

Java 1D Report

Java 1D Report

Kronos

- Navigation and User Journey
- System Design and Implementation
- Design patterns
- Discussion and lessons learnt
- Conclusion
- Contributions of each team member
- References
- Appendix

Group 1-1 Members:

Shermine Chua Xin Min 1003626

Li Yuxuan 1003607

Keith Ng Zian Kai 1003515

Tiang Pei Yuan 1003323

Tan Dao Rong, Eugene 1003442

Chan Jie Lin 1003316

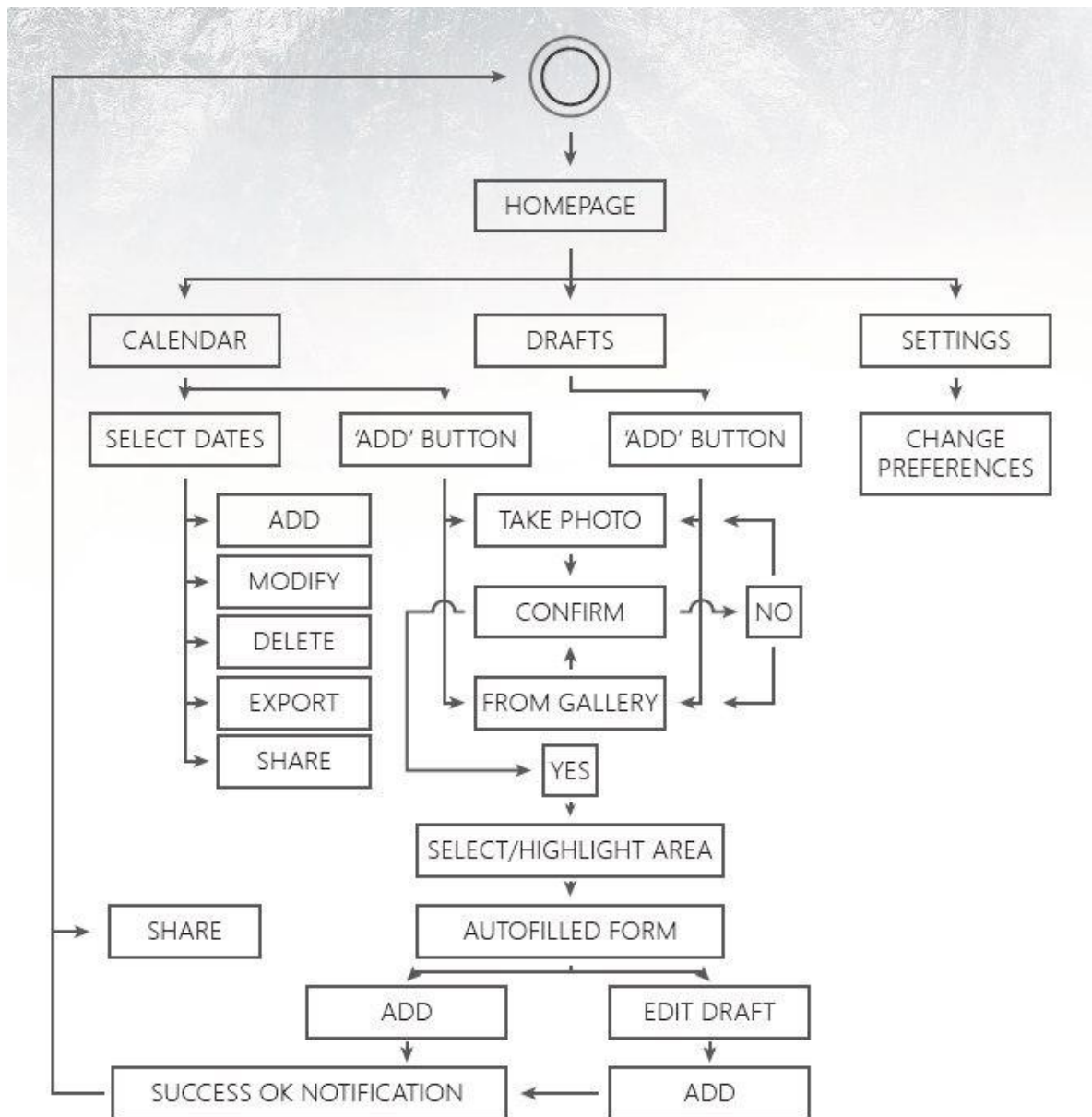


Problem: SUTD students are constantly exposed to event advertisements and notifications. These come in various forms ranging from hardcopy posters to emails. It is time-consuming and a chore to manually record down event details.

When it comes down to crunch time, such as when you are in a lift and you see an event poster that piques your interest, it is a challenge to check your availability and record the details in time before you exit the lift.

Hence, Kronos aims to alleviate the problem by providing a one-stop solution application, which integrates Optical Character Recognition (OCR) together with our own Calendar. Kronos will enable users to instantly register events of interest with a simple picture, and at one glance view all the events that have been registered on the calendar.

Navigation and User Journey



Home Page (Refer to Figure 1):

Our Calendar view. Calendar displays day-to-day events upon date selection.

'+' button allows for addition of calendar events via manual input, selection of photo from gallery or taking a picture. The respective buttons will re-direct users to the camera or gallery to select a picture, or a form that users can manually fill up the details of their events.

Cropping Page (Refer to Figure 2):

This page contains an ImageView which displays the image on which the OCR algorithm will be run on.

On the ImageView, users will be able to adjust a rectangular cropping box which is named Title. This is a visualisation of the image which will be cropped and processed by the OCR for automatic form input.

'✓' button allows user to confirm that they are satisfied with the image taken and the size of the rectangular box which they want to crop the image.

Forms Input Page (Refer to Figure 3):

This page contains several input fields which allows the users to modify the autofilled texts if any, or add if they are missing.

'Save as draft': Saves the draft into SharedPreferences which can be accessed and modified later on.

'Add to Calendar': Saves the information from the input fields into SharedPreferences and displays them in the in-built Calendar in Kronos

Drafts Page (Refer to Figure 4a & 4b):

In Figure 4a, there is a list view of the drafts that the users have previously saved. Users can select the draft to enter the draft view page shown in Figure 4b.

In Figure 4b, there is the 'edit' button at the top right corner of the page. Pressing the 'edit' button will redirect users to the form page where they previously left off and saved.

System Design and Implementation

In Kronos, we utilised OCR technology to aid in the recognising of event poster details in pictures. The OCR technology is able to process the image and identify details present in the image. These details are then processed before automatically entered into the calendar.

We also made use of communication and sharing technology such as Google Calendar to allow users to share their schedule with other people.

Kronos is a calendar app that contains a data pipeline which takes input from images of posters, processes the data into inputs to a calendar event, which will be stored into Kronos' calendar. This information can then be exported and shared using the users' preferred calendar, should users choose to do so, reducing the need for users to have to learn a completely new interface of sharing their schedule.

We applied object-oriented programming principles in our application. We have 2 modules, the application module and the cropping module. In each module, we have several classes created, each created with just one purpose, following the Single Responsibility Principle. The modularity of the code allowed us to quickly debug and modify contents without changing other parts of the codes that are simply consuming those utilities. For example, PhotoCaptured class is handling bitmap processing including resizing and rotation. GalleryUtils class is handling the conversion of date and time for the Gallery form page. GetOAuthToken class is specifically for getting the OAuth token for Google account to use the Google Cloud Vision service. These utility classes, when written in different .java files, made it easy for us to locate the root error and by modifying and correcting one mistake, it updates across all the other fragments and activities which are using these utilities.

In the application module for instance, we have these classes:

- PhotoCaptured
 - Contains methods which manage the photos captured by the users.
- Drafts Fragment
 - Manages the Drafts Page in the application.
- EditOCR Fragment
 - Manages the EditOCR Page in the application.
- Form
 - Manages the Form Page in the application.
- GalleryUtils
 - Contains methods which allow the conversion for DateTime objects.
- GetOAuthToken
 - Obtains the token for the use of the cloud OCR API.

- Home Fragment
 - Manages the Home Page in the application.
- View Animation
 - Manages the rotation of the image captured by the user.
- View Entry
 - Manages the View Entry Page in the application.
- Settings Fragment
 - Manages the Settings Page in the application.
- MainActivity
 - Manages permissions and the fragments to be shown at each instance.

Our app also utilises Shared Preferences as a method of enabling Data Persistence as taught in Android Lesson 2. We used Shared Preferences as the storage for our calendar events as well as a way to retrieve the information, as we do not require as much storage space as a database.

Logcat and Toast

Logcat and Toasts were used to help us debug our code.

Intents

Our application made use of explicit and implicit intents to pass data from one page to another.

Async tasks

Our application have several instances where async task is used to run processes in the background. For instance, our cropping module and our request to cloud Firebase API for OCR image to text conversion were done in the background, so the user will be able to view and modify other fields while waiting for the processes to finish.

The UI/UX is intuitive and easy to navigate as we carefully chose simplified icons to ensure that Kronos requires minimal explanation to utilise. It is friendly to all users and easy to use with a simple color scheme as well.

Design patterns

Singleton: PhotoCapture class

Due to the fact that we need to access the photo that the user captured for a particular calendar entry in various activities and fragments, we decided to use a Singleton design pattern to store the image path as a universally accessible attribute. Following that, we need to use some bitmap rotation functionality to make sure the orientation of the bitmap shown is always the right orientation. Hence, the image taken would need to be processed every time the user takes a photo.

Considering all these requirements, a Singleton class that handles these functionalities would be what we need.

With reference to the snippet of code for the photoCaptured class, this Singleton class will handle the processing of the bitmap as well as generate a unique image file name for each photo taken. The image path of the photo taken could be directly accessed using the `getImgPath()` function under the Singleton class.

Discussion and lessons learnt

What differentiates us from other calendar applications is the added functionality of OCR on top of the normal calendar functionalities itself.

CalendarView Widget

Initially, our group used the built-in CalendarView. It allowed us to select dates but not store information to the individual dates itself. However, customisation that could be done was limited to just colour schemes. Therefore, to get a more ideal calendar, we decided to use third-party calendars. After much consideration, we implemented the Material Calendar View by Applandeo. This calendar view had more customisations, allowing us to add an icon on the dates itself to signify that there was an event on the particular date. This is a very important UI element as at one glance of the calendar, the user will be able to tell whether they have any scheduled events on that day.

Data Storage

After looking at the different types of data storage available, from online databases such as Firebase as well as offline databases, we have decided to use SharedPreferences as our data storage as:

- 1) Users are unable to access the data and hence, they will be unable to modify and corrupt the data.
- 2) Data stored will be in the users' local directory which means that there will be less strain and overhead time compared to have cloud storage, as every user will have their own data.
- 3) Offline access is also enabled due to the fact that we do not require to GET from a cloud storage.

Standardisation of storage data is important so that the data passed from one part of the application to another can be immediately used, and the pages can be worked on concurrently.

Hence, we created a standardization document which everyone can refer to for formatting of storage data. This documentation can be found here:

<https://github.com/Etternal/java-1D/blob/master/SharePreferences%20Format%20Standard.pdf>

Version Control

Version control is important in software projects. We have learnt how to effectively use Github to do version control to track modifications of different parts of the code. If a mistake is made, we will be able to revert to a previous version to work on. In this project, we have set the master branch to always contain the most updated stable version of the application, and work on branches to include new features. When the new features are stable, they will be merged back into the master branch. This allows for more efficient workflow as everyone can work on different parts of the same application on a development branch concurrently.

Fragments

To better facilitate the switch between pages, fragments were used. In the original design of the app, we had multiple pages - the camera, the calendar, the drafts and settings. In addition, we wanted to have a bottom navigation bar to improve users' experience in the transitions between pages. Fragments were needed to work with the bottom navigation bar, hence, we decided to use fragments for the framework of the application. The fragments worked differently from the activities and its data can be managed with the use of intents. However, we ran into difficulties during the integration of the pages into the main application as not everyone is familiar with fragments, much of the code from the individual parts had to be modified to accommodate the use of fragments.

One example can be seen with the following:

When using activities and intents, one can call out the id from the .xml file by simply using the function below:

```
eg_button = findViewById(R.id.eg_button);
```

However, when using fragments, to call out a specific id, one must use the following:

```
view view = inflater.inflate(R.layout.fragment, container, false);
eg_button = view.findViewById(R.id.eg_button);
```

Fragments are simply views on the activity that are changed upon the changing of fragments. To use the calling of the id, the layout of the desired view must first be inflated to replace the view of the current view. There after, the id can be called out using the same `findViewById()` function.

Conclusion

In summary, this application is effective in allowing people to efficiently store events into their calendar through OCR technology, which can be exported to their external calendar applications. The simple and clean interface also makes the application easy to use, allowing users to have a smooth experience. Our application has the basic functionality of calendars to add and display events on the calendar with the added functionality of allowing users to take a picture, crop out essential portions and run it through the OCR API to convert image to text, to hasten the process of creating events. In addition, users will be able to take advantage of the sharing or exporting functionality in our calendar to share their calendar with others, or export events their preferred calendar applications.

Contributions of each team member

Pei Yuan - Calendar framework and functionalities

Shermine - Calendar functionalities and Application Framework

Yuxuan - Poster Design, Cloud OCR, Editing Page, Sharing/Exporting Function, Final integration

Eugene - User Interface, Application Framework

Wang Wei - Drafts Page

Keith - Local OCR, Pulling images from gallery

Jie Lin - Cropping Page, Notifications

References

www.stackoverflow.com :)

Google Vision API - for cloud OCR

Firebase ML kit - for offline OCR

Modified Cropping module from Vishal Thakkar - https://www.youtube.com/watch?v=nc_Zmkf16kM&feature=youtu.be

Calendar Library - <https://github.com/Applandeo/Material-Calendar-View>

Profs Ngai-Man and Norman - For teaching us 50.001 :)

TA that never fails to show up and always sits at the back of the class

Appendix

```
class photoCaptured {
    private static final photoCaptured ourInstance = new photoCaptured();
    private String imgPath;
    private Context context;
    static photoCaptured getInstance() {
        return ourInstance;
    }
    private photoCaptured() {}
    public String getImgPath(){
```

```

        return imgPath;
    }
    public void setImgPath(String path){
        imgPath = path;
    }
    public void setContext(Context c){context=c;}
    public File createImageFile() throws IOException {
        // Create an image file name
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
        String imageFileName = "KRONOS_" + timeStamp;
        File rootPath = context.getExternalFilesDir("Pictures");
        File image = new
File(rootPath.getAbsolutePath()+"/"+imageFileName+".jpg");
        // Save a file: path for use with ACTION_VIEW intents
        imgPath = image.getAbsolutePath();
        return image;
    }
    private Bitmap rotateImage(Bitmap source, float angle) {
        Matrix matrix = new Matrix();
        matrix.postRotate(angle);
        return Bitmap.createBitmap(source, 0, 0, source.getWidth(),
source.getHeight(),
            matrix, true);
    }
    public Bitmap processThumbnail(Bitmap myBitmap, String path) throws
IOException {
        ExifInterface ei = new ExifInterface(path);
        int orientation = ei.getAttributeInt(ExifInterface.TAG_ORIENTATION,
            ExifInterface.ORIENTATION_UNDEFINED);
        Bitmap rotatedBitmap;
        switch(orientation) {
            case ExifInterface.ORIENTATION_ROTATE_90:
                rotatedBitmap = this.rotateImage(myBitmap, 90);
                break;
            case ExifInterface.ORIENTATION_ROTATE_180:
                rotatedBitmap = this.rotateImage(myBitmap, 180);
                break;
            case ExifInterface.ORIENTATION_ROTATE_270:
                rotatedBitmap = this.rotateImage(myBitmap, 270);
                break;
            case ExifInterface.ORIENTATION_NORMAL:
            default:
                rotatedBitmap = myBitmap;
        }
        return rotatedBitmap;
    }
}

```

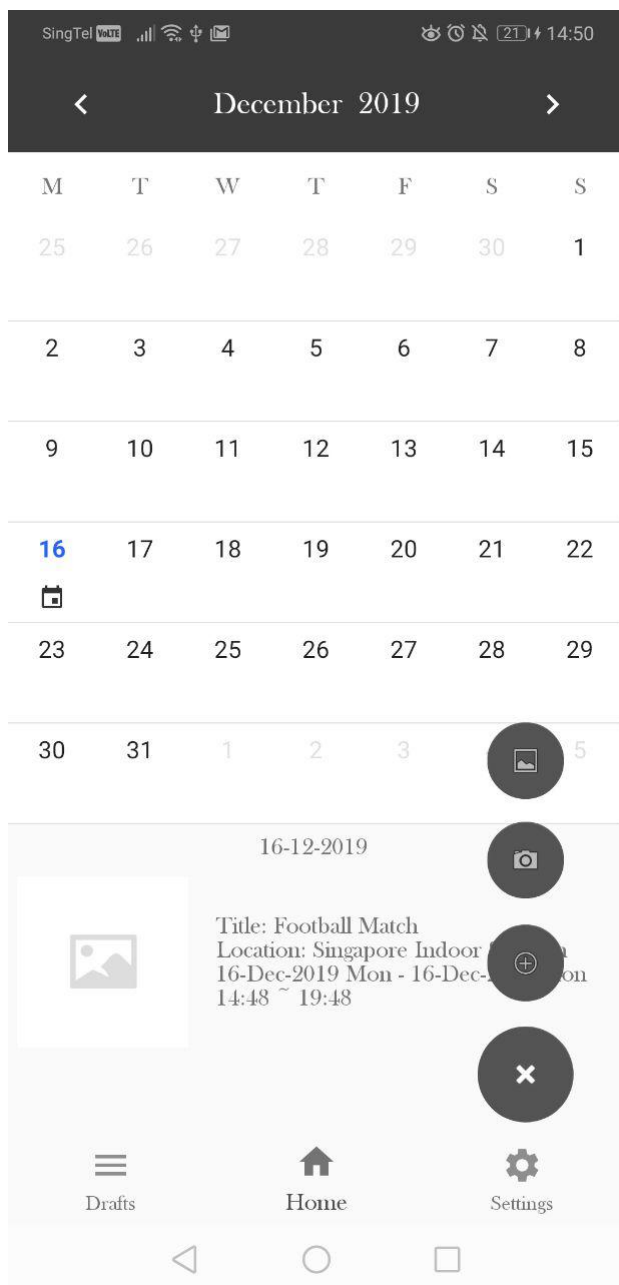


Figure 1: Home Page with event

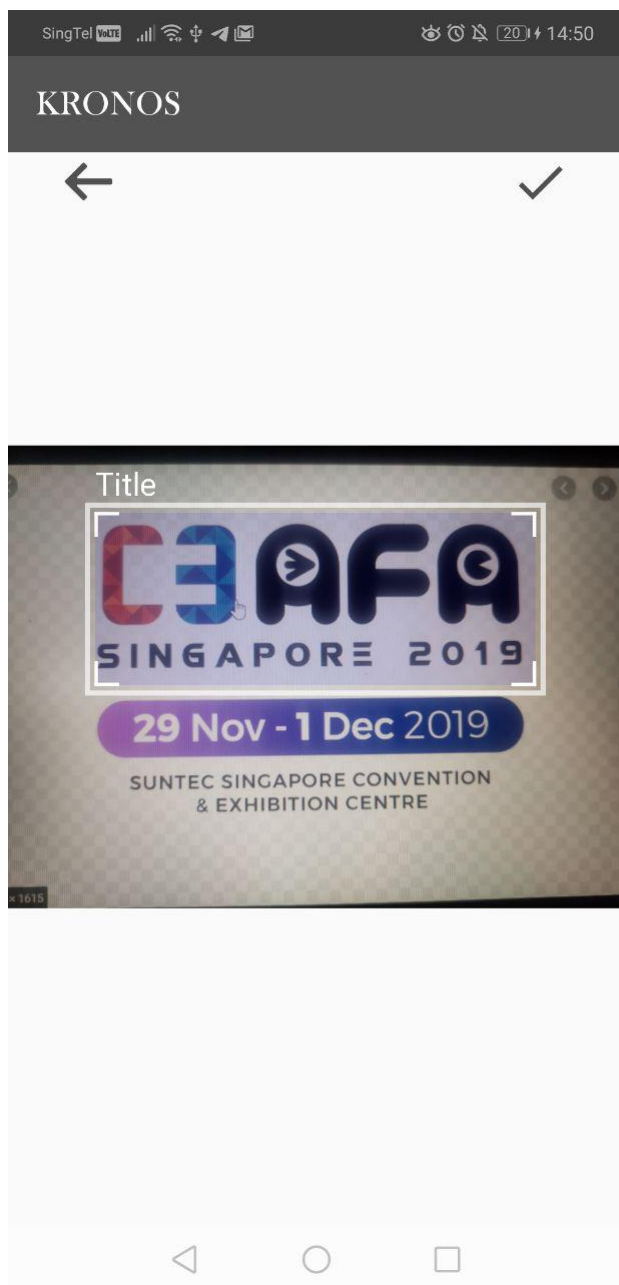



Figure 2: Cropping Page

8:32





Event Title

From

10-Dec-2019 Tue

08:31

To

10-Dec-2019 Tue

12:30

Input location...

Input description

SAVE AS DRAFT

ADD TO CALENDAR

Figure 3: Forms Input Page

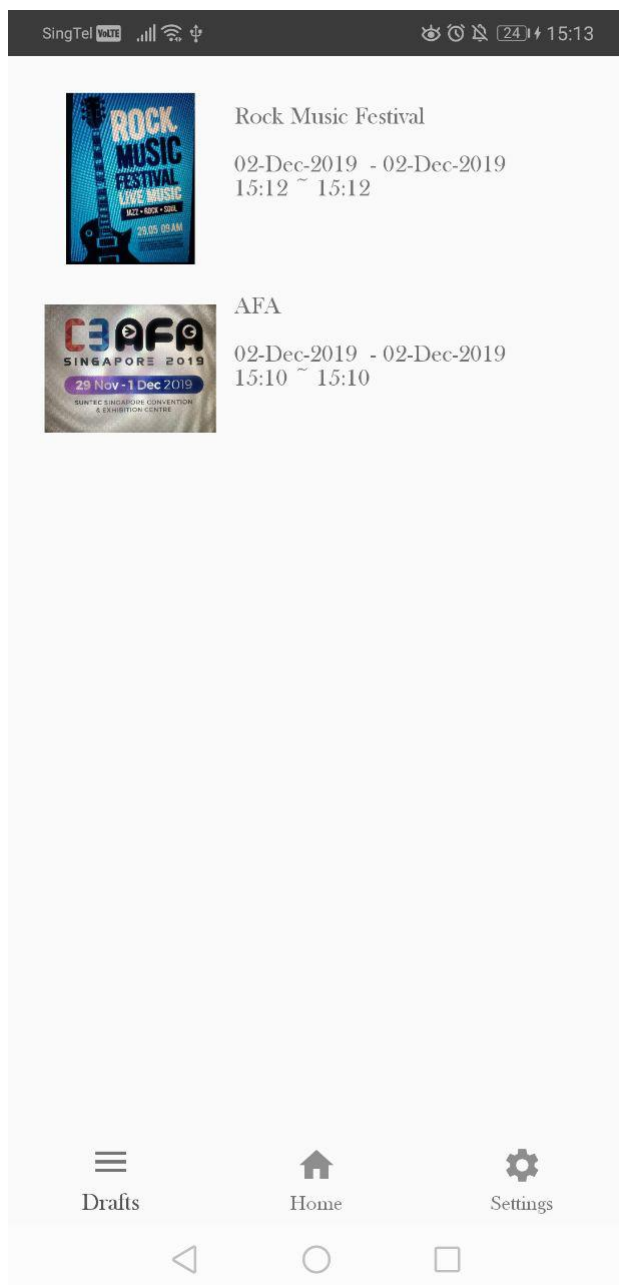


Figure 4a: Drafts List Page

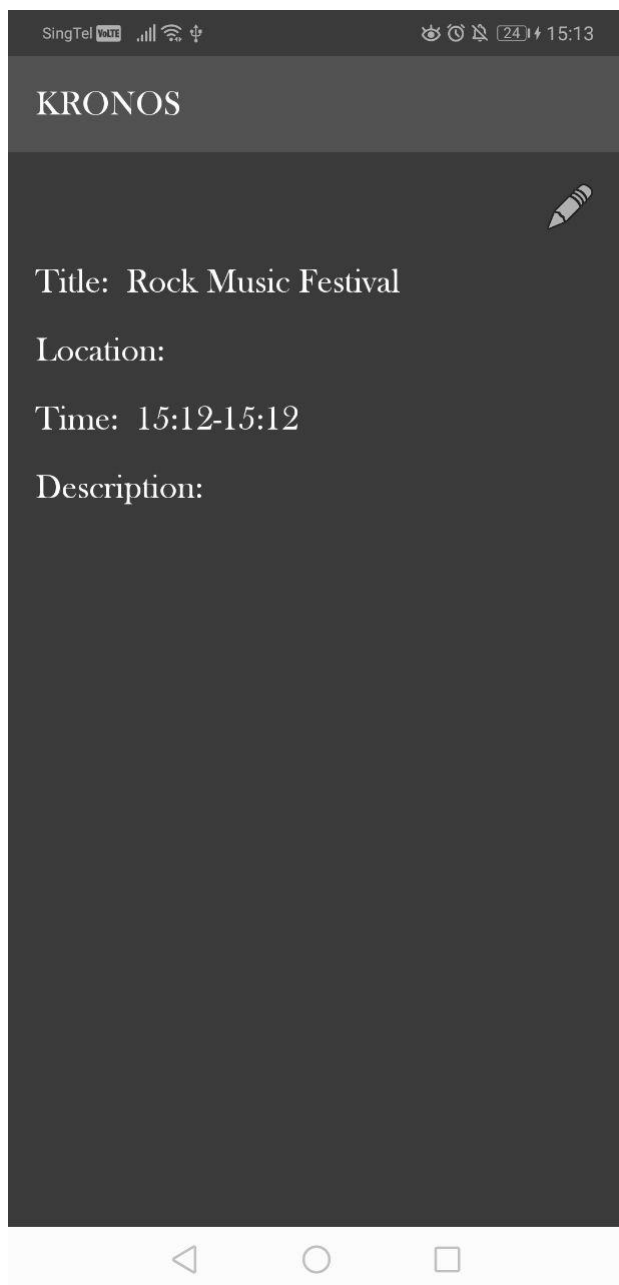


Figure 4b: Draft View Page