VERKADA

Motion Search by Haroon Iftikhar

The application utilizes the MVP (Model-View-Presenter) design pattern. The benefits of which include and are not limited to:

- View more separated from Model. The Presenter is the mediator between Model and View.
- Easier to create unit tests
- All application logic in the Presenter
- One to one mapping between View and Presenter

Presenter

The VerkadaPresenter is responsible to act as the middle man between all the views and models. It retrieves data from the Model and returns it formatted to the view. But unlike the typical MVC, it also decides what happens when you interact with the views.

View

The views, searchFragment and tableFragment, contain a reference to the presenter. The only thing that the view does is call a method from the Presenter every time there is an interface action.

Model

In an application with a good layered architecture, this model is only the gateway to the domain layer or business logic. See it as the provider of the data we want to display in the view.

✓ Cell

Each of the subregions in the search region is an instance of this class

✓ Motion

The instance of this class defines an object that contains all elements of the jsonDictionary used to make a POST request to the server

✓ Server

The POST method of this class runs on an AsyncTask, creating an HttpClient and making a POST request to the url, sending a jsonObject that accumulates an instance of the Motion class, and receiving the http response.

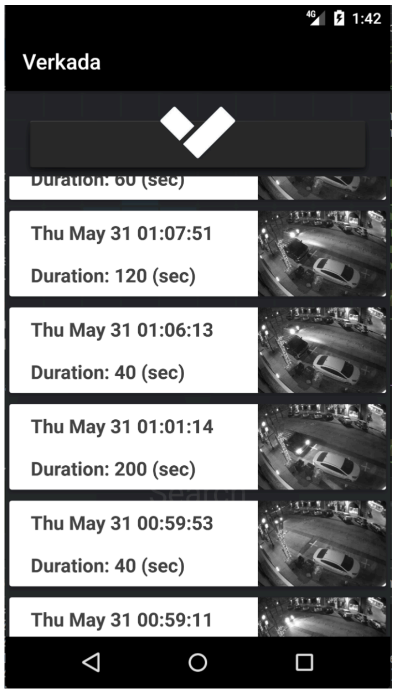
The motionAtParser method of this class parses the http response jsonObject into a two-dimensional list containing two elements: [timestamp, duration of motion in seconds]

✓ VerkadaError

Public enum of errors possible throughout the application

✓ TableFragment

This fragment has the view made up of the results from the http response. The fragment receives a bundle of arguments from the searchFragment's VerkadaPresenter, which is a list of results from the server. This is then set to the zones of the tableFragment's instance of VerkadaPresenter. The recyclerView of the fragment has its adapter set as ResultsViewAdapter.



Duration: 60 (sec)

Thu May 31 01:07:51
Duration: 120 (sec)

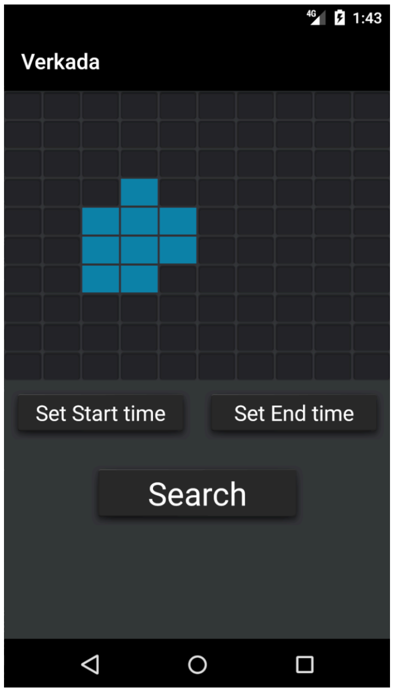
Thu May 31 01:06:13
Duration: 40 (sec)

Thu May 31 01:01:14
Duration: 200 (sec)

Thu May 31 00:59:53
Duration: 40 (sec)

Thu May 31 00:59:11

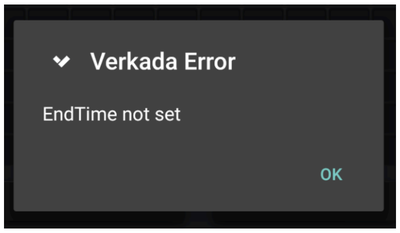
This fragment is only added to the fragment stack, therefore the searchFragment beneath it is preserved. Upon hitting the back button, the tableFragment is destroyed and the searchFragment comes into view.



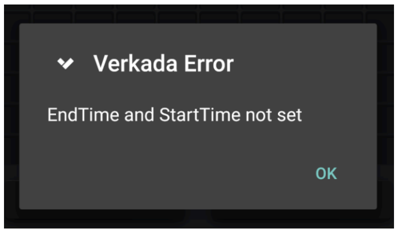
- ✓ SearchFragment

  This fragment has the view of the search region. The search region of 10x10 is made up of cardViews that change background color upon selection/deselection. The startTime and endTime buttons pop up timePickerDialogs that remember the chosen time upon confirmation. These times are now used to initialize the Motion object used to make an http request.

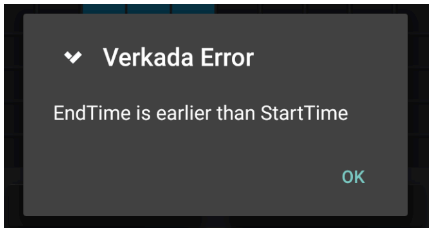
  Error dialogs pop up in case of the startTime, endTime or both not being set.


Verkada Error

EndTime not set


Verkada Error

EndTime and StartTime not set


Verkada Error

EndTime is earlier than StartTime

Note: The initial requirement of making an http request with an endTime of now and a startTime of an hour earlier has been implemented and commented out for the bonus feature of start and end time selection to work

- ✓ LaunchFragment

  This fragment has the view of the initial splash screen made up of the Verkada logo

- ✓ ResultsViewAdapter

    This adapter handles the recyclerView in TableFragment. It decides how many rows need to be in the recyclerView. It updates the timestamp and duration at each row of the recyclerView and makes a GET request for the image thumbnail on an AsyncTask.

- ✓ VerkadaPresenter

    This is the sole presenter of the application. It contains all the logic and calculation methods. This makes it easier to unit test the application since to do so we only have to test this class. It also initiates the startTime and endTime selection timePickerDialogs and remembers the last chosen time for each of them. It creates an instance of the Server class on search button being clicked and runs the POST method on an AsyncTask. It then calls the interface's method to go to the tableFragment once all the results have been parsed from the http response.
    The JUnit tests for this class can be found under the tests package.

- ✓ MainActivity

    This activity implements the interface built. It overrides methods that begin fragment transactions to the designated fragments, adding any argument bundles if needed by the receiver fragment.

- ✓ SplashActivity

    This activity displays the launchFragment before transitioning to the MainActivity using an intent.

- ✓ MainFragmentInteractionListener

    This is the interface built. It includes methods that begin fragment transactions to the designated fragments within the same activity.

✓ Extra

The application utilizes Adaptive launcher icons, which can display a variety of shapes across different device models. Adaptive icons support a variety of visual effects. For example, an adaptive launcher icon can display a circular shape on one OEM device, and display a squircle on another device. Each device OEM provides a mask, which the system then uses to render all adaptive icons with the same shape.

The layouts of the application utilize ConstraintLayout which prevents the view hierarchy from increasing at an ever increasing rate on complex layouts.

The application utilizes packaged fonts, which are included in the assets folder.