Deliverable #3

System Architecture & Design

Team Name : **Rental Central**

Application Name : **Sturents**

Written By:

Jake Jazokas - 101083496 - Lead Design

Porya Isfahani – 100937103 - Cloud Architect

AJ Ricketts – 101084146 - Front-End Developer

Jeffery Chen - 101070633 - Mobile Specialist

Link to all the diagrams in case they are too compressed to read
https://drive.google.com/open?id=16ZQ01oBK82O3IHnQ5Up2qQzDtPAsZUUh

Architecture

Identification

- The android application follows the architectural pattern of MVC, and is written in an object oriented style.
- The interface between the RDS server and the application implements a client-server style, where the client is the instance of the application and the server is the RDS server.

Description

Architectural Patterns

MVC

- The overarching pattern of our application is MVC, the controllers being the activities:
  - ExpandedCardViewActivity
  - MainActivity
  - MyListingsActivity
  - MySettingsActivity.
- The models are:
  - Listings
  - SavedListingsArray.
- The views are given in the xml files as well as RentalCardView.
- The controllers are called when the application starts, or the user switches to a new window. They handle the events for changing the current window that is displayed. The controllers also parse the json data from the server and create Listing models with that data.
- The model contains the details for a given listing, and the SavedListingArray contains an array of all the saved listings for a user, this structure is written to local storage in order to be retained and accessed by other parts of the application.
- The views are able to query the SavedListingArray in order to get the saved listings data for the current user. The RentalCardView is updated based on the Listing data that the MainActivity controller parses from the local json.

Styles

Client-Server

- The interaction between the server and the android application uses the client-server design pattern. There can be many different applications (clients) connecting to the server at the same time. The server keeps a populated table full of listings, that the application (client) can query in order to populate their local fields.

Object Oriented

- The application uses an object oriented style throughout the application code in order to encapsulate the many different objects that the application contains, in order to provide fluid interconnection between the objects.

Justification

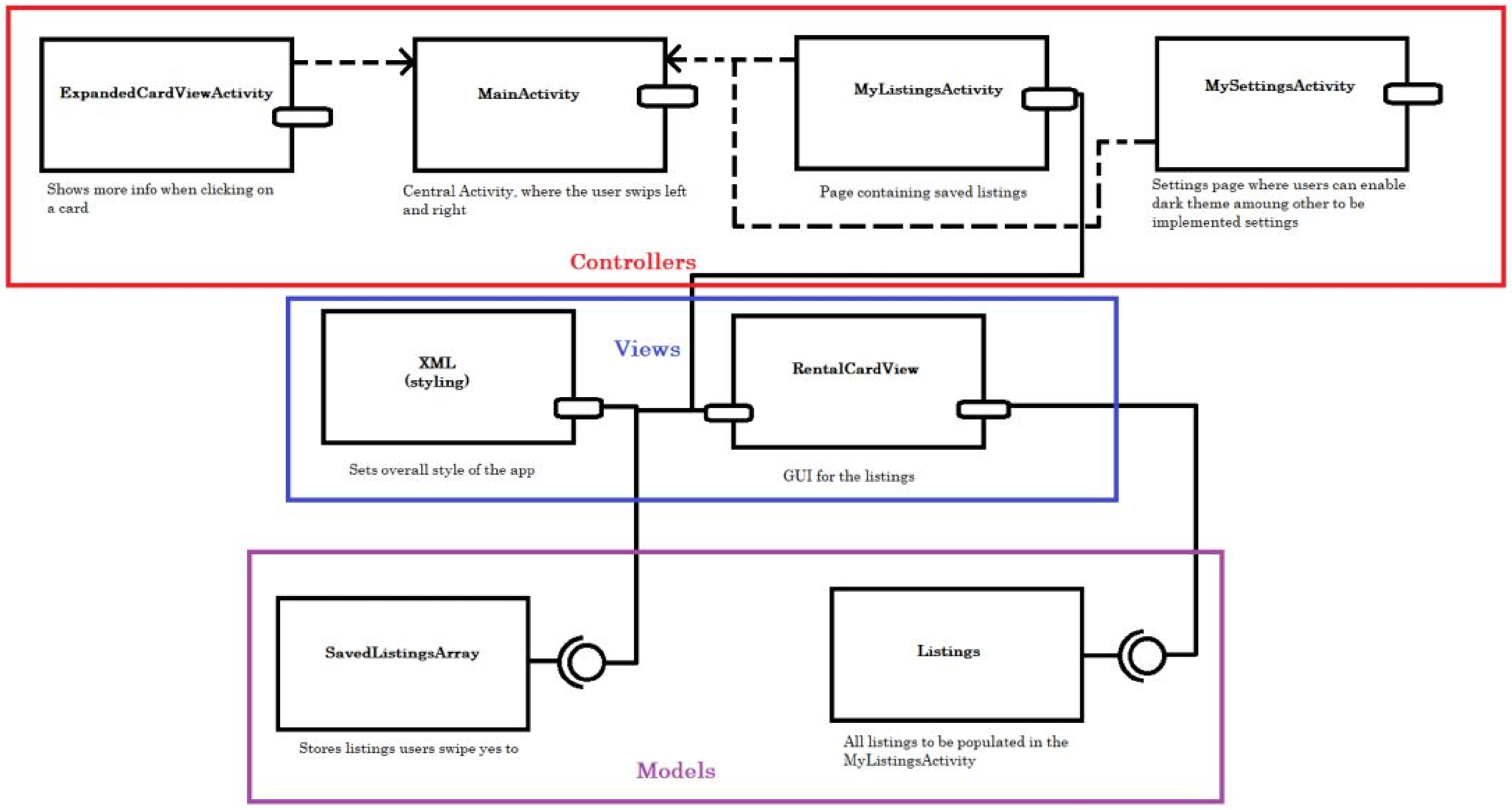
Styles Justification

Looking at the functional requirements, the use of the RDS within our app meant there needed to be some sort of client-server interaction. To meet our non-functional requirements we needed to potentially be able to deal with a large number of users and have this be scalable, because of this, the client-server style exactly fit our needs. The server is designed specifically for this exact use and remains cost effective for our projected user base right now. Given the many activities, views, and adapters needed for our application, the object oriented programing style was best suited. This allowed for an organized and easy to understand layout for working with the many complex and interrelated parts of the app.

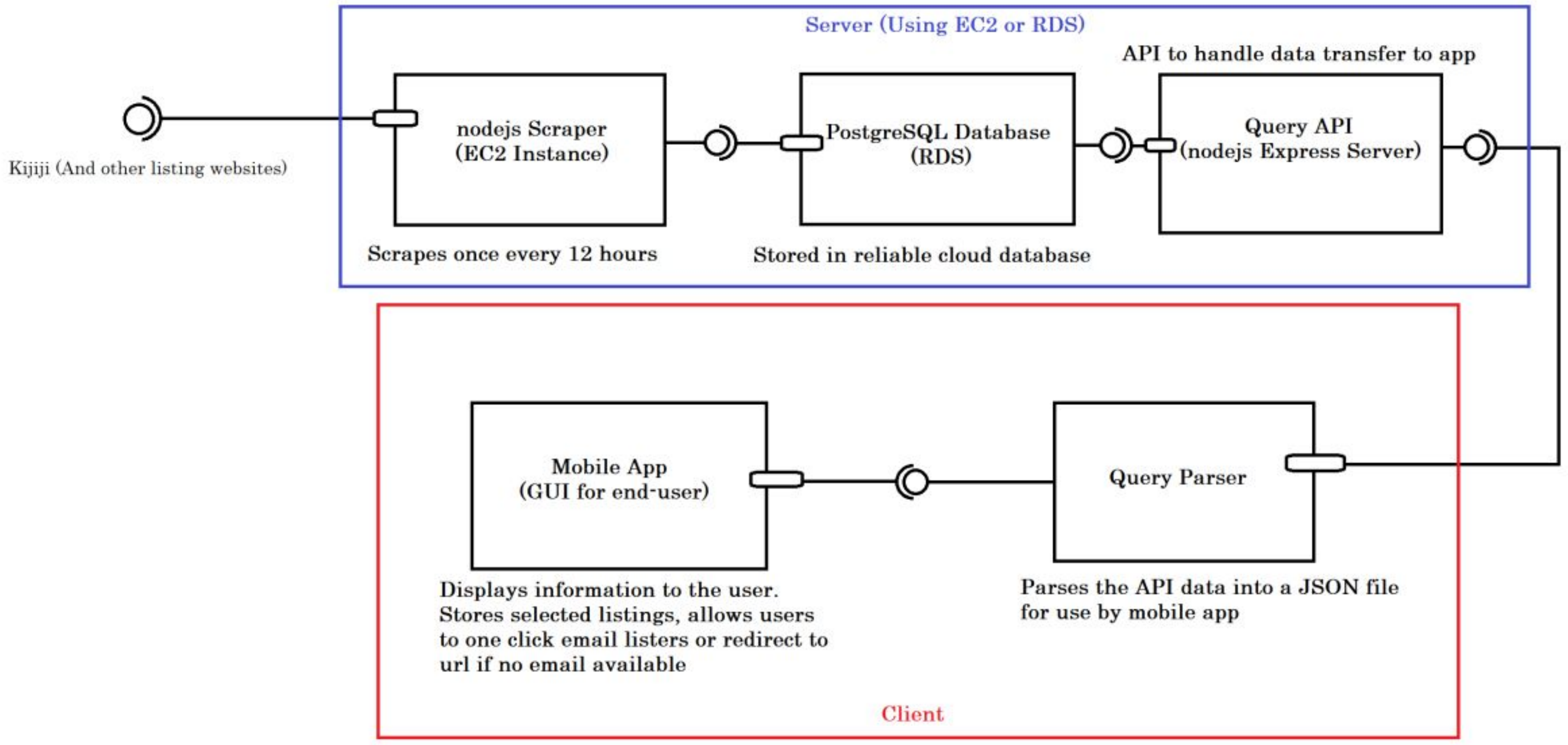
Architectural Patterns Justification

MVC was the most natural design pattern to use while creating the app as there are many different views each with a lot of different functionalities. With the need to handle and store data in the application (in the models) and control the different views (the activities) MVC allowed us to implement our functional properties in a way everyone on the team could easily understand.

MVC Architecture Component Diagram


ExpandedCardViewActivity
Shows more info when clicking on a card
MainActivity
Central Activity, where the user swips left and right
MyListingsActivity
Page containing saved listings
MySettingsActivity
Settings page where users can enable dark theme amoung other to be implemented settings
Controllers
XML (styling)
Sets overall style of the app
Views
RentalCardView
GUI for the listings
SavedListingsArray
Stores listings users swipe yes to
Listings
All listings to be populated in the MyListingsActivity
Models

Client-Server Architecture
Component Diagram


Server (Using EC2 or RDS)
Kijiji (And other listing websites)
nodejs Scraper (EC2 Instance)
Scrapes once every 12 hours
PostgreSQL Database (RDS)
Stored in reliable cloud database
API to handle data transfer to app
Query API (nodejs Express Server)
Mobile App (GUI for end-user)
Displays information to the user. Stores selected listings, allows users to one click email listers or redirect to url if no email available
Query Parser
Parses the API data into a JSON file for use by mobile app
Client

Design

System Design and Structure

1. Node scraper is automatically run within the AWS server, populating the RDS table with the most recent listing data that it scrapes from the internet.
2. The AWS server stores the listing data within an RDS table.
3. When an instance of the application is run, the MainActivity onCreate method is called. Within this method:
   - The view is set and a query is made to the RDS table which returns data that is then parsed into a JSON file. A local file is created on the users device in order to store the users saved listings. The JSON file which contains data from the RDS, is used to create a list of Listing objects. RentalCardView(s) are created for each Listing, and are then added to the SwipePlaceholderView.
4. Within the MainActivity, when the user clicks on a card within the main activity the ExpandedCardView is instantiated and its onCreate method is called:
   - The ExpandedCardView is passed three objects: listingImages, listingDescription, and listingTitle. These objects are used in order to populate the expandedDescription, expandedTitle, and expandedImageSlider views. The expandedImageSlider is populated through the use of the SliderAdapter.
5. Within the MainActivity, when the user clicks on the settings button MySettingsActivity is instantiated and its onCreate method is called:
   - The view is set, and the user is shown the option bars that are able to change range, location, price, and dark mode toggle button.
6. Within the MainActivity, when the user clicks on the saved listings button MyListingsActivity is instantiated and its onCreate method is called:
   - The view is set and the RecyclerView is populated using the RecyclerListingViewAdapter which converts Listing objects to the visible Card Views. Each card view has a listener attached which handles onClick and onSwipe actions.

Design Patterns

Adapter

RecyclerListingViewAdapter is used in order to populate the RecyclerView within MyListingsActivity with Listing objects, the adapter converts the java objects into xml data that is then shown in the view.

SliderAdapter is used in order to populate the SliderView within sturents_expanded_card_view.xml with Images that were retrieved from the urls of the Listing objects.

There will eventually be an adapter that will be used in order to parse the RDS data table into a local JSON file. The reason that this is needed is because the application creates its Listing objects based off of JSON data.

Singleton

The node scraper uses a singleton design pattern, as there is only ever one instance of this file running on the RDS server. This file, when run, will change the data held within the RDS server.

Rationalization

Design Patterns Reasoning

The design of this system was based heavily on the predetermined non-functional requirements. Cost, Scalability, and Usability were big focus points when designing the system. This led us to try and keep as much of the design separate, and future proof as we could.

The adapter architectural design pattern directly supports our functional properties as the web scraper inputs data into an SQL table in RDS, JSON data is then adapted from the RDS table for use in the application. By having this interaction between the web scraper, the RDS table, and the app, the adapter design pattern was best suited as different types of data was easier to work with at the different stages.

The singleton design pattern was used as for the amount of users and scale of the application at the moment, only one instance of the web scraper is needed to populate the RDS table.

Applicability Compared to Alternative Designs

Another approach to the design of our system would be to not use the client-server design style, instead both the scraping (pulling of listings) and the storing of listings could have been done on the device. Firstly, this would have been less scalable, used more memory, and could have been more resource intensive. Our design is superior to this approach because the tasks are delegated to systems, this in turn reduces the resource and storage requirements for the user.

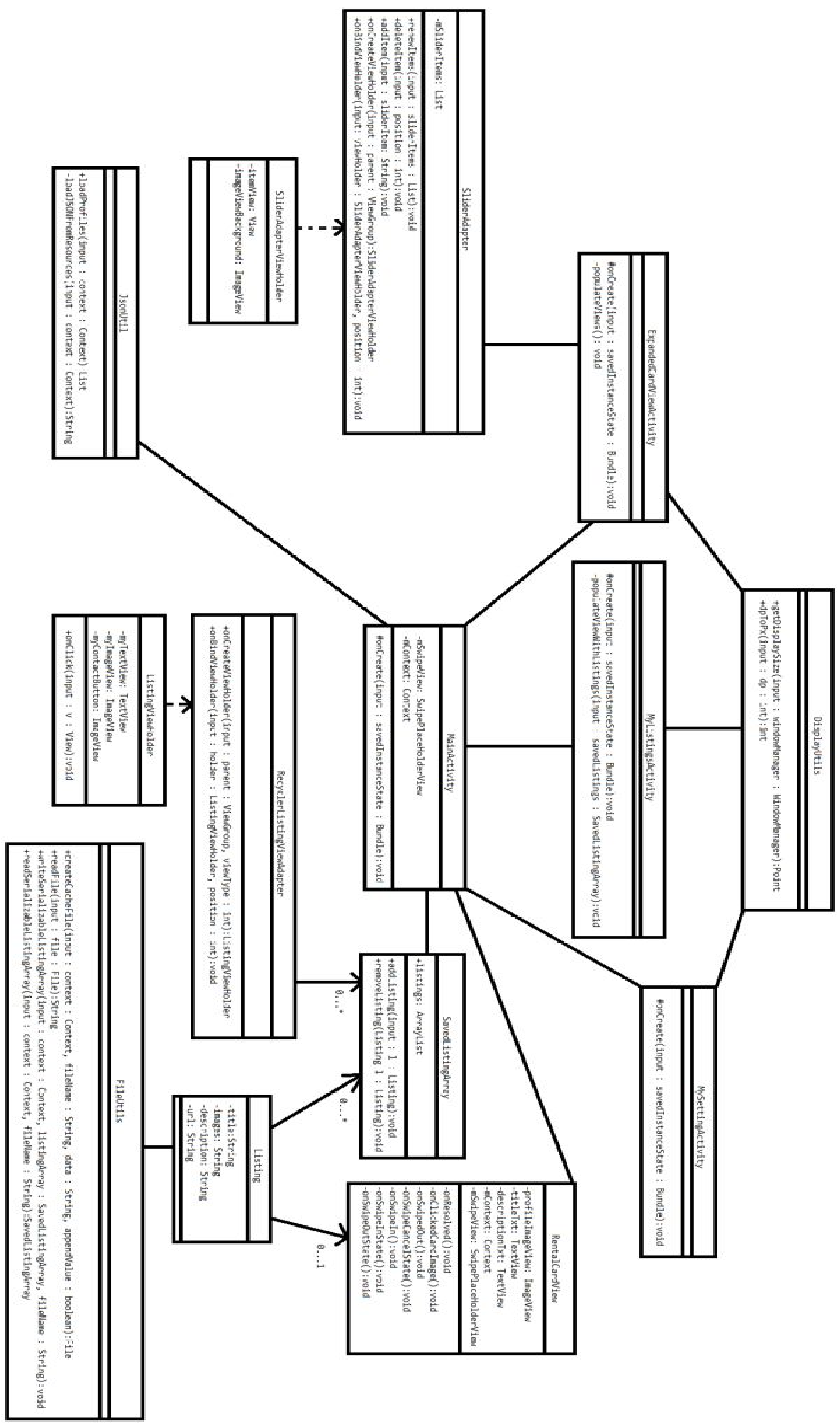
Minimizing Coupling and Accommodating Changing Requirements

By understanding the potential for high coupling with the object-oriented programming style, the system was designed to combat this by having the functionality of the various parts separated. This was achieved by attempting to have only data passed between systems (the scraper, RDS, and application) as well as the classes (classes are mainly independent of each other besides passing the listings data or the context between each other). Implementing a loosely coupled system has allowed for the possibility of accommodating changing requirements in the future. As long as the data being passed remains the same type, the system will continue to function. For example if a feature was removed from the application, major parts of the application would not have to be rewritten. This practice of loose coupling has already allowed us to change various things in the application to accomplish the same thing (changing the scraper from Python to NodeJS to increase the speed of pulling listings).

One way the system may need to evolve, and how we would support it

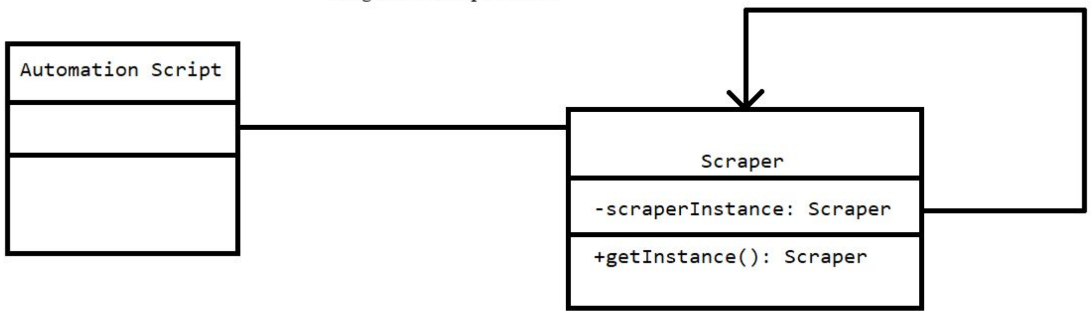
One way the system needs to evolve is the singleton architecture design pattern used by the scraper. In our functional properties, the ability to change location and range was outlined, which is restricted at the moment by the singleton approach. To implement these last features, there would need to be more than one instance of the scraper running to handle all the different locations. In order to accommodate multiple users possibly all using the change location feature, there would need to be a different approach taken for this part of this system.

UML Diagram for App Design

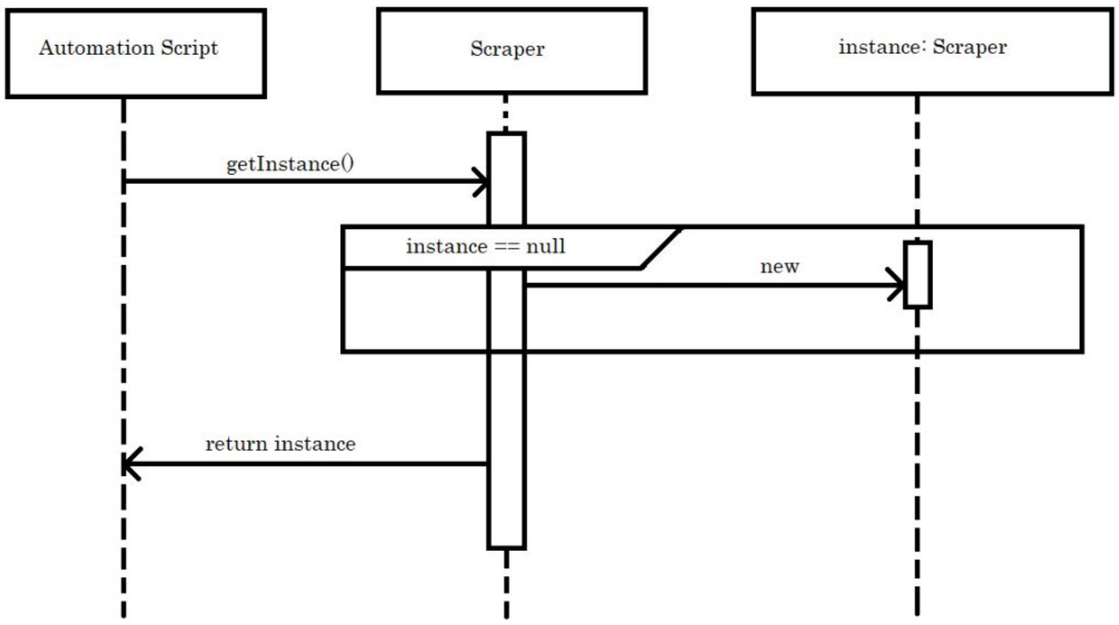

DisplayUtils
+getDisplaySize(input : windowManager : WindowManager):Point
+dpToPx(input : dp : int):int
MySettingActivity
#onCreate(input : savedInstanceState : Bundle):void
ExpandedCardViewActivity
#onCreate(input : savedInstanceState : Bundle):void
-populateViews(): void
MyListingsActivity
#onCreate(input : savedInstanceState : Bundle):void
-populateViewWithListings(input : savedListings : SavedListingArray):void
RentalCardView
-profileImageView: ImageView
-titleTxt: TextView
-descriptionTxt: TextView
-mContext: Context
-mSwipeView: SwipePlaceHolderView
-onResolved():void
-onClickedCardImage():void
-onSwipedOut():void
-onSwipeCancelState():void
-onSwipeIn():void
-onSwipeInState():void
-onSwipeOutState():void
SliderAdapter
-mSliderItems: List
+renewItems(input : sliderItems : List):void
+deleteItem(input : position : int):void
+addItem(input : sliderItem: String):void
+onCreateViewHolder(input : parent : ViewGroup):SliderAdapterViewHolder
+onBindViewHolder(input: viewHolder : SliderAdapterViewHolder, position : int):void
MainActivity
-mSwipeView: SwipePlaceHolderView
-mContext: Context
#onCreate(input : savedInstanceState : Bundle):void
SavedListingArray
+listings: ArrayList
+addListing(input : l : Listing):void
+removeListing(Listing l : Listing):void
0...*
0...*
0...1
SliderAdapterViewHolder
+itemView: View
+imageViewBackground: ImageView
RecyclerListingViewAdapter
+onCreateViewHolder(input : parent : ViewGroup, viewType : int):ListingViewHolder
+onBindViewHolder(input : holder : ListingViewHolder, position : int):void
Listing
-title:String
-images: String
-description: String
-url: String
JsonUtil
+loadProfiles(input : context : Context):List
-loadJSONFromResources(input : context : Context):String
ListingViewHolder
-myTextView: TextView
-myImageView: ImageView
-myContactButton: ImageView
+onClick(input : v : View):void
FileUtils
+createCacheFile(input : context : Context, fileName : String, data : String, appendValue : boolean):File
+readFile(input : file : File):String
+writeSerializableListingArray(input : context : Context, listingArray : SavedListingArray, fileName : String):void
+readSerializableListingArray(input : context : Context, fileName : String):SavedListingArray

Link to uncompressed version of the uml
:https://drive.google.com/file/d/1Jg6NwQer6oPykrIC2xfHQPEyZy6x6Ego/view

Singleton Scraper UML



Singleton pattern - Sequence Diagram



Overview Of What Each Member Is Doing

- **Jake Jazokas**
    - Created the node scraper (can be expanded to scrape more sites)
    - Created the application and added the features:
        - The listing swipe card(s) within the main activity
        - The ability to save ‘liked’ listings locally
        - The ability to view an expanded version of each listing
            - The ability to view and swipe through all the images from that listing
        - The ability to view all ‘liked’ listings, and the ability to either swipe them to see a button that will take you to an email/url for the listing or the ability to click on the saved listing to view its expanded version.
        - The blank template for the settings page.
    - (All of the application features thus far have been implemented by me, so I will be focusing on code reviews and helping everyone else finish the needed features for the application)
- **Porya Isfahani**
    - Created Cloud Infrastructure
        - Deploying postgreSQL Database in Amazon Relational Database Service
        - Deploying API server & Scraper server in Amazon Elastic Compute 2 instances
    - Creating Automation Script for scraper to run every 12 hours
    - Pushing scraped data into postgreSQL database with info from the scraper
    - Created API for app to interact with Database
    - Creating app connection with API and parsing data into app readable format
- **AJ Ricketts**
    - Implementing the range and dark mode feature
- **Jeffery Chen**
    - Implementing the ability to change location