GitHub Username: nireno

Problem:

At my workplace, employees are required to track their time spent on tasks and categorise these tasks by type (development, external-meeting etc). Excel spreadsheets are still used as the main tracking method which poses a problem especially if employees are off site or simply away from their PCs.

Proposed Solution: Slice

Description

Slice is a time tracker that provides a calendar-like interface for managing time logs.
It uses geofencing to conveniently start and stop time tracking when entering and leaving the workplace.

Intended User

This app is primarily intended as an in house time tracker for engineers and developers at my workplace.

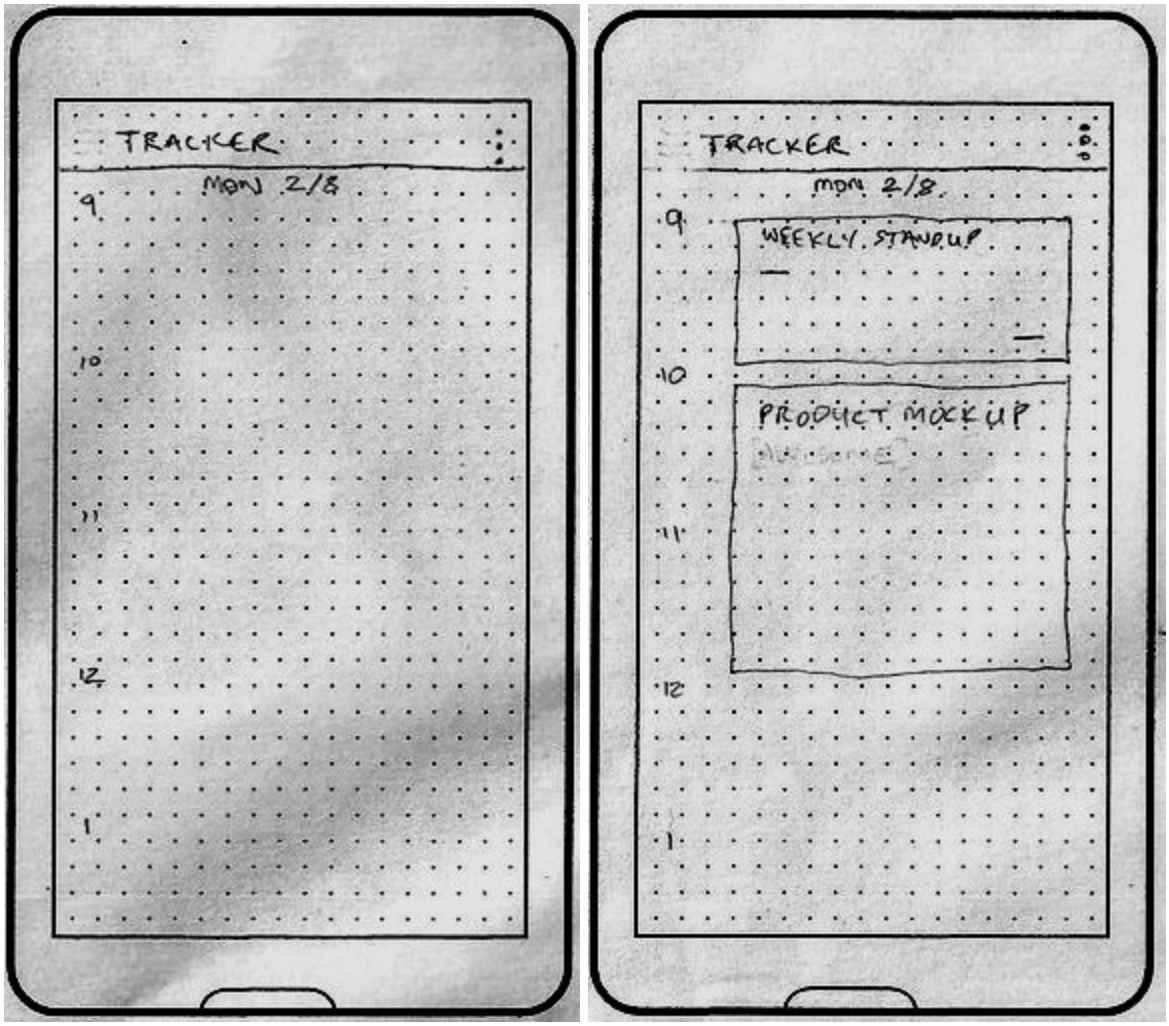
Features

The following are the main features:

- Calendar-like interface for visualizing time spent on tasks
- Widget to display the current task
- Automatically start and stop time tracking when entering and leaving the workplace.

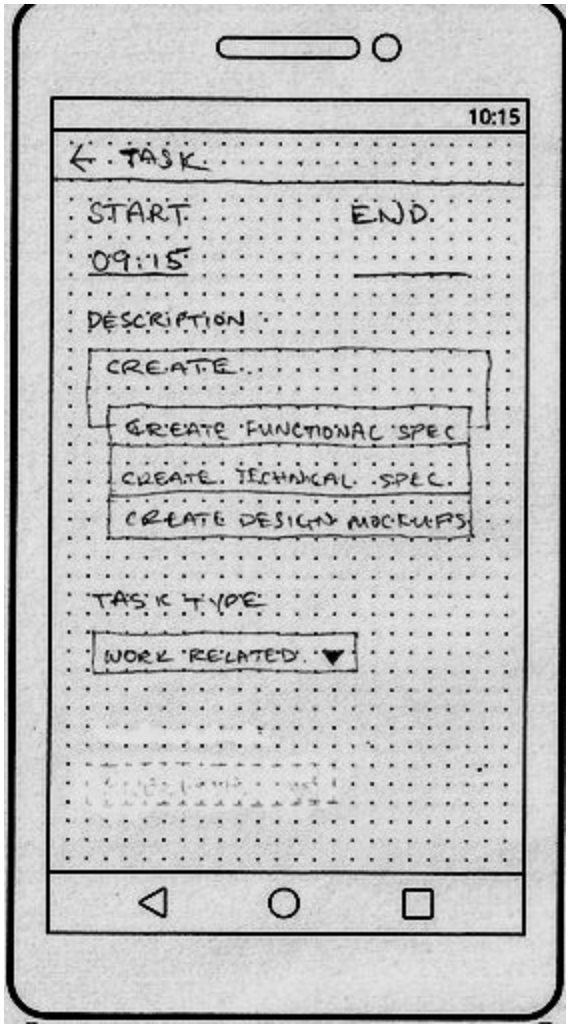
User Interface Mocks

Empty Time Tracker Screen



This is the main time tracker view. The interface is similar to a calendar app with the time on the left and the date at the top. Tap anywhere in the calendar view to bring up the task editor. The time at the location of the tap will be used as the task's start time in the Edit Task view. Tap on an existing task to edit it.

Edit Tasks Screen

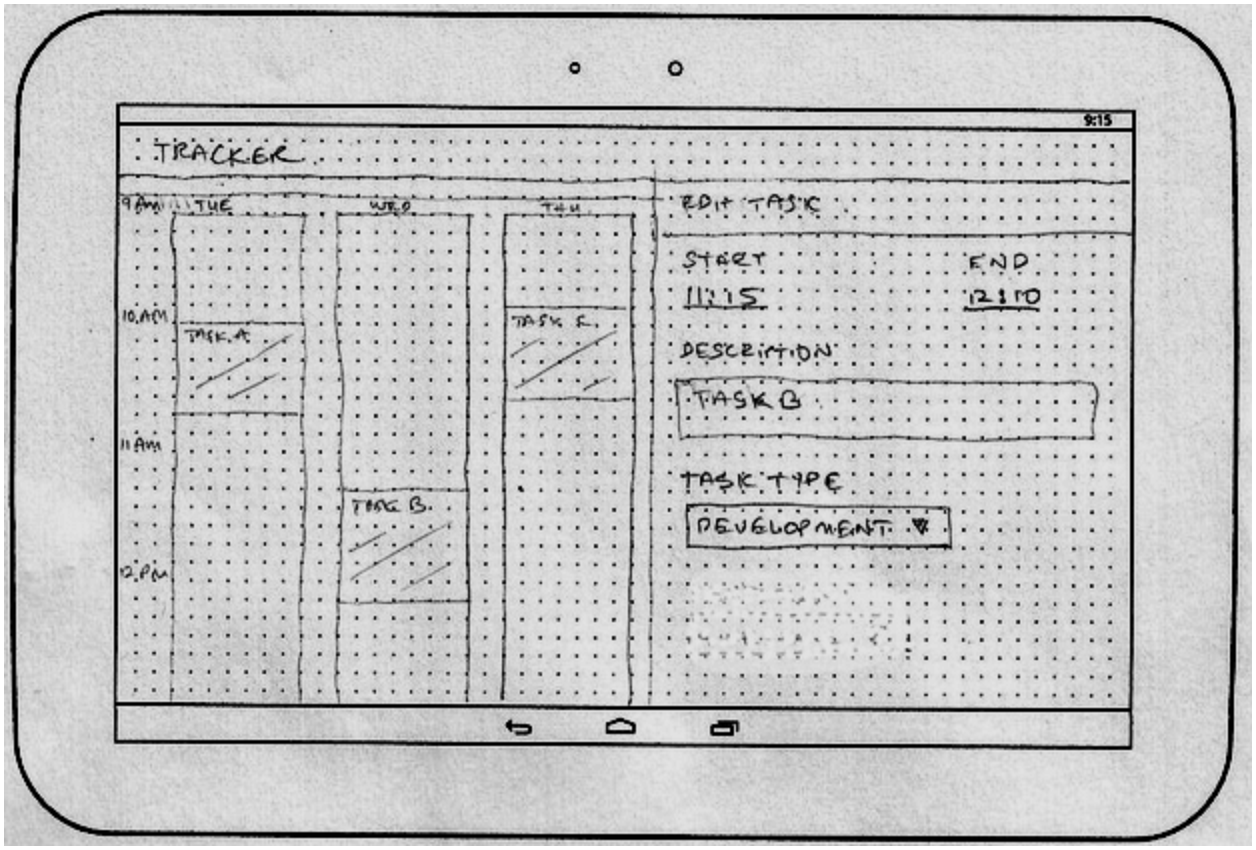


The edit task screen is activated for two cases:

1. you tap an empty spot in the calendar - Start time is automatically filled with the time at the location of the tap all.
2. you tap an existing time log in the calendar - Allows you to edit the log entry

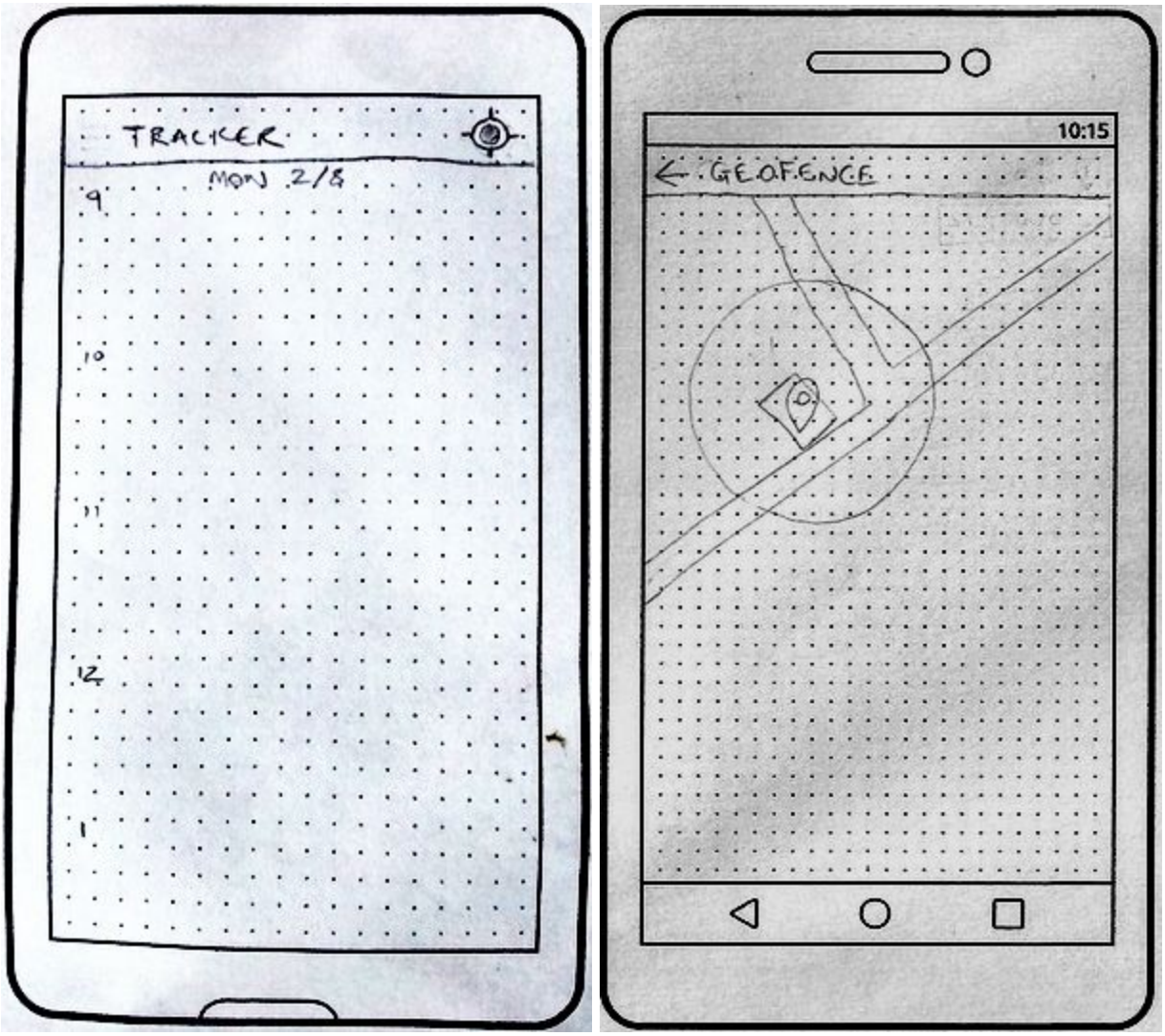
While typing the description the user will get a live search dropdown of previous tasks with similar descriptions.

Tracking time on Tablet



On tablet devices the Tracker and Edit Task screens will be shown in a master/detail layout.

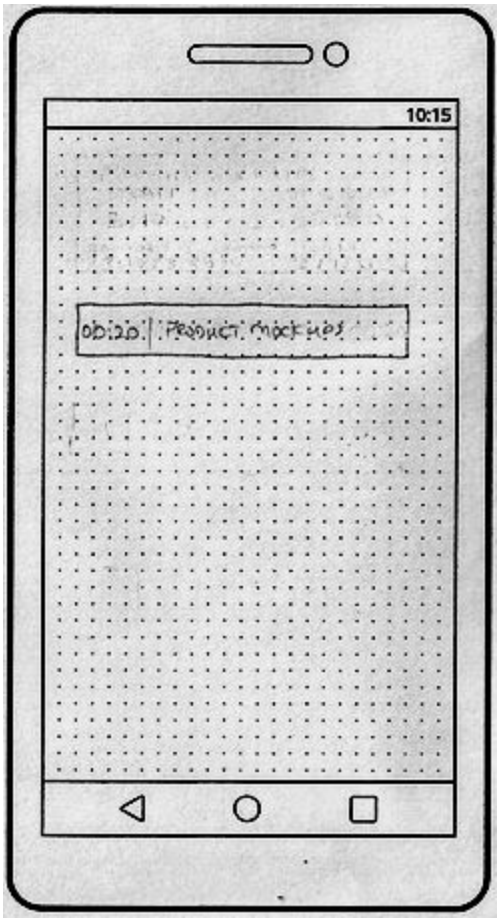
The settings page



Tap the location finder icon in the menu bar to bring you to a google map activity for selecting a location around which a geofence will be placed. The geofence will be used to set an initial start time so that if there are not any tasks set for the day, this will be used as the default start time for the first task.

Also on leaving the geofence any running task will be stopped such that its end time is set to the time of exiting the fence.

Homescreen Widget



The home-screen widget will display the current running task and it’s elapsed time. Tapping on the widget will take you to the EditTaskActivity for the running task.

Key Considerations

Handle data persistence

Data to be stored:
Geofence Location - SharedPreferences
Task and Time log data - SQLite + Content provider

Corner cases in the UX

How to display task with just a start time on the calendar view: have a minimum card size in the calendar view. If the difference between the start time and current time will result in a calendar entry that is too small, just use an appropriate minimum size.
If there is no task without an end time (i.e no running task) the homescreen widget will show the duration of the last task.

Libraries

Android-Week-View - Provides the calendar view functionality for tasks and capturing the click events with the appropriate timestamp.

MaterialDateTimePicker - Provides a familiar interface for picking a time which is also easier, and quicker to use.

Next Steps: Required Tasks

Project Setup

Setup a new Android Studio project and include the following dependencies in its build.grade as follows:

```
dependencies {
  compile 'com.wdullaer:materialdatetimepicker:2.2.0'
  compile 'com.github.alamkanak:android-week-view:1.2.6'
}
```

These are the dependencies for Android-Week-View and MaterialDateTimePicker.

Implement Content Provider

Classes to implement

- LogEntry: should implement BaseColumns to define the table for storing time logs
- SliceContract Class: encapsulates the LogEntry table
- SliceDbHelper: an SQliteOpenHelper which uses the contract to manage creating and upgrading the SQLite database.
- SliceProvider: A ContentProvider for accessing LogEntry data.

References:
http://developer.android.com/training/basics/data-storage/databases.html
http://developer.android.com/guide/topics/providers/content-provider-basics.html
http://developer.android.com/guide/topics/providers/content-provider-creating.html
https://github.com/udacity/Advanced_Android_Development/tree/master/app/src/main/java/com/example/android/sunshine/app/data

Implement UI for Each Activity and Fragment

Build the following UI elements:

- MainActivity (Calendar view): The main Activity displays the calendar. See the documentation for setting up and configuring the view for click events in empty space and on existing calendar “events”. Use the SliceProvider to read LogEntry data for the current day.
- EditTaskActivity: Use the MaterialDateTimePicker library when building the start and end time UI elements. Use the SliceProvider to provide the necessary CRUD functionality. Use AsyncTask to perform live search of existing task descriptions.
- Tablet layout with MainActivity and EditTaskActivity

References:
https://github.com/udacity/Advanced_Android_Development/tree/master/app/src/main/java/com/example/android/sunshine/app/widget
http://developer.android.com/guide/topics/appwidgets/index.html

Implement the Geofence using Google Play Services

The geofence will require the use of the following Google Play Services:

- Location Service:for recognising when the device enters and leaves the fence
- Maps: for providing a settings interface for configuring the position of the fence.

1. Setup the menu item for the geofence/google maps activity
2. Create a google maps activity. See the developer guide.
3. Capture user click event with onMapClick(LatLng).
4. Add a marker at the point of the click
5. Draw a circle around the marker to show the geofence radius.
6. Use SharedPreferences to store the latitude and longitude values. These values should be used to populate the map the next time the activity is shown.
7. Follow the instructions and code samples here for setting up the geofence.

Building the app for release

1. Create a keystore
2. Create a private key
3. Add the signing configuration to the build file for the app module. Example:

```
...
android {
```

```
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            storeFile file("myreleasekey.keystore")
            storePassword "password"
            keyAlias "MyReleaseKey"
            keyPassword "password"
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
...
```

See more information about preparing the app for release here:
http://developer.android.com/tools/publishing/preparing.html#publishing-configure