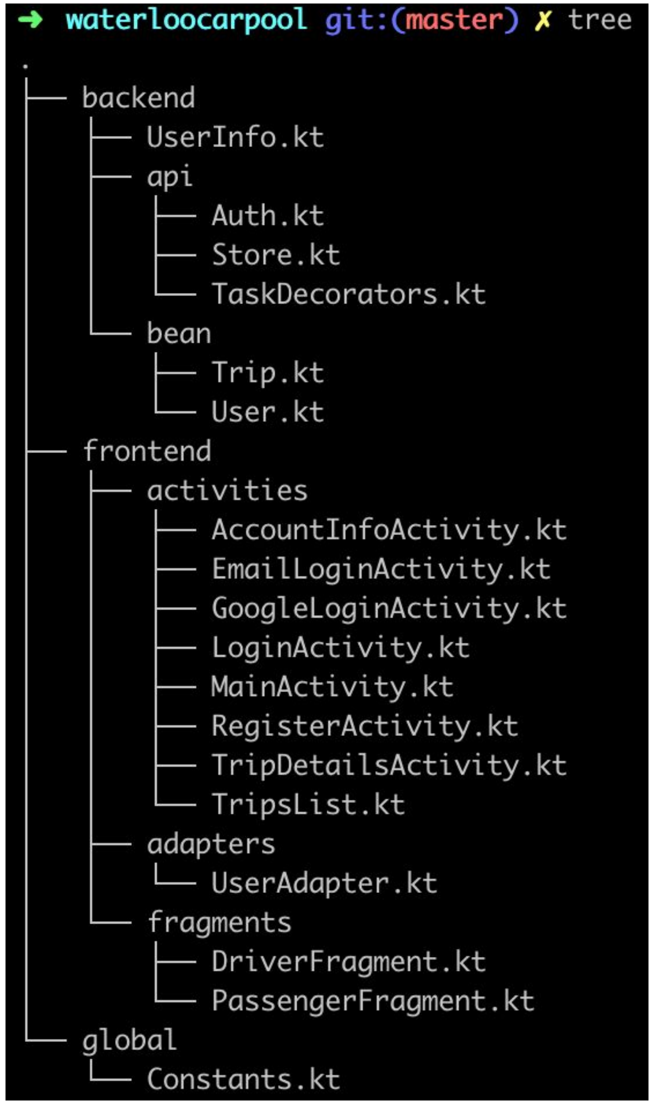
Example:

Creators: Buttons

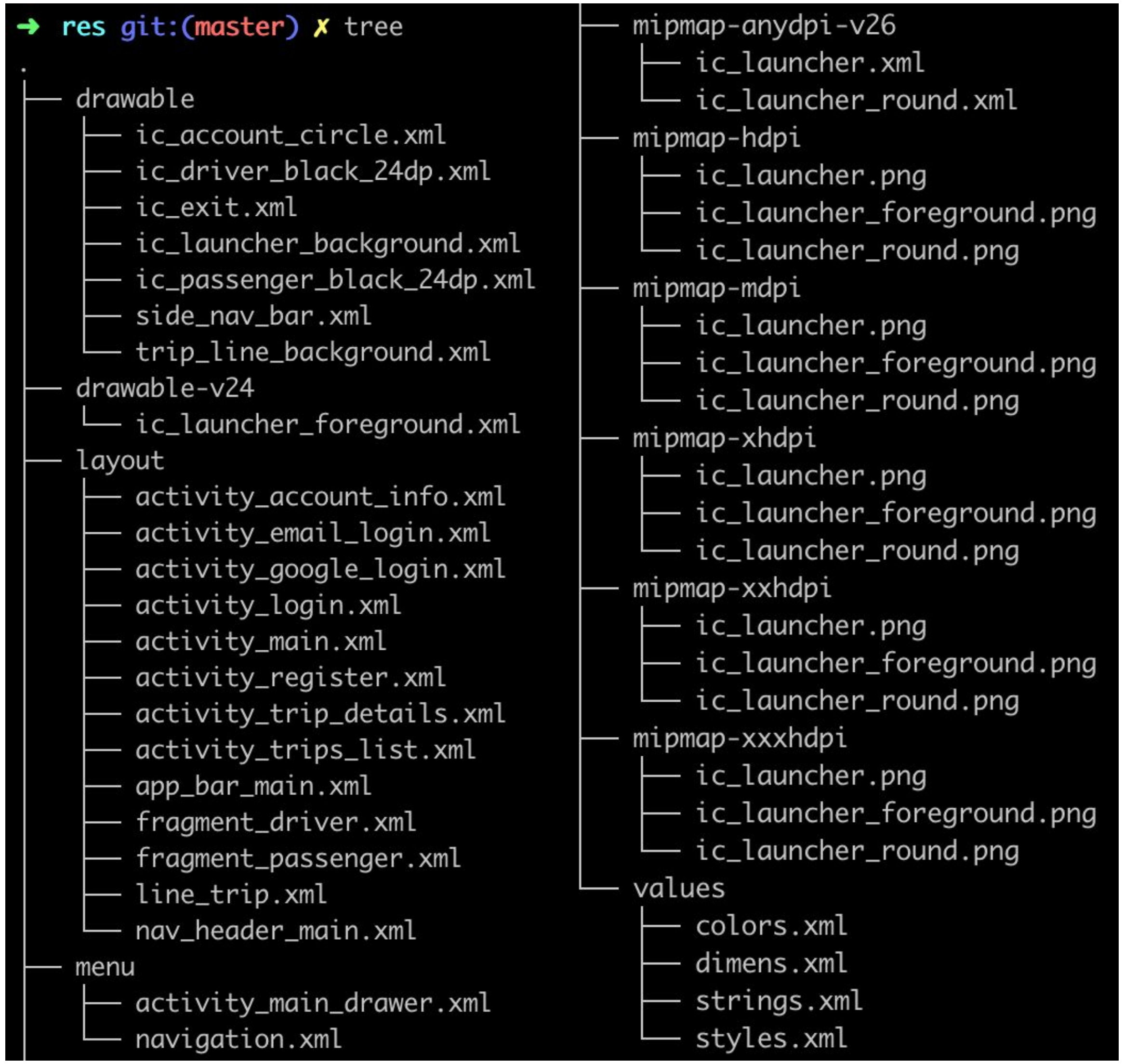
Consumers: Listeners



Application Catalog



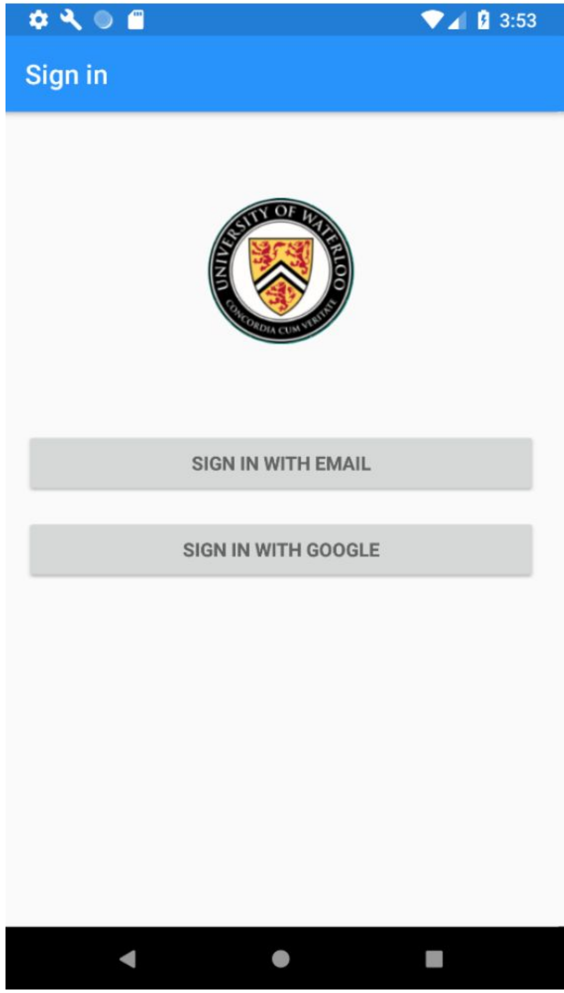
```
➜ waterloocarpool git:(master) ✗ tree
.
├── backend
│   ├── UserInfo.kt
│   ├── api
│   │   ├── Auth.kt
│   │   ├── Store.kt
│   │   └── TaskDecorators.kt
│   └── bean
│       ├── Trip.kt
│       └── User.kt
├── frontend
│   ├── activities
│   │   ├── AccountInfoActivity.kt
│   │   ├── EmailLoginActivity.kt
│   │   ├── GoogleLoginActivity.kt
│   │   ├── LoginActivity.kt
│   │   ├── MainActivity.kt
│   │   ├── RegisterActivity.kt
│   │   ├── TripDetailsActivity.kt
│   │   └── TripsList.kt
│   ├── adapters
│   │   └── UserAdapter.kt
│   └── fragments
│       ├── DriverFragment.kt
│       └── PassengerFragment.kt
└── global
    └── Constants.kt
```

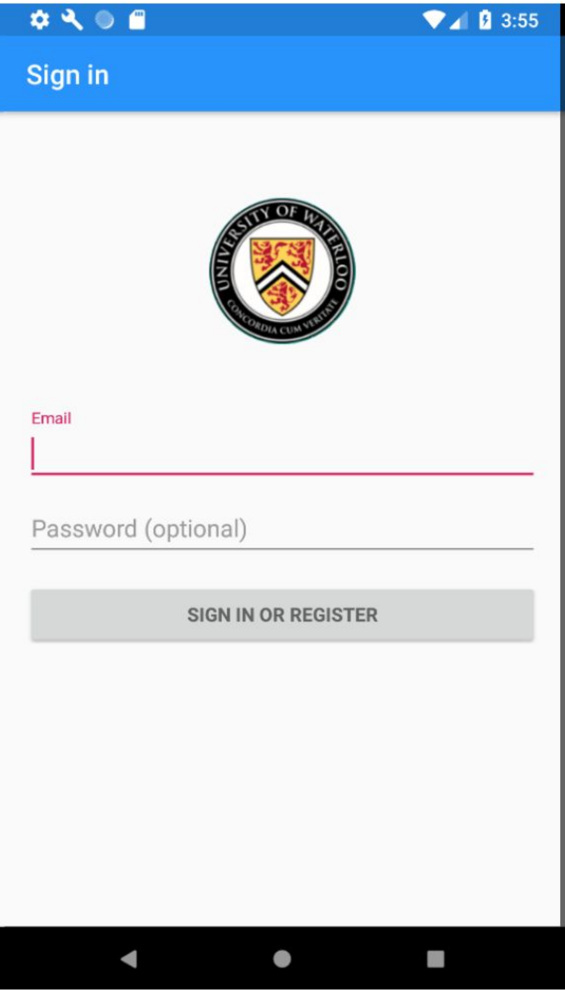


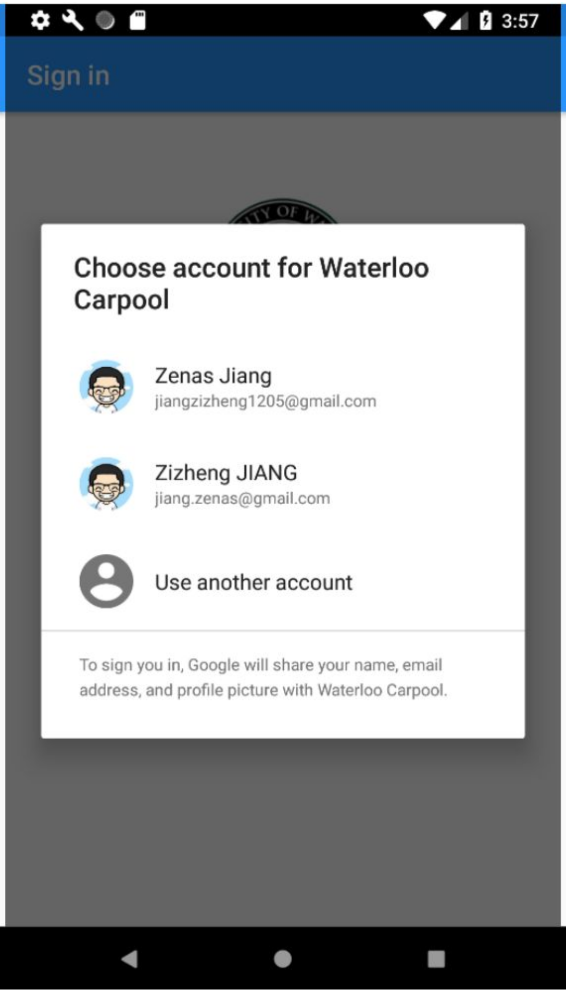
```
➜ res git:(master) ✗ tree
.
├── drawable
│   ├── ic_account_circle.xml
│   ├── ic_driver_black_24dp.xml
│   ├── ic_exit.xml
│   ├── ic_launcher_background.xml
│   ├── ic_passenger_black_24dp.xml
│   ├── side_nav_bar.xml
│   └── trip_line_background.xml
├── drawable-v24
│   └── ic_launcher_foreground.xml
├── layout
│   ├── activity_account_info.xml
│   ├── activity_email_login.xml
│   ├── activity_google_login.xml
│   ├── activity_login.xml
│   ├── activity_main.xml
│   ├── activity_register.xml
│   ├── activity_trip_details.xml
│   ├── activity_trips_list.xml
│   ├── app_bar_main.xml
│   ├── fragment_driver.xml
│   ├── fragment_passenger.xml
│   ├── line_trip.xml
│   └── nav_header_main.xml
├── menu
│   ├── activity_main_drawer.xml
│   └── navigation.xml
├── mipmap-anydpi-v26
│   ├── ic_launcher.xml
│   └── ic_launcher_round.xml
├── mipmap-hdpi
│   ├── ic_launcher.png
│   ├── ic_launcher_foreground.png
│   └── ic_launcher_round.png
├── mipmap-mdpi
│   ├── ic_launcher.png
│   ├── ic_launcher_foreground.png
│   └── ic_launcher_round.png
├── mipmap-xhdpi
│   ├── ic_launcher.png
│   ├── ic_launcher_foreground.png
│   └── ic_launcher_round.png
├── mipmap-xxhdpi
│   ├── ic_launcher.png
│   ├── ic_launcher_foreground.png
│   └── ic_launcher_round.png
├── mipmap-xxxhdpi
│   ├── ic_launcher.png
│   ├── ic_launcher_foreground.png
│   └── ic_launcher_round.png
└── values
    ├── colors.xml
    ├── dimens.xml
    ├── strings.xml
    └── styles.xml
```

UNIVERSITY OF WATERLOO

Functional Properties - Login and Register







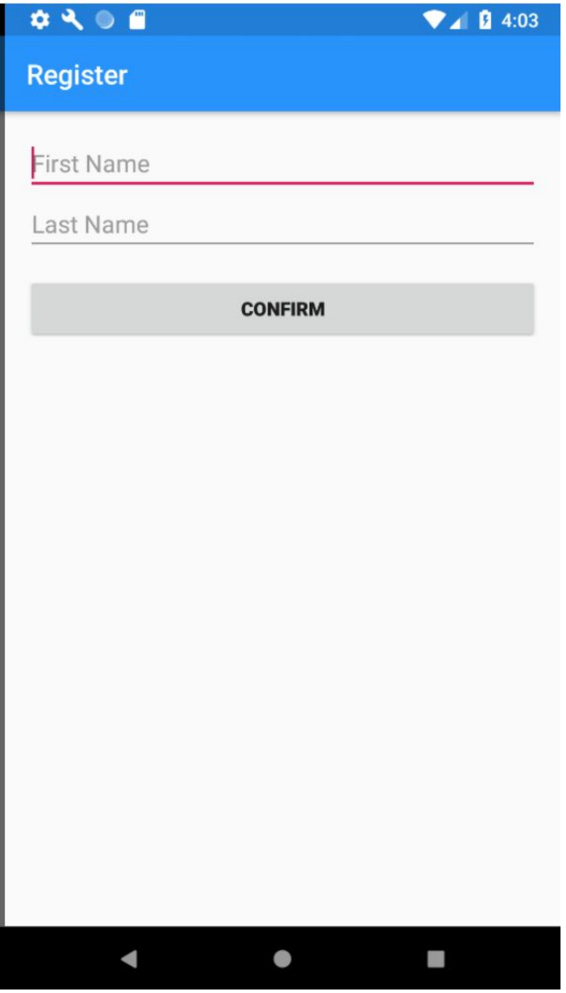
Choose account for Waterloo Carpool

Zenas Jiang
jiangzizheng1205@gmail.com

Zizheng JIANG
jiang.zenas@gmail.com

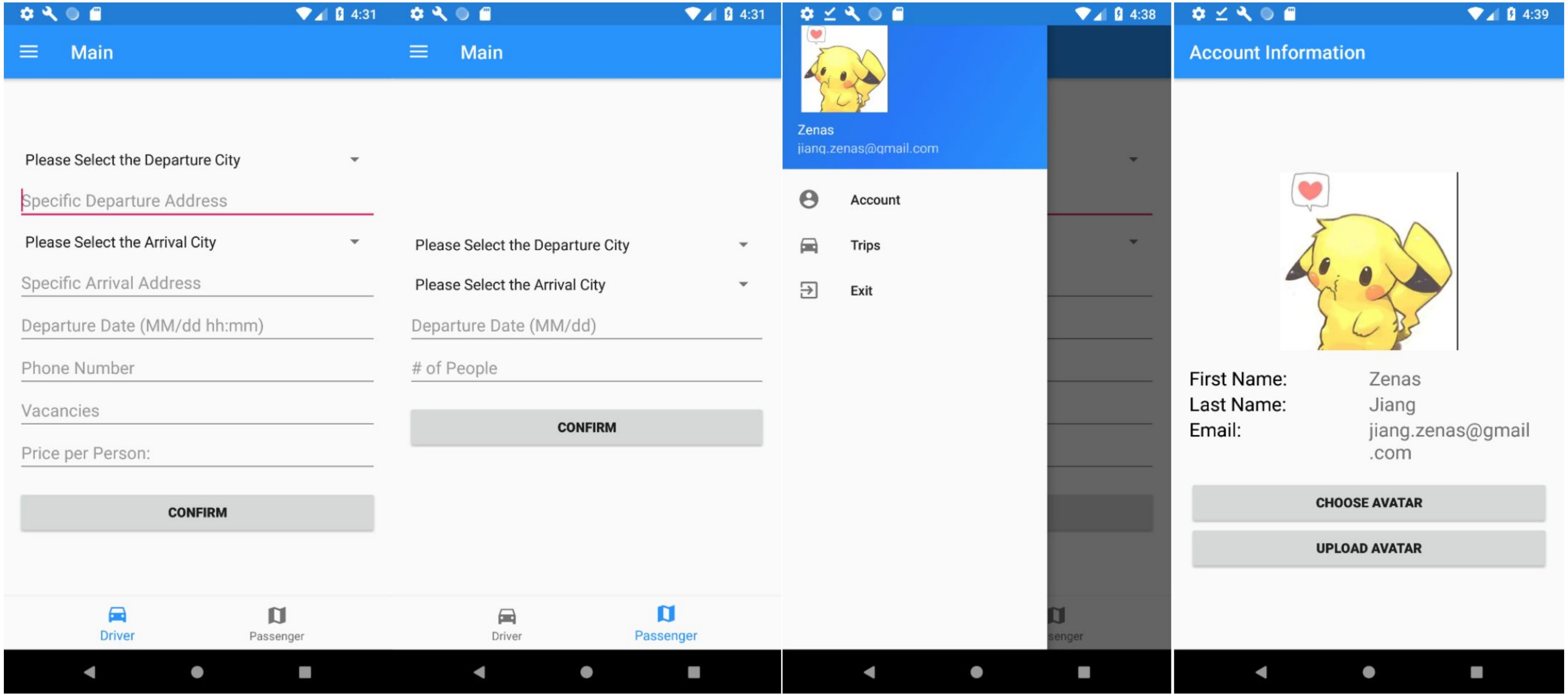
Use another account

To sign you in, Google will share your name, email address, and profile picture with Waterloo Carpool.



https://youtu.be/idMXVOn5yOc

Functional Properties - Main and Drawer


4:31

Main

Please Select the Departure City

Specific Departure Address

Please Select the Arrival City

Specific Arrival Address

Departure Date (MM/dd hh:mm)

Phone Number

Vacancies

Price per Person:

CONFIRM

Driver

Passenger

4:31

Main

Please Select the Departure City

Please Select the Arrival City

Departure Date (MM/dd)

# of People

CONFIRM

Driver

Passenger

4:38

Zenas

jiang.zenas@gmail.com

Account

Trips

Exit

4:39

Account Information

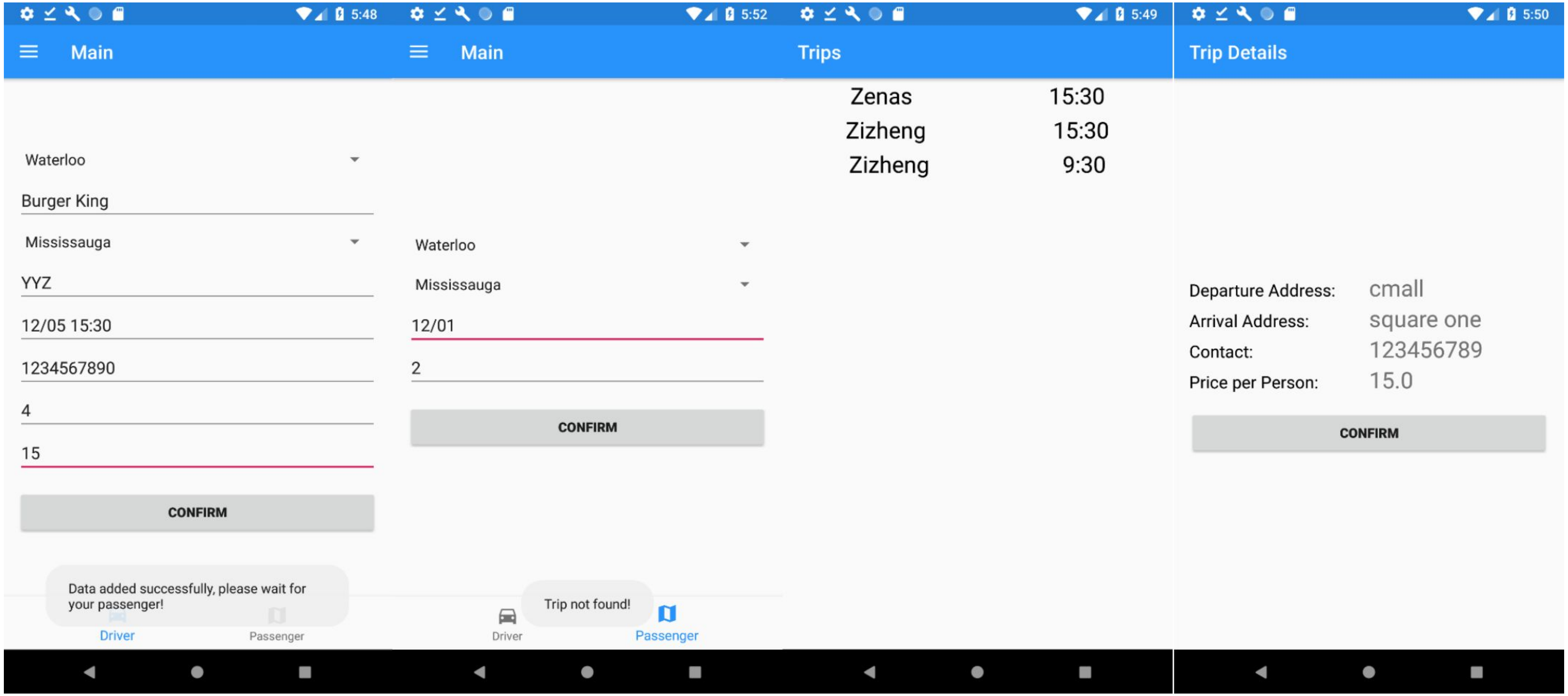
First Name:	Zenas
Last Name:	Jiang
Email:	jiang.zenas@gmail .com

CHOOSE AVATAR

UPLOAD AVATAR

https://youtu.be/idMXVOn5yOc

Functional Properties - Database


5:48

Main

Waterloo

Burger King

Mississauga

YYZ

12/05 15:30

1234567890

4

15

CONFIRM

Data added successfully, please wait for your passenger!

5:52

Main

Waterloo

Mississauga

12/01

2

CONFIRM

Trip not found!

5:49

Trips

Zenas	15:30
Zizheng	15:30
Zizheng	9:30

5:50

Trip Details

Departure Address:	cmall
Arrival Address:	square one
Contact:	123456789
Price per Person:	15.0

CONFIRM

https://youtu.be/idMXVOn5yOc

Non Functional Properties - Response Time

- Recycler View vs. List View:
  1. Recycler View is much better than List View:
     - Load the displayed content in real time.
     - Less memory usage and faster response.
  2. Needs to design each line's layout for this view. Needs to be monitored and ensure that the old data is cleared every time we generate a new adapter.

- Parcelable vs. Serializable:
  - We choose parcelable because the data is already in our memory.
  - Parcelable is more efficient and suitable for our application. (We don't need to transfer data by network and the data doesn't need to be saved).

Non Functional Properties - Scalability & Reliability

- The application can accept at least 20 users online simultaneously.
  1. The NoSQL database of our project is deployed on a professional third-party platform which has 24 x 7 customer support and ensures reliability.
  2. Each user has no authentication of modifying data of other users in the database, so there are no conflicts between each users.

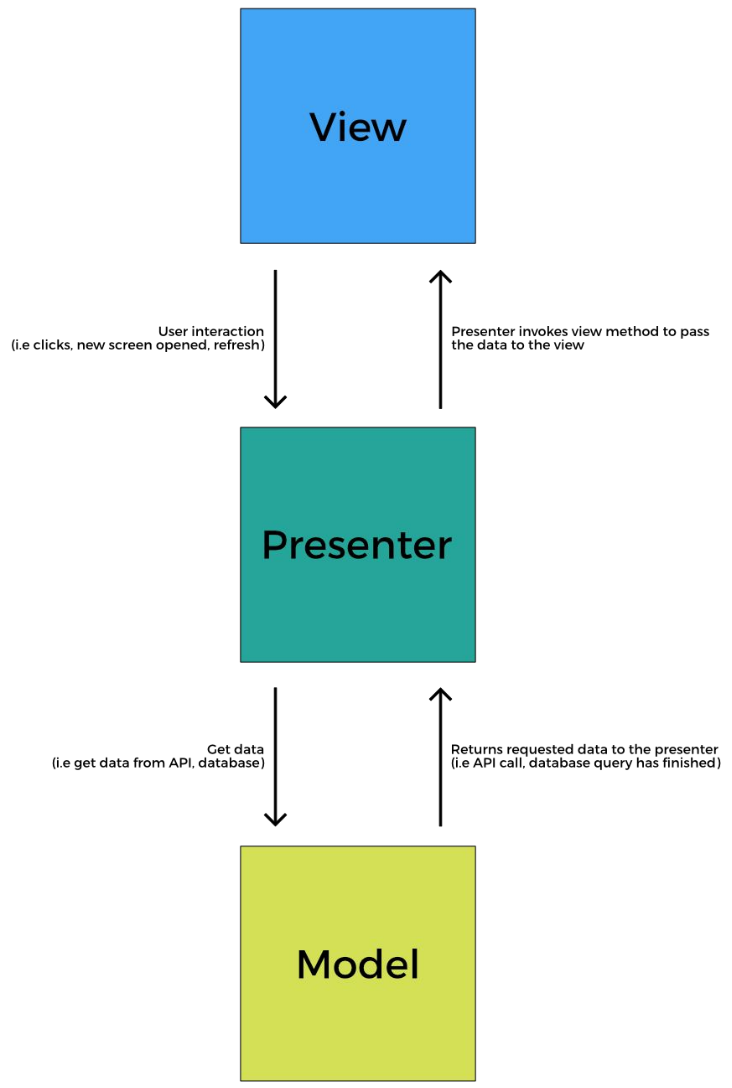
- The Login-Register part of our project allows extension on more login methods, such as Facebook Login, Twitter Login and etc.

Design Patterns - MVP

View: Layouts in Android (mainly .xml files). It cannot contain any business logic and it can only communicate with presenter.

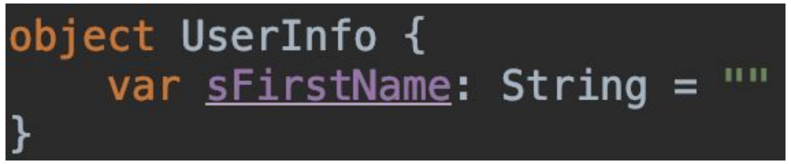
Presenter: Performs needed actions on **Models** and returns the results on **Views.**

Model: responsible for all data related operations (fetching the data from API, querying the database etc.)


View
User interaction
(i.e clicks, new screen opened, refresh)
Presenter invokes view method to pass the data to the view
Presenter
Get data
(i.e get data from API, database)
Returns requested data to the presenter
(i.e API call, database query has finished)
Model

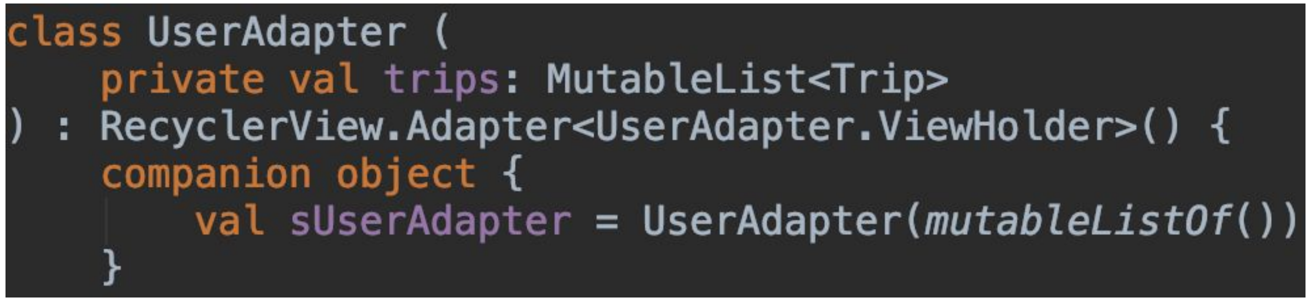
Design Patterns - Singleton design pattern

Each user in our application will have one and only one account and their username implemented by **Auth** and **Register**.

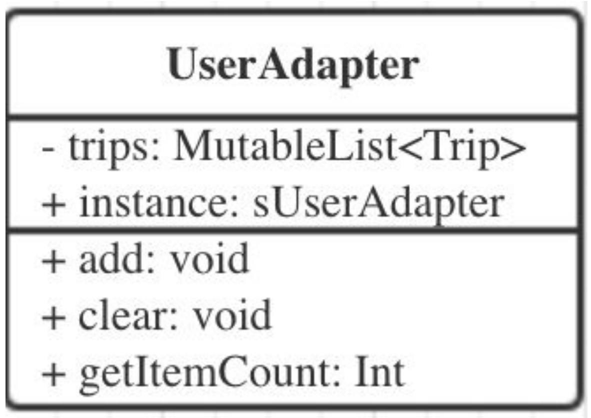


```
object UserInfo {
    var sFirstName: String = ""
}
```

Each user has one and only one adapter for s/he displays related data from the database.

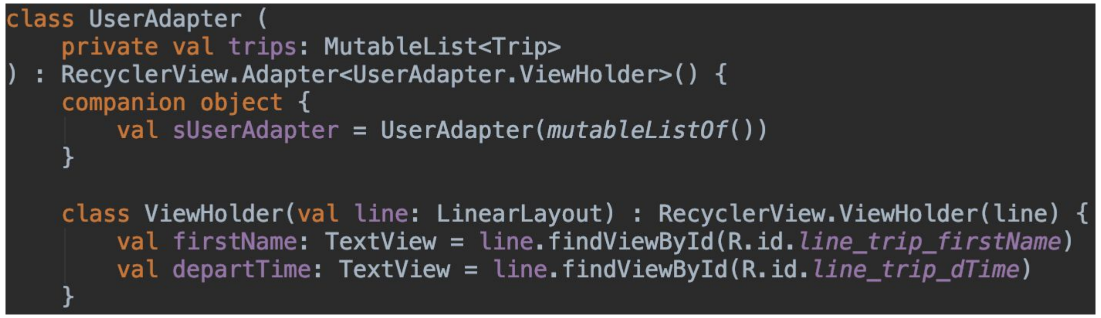


```
class UserAdapter (
    private val trips: MutableList<Trip>
) : RecyclerView.Adapter<UserAdapter.ViewHolder>() {
    companion object {
        val sUserAdapter = UserAdapter(mutableListOf())
    }
```

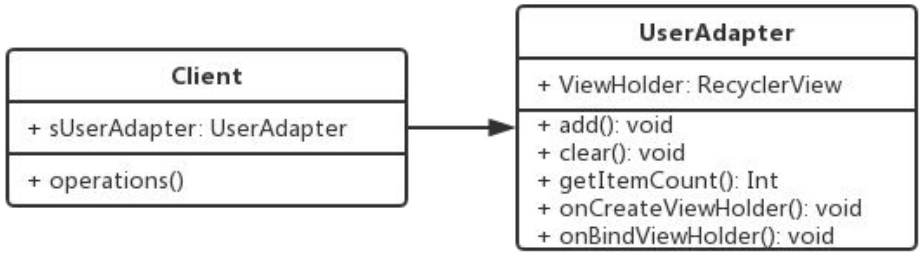


Design Patterns - Adapter Design Pattern

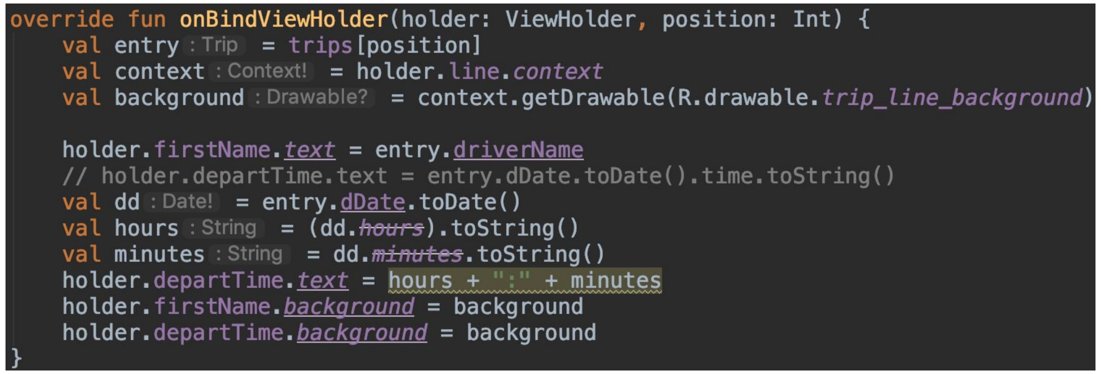
Get the data from database and then and present it in the format we want.



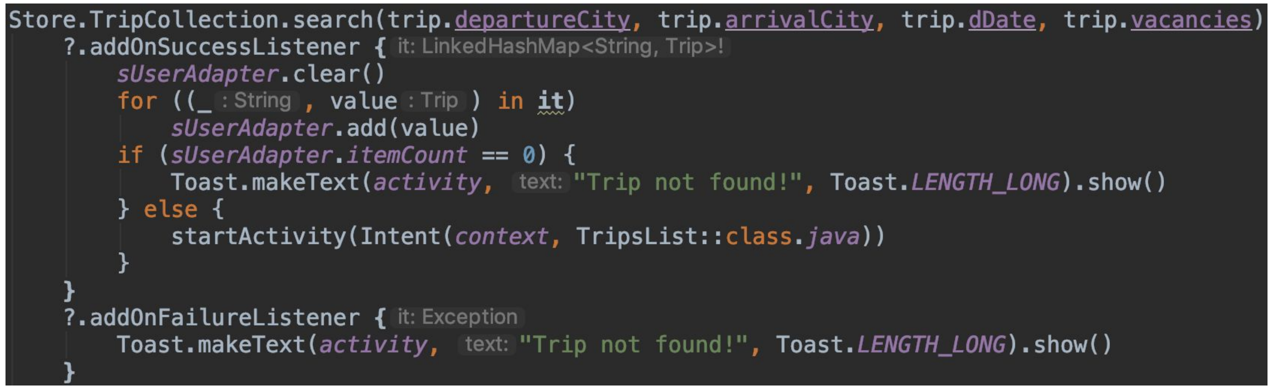
```
class UserAdapter (
    private val trips: MutableList<Trip>
) : RecyclerView.Adapter<UserAdapter.ViewHolder>() {
    companion object {
        val sUserAdapter = UserAdapter(mutableListOf())
    }

    class ViewHolder(val line: LinearLayout) : RecyclerView.ViewHolder(line) {
        val firstName: TextView = line.findViewById(R.id.line_trip_firstName)
        val departTime: TextView = line.findViewById(R.id.line_trip_dTime)
    }
```



Client
+ sUserAdapter: UserAdapter
+ operations()
UserAdapter
+ ViewHolder: RecyclerView
+ add(): void
+ clear(): void
+ getItemCount(): Int
+ onCreateViewHolder(): void
+ onBindViewHolder(): void



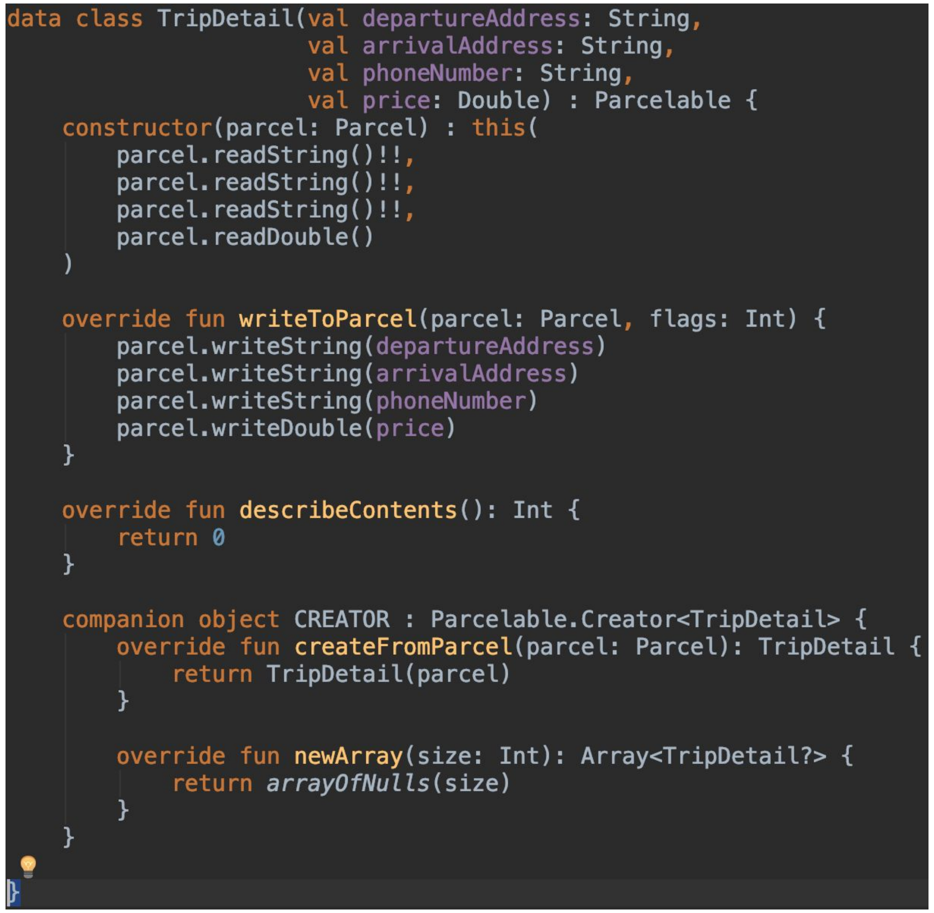
```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val entry : Trip  = trips[position]
    val context : Context!  = holder.line.context
    val background : Drawable?  = context.getDrawable(R.drawable.trip_line_background)

    holder.firstName.text = entry.driverName
    // holder.departTime.text = entry.dDate.toDate().time.toString()
    val dd : Date!  = entry.dDate.toDate()
    val hours : String  = (dd.~~hours~~).toString()
    val minutes : String  = dd.~~minutes~~.toString()
    holder.departTime.text = hours + ":" + minutes
    holder.firstName.background = background
    holder.departTime.background = background
}
```



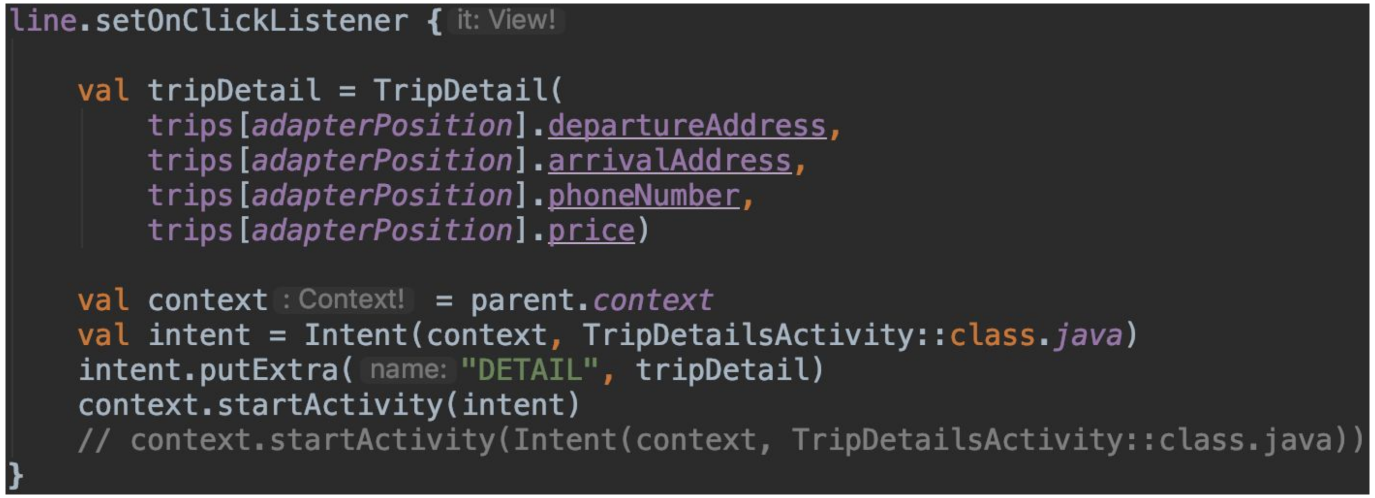
```
Store.TripCollection.search(trip.departureCity, trip.arrivalCity, trip.dDate, trip.vacancies)
    ?.addOnSuccessListener { it: LinkedHashMap<String, Trip>!
        sUserAdapter.clear()
        for ((_ : String , value : Trip ) in it)
            sUserAdapter.add(value)
        if (sUserAdapter.itemCount == 0) {
            Toast.makeText(activity,  text: "Trip not found!", Toast.LENGTH_LONG).show()
        } else {
            startActivity(Intent(context, TripsList::class.java))
        }
    }
    ?.addOnFailureListener { it: Exception
        Toast.makeText(activity,  text: "Trip not found!", Toast.LENGTH_LONG).show()
    }
```

Design Patterns - Data Access Object Design Pattern

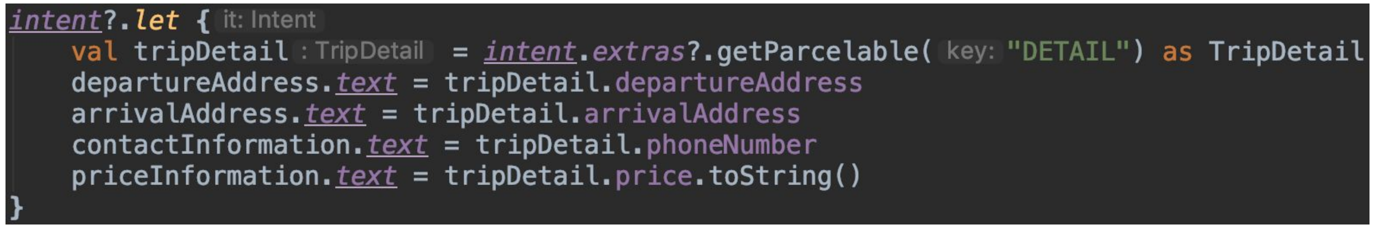
The **Parcelable** in Android offers the data access object interface.



```
data class TripDetail(val departureAddress: String,
                      val arrivalAddress: String,
                      val phoneNumber: String,
                      val price: Double) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString()!!,
        parcel.readString()!!,
        parcel.readString()!!,
        parcel.readDouble()
    )

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(departureAddress)
        parcel.writeString(arrivalAddress)
        parcel.writeString(phoneNumber)
        parcel.writeDouble(price)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<TripDetail> {
        override fun createFromParcel(parcel: Parcel): TripDetail {
            return TripDetail(parcel)
        }

        override fun newArray(size: Int): Array<TripDetail?> {
            return arrayOfNulls(size)
        }
    }

}
```



```
line.setOnClickListener { it: View!

    val tripDetail = TripDetail(
        trips[adapterPosition].departureAddress,
        trips[adapterPosition].arrivalAddress,
        trips[adapterPosition].phoneNumber,
        trips[adapterPosition].price)

    val context : Context! = parent.context
    val intent = Intent(context, TripDetailsActivity::class.java)
    intent.putExtra( name: "DETAIL", tripDetail)
    context.startActivity(intent)
    // context.startActivity(Intent(context, TripDetailsActivity::class.java))
}
```



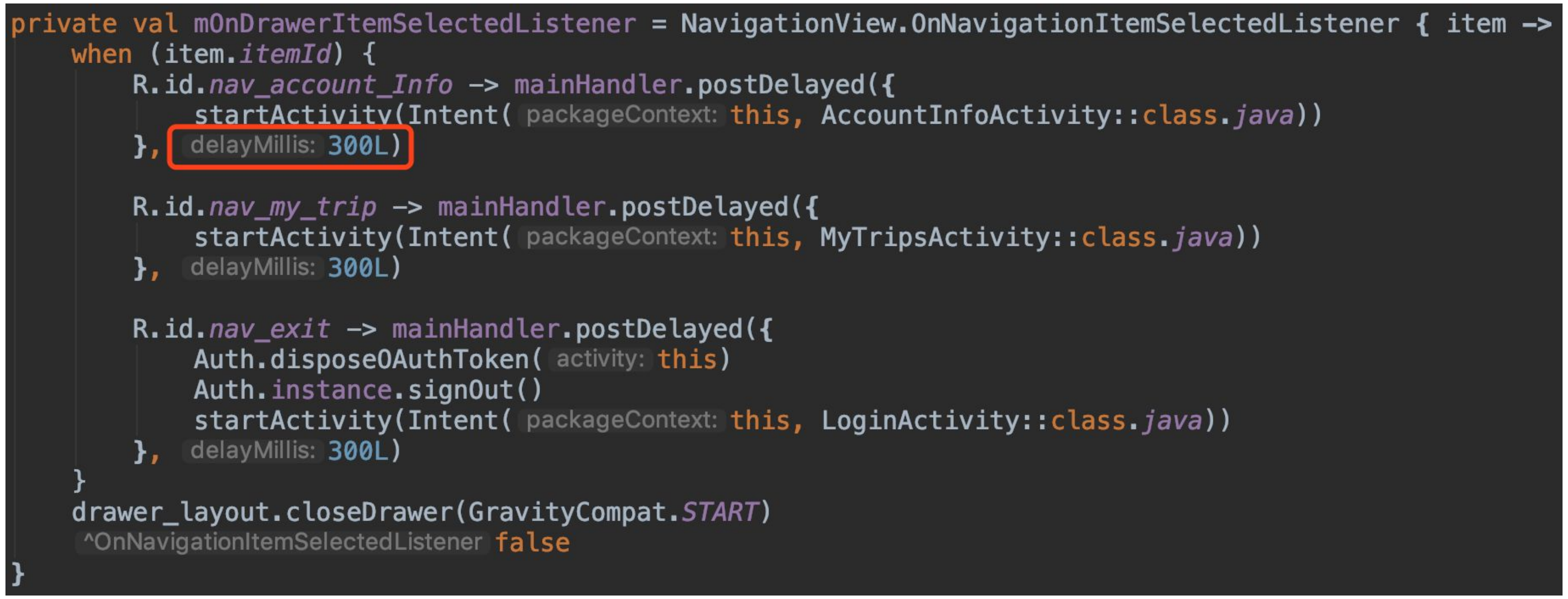
```
intent?.let { it: Intent
    val tripDetail : TripDetail  = intent.extras?.getParcelable( key: "DETAIL") as TripDetail
    departureAddress.text = tripDetail.departureAddress
    arrivalAddress.text = tripDetail.arrivalAddress
    contactInformation.text = tripDetail.phoneNumber
    priceInformation.text = tripDetail.price.toString()
}
```

UNIVERSITY OF WATERLOO

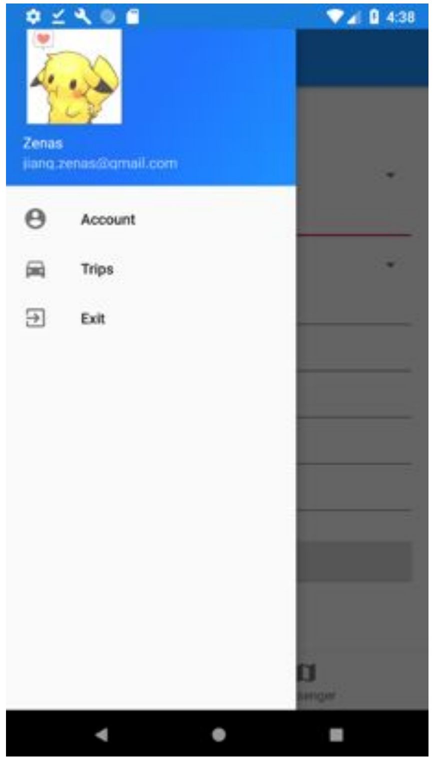
Technical Challenges

- Database: 1. Need to add indexes for some attributes to make sure some query can work correctly. 2. NoSQL doesn’t support range query based on timestamp type. So we use the filter to optimize the database query statement.

- UI animation optimization: The animation may be stuck.



```
private val mOnDrawerItemSelectedListener = NavigationView.OnNavigationItemSelectedListener { item ->
    when (item.itemId) {
        R.id.nav_account_Info -> mainHandler.postDelayed({
            startActivity(Intent( packageContext: this, AccountInfoActivity::class.java))
        }, delayMillis: 300L)

        R.id.nav_my_trip -> mainHandler.postDelayed({
            startActivity(Intent( packageContext: this, MyTripsActivity::class.java))
        }, delayMillis: 300L)

        R.id.nav_exit -> mainHandler.postDelayed({
            Auth.disposeOAuthToken( activity: this)
            Auth.instance.signOut()
            startActivity(Intent( packageContext: this, LoginActivity::class.java))
        }, delayMillis: 300L)
    }
    drawer_layout.closeDrawer(GravityCompat.START)
    ^OnNavigationItemSelectedListener false
}
```



Future improvement

To make our application more **user-friendly**, we are currently implementing some other details such as user profiles functionality (change/upload their avatar). We also want to let users know their current **trips information** by communicating with the databases, but this means that our database needs to be redesigned, which may cause all of activities which communicate with the database need to be rewritten. We are still working on it and considering whether there is a better way to achieve this function.

UNIVERSITY OF WATERLOO

UNIVERSITY OF WATERLOO

Thank you!