GitHub Username: dkaushik95

Anunciar

The Announcement App

Table of contents

1. Description
2. Intended User
3. Features
4. User Interface Mocks
    a. Screen 1
    b. Screen 2
    c. Screen 3
    d. Screen 4
    e. Screen 5
5. Key Considerations
    a. How will your app handle data persistence?
    b. Describe any corner cases in the UX.
    c. Describe any libraries you'll be using and share your reasoning for including them.
    d. Describe how you will implement Google Play Services.
6. Next Steps: Required Tasks
    a. Task 1: Project Setup
    b. Task 2: Implement UI for Each Activity and Fragment
    c. Task 3: Final touch up for the app
    d. Task 4: SR flavor for the app

Description

In a group chat system, we often lose out important information which act as an announcement. With the ability for everyone to react to a particular messages, the important message is hidden from the ones who do not check the chat so often. Reading through a plethora of messages is a pain in order to find relevant information.

To remove this, I introduce you to Anunciar, a one to many communication application, where the heads of groups has the right to post important information and every other member has the ability to view it. If there is a deadline, you can add that information to a calendar so that you get the information you need, when you need it.

We are also trying to solve the problem of a unified calendar so that the time table and its changes can be viewed from one place.

Future scope of the project would be using prediction to help the user what to do next when an announcement arrives.

Suppose you have an announcement to go to an extra class on a particular date at a particular time, prediction can help you create a reminder without you doing the hard work.

Intended User

There are two class of intended users, one is the Class representative, who will use the application to post new announcement.
Another class of users is the Students, who will receive notifications about the new announcement created by the class representative.

Features

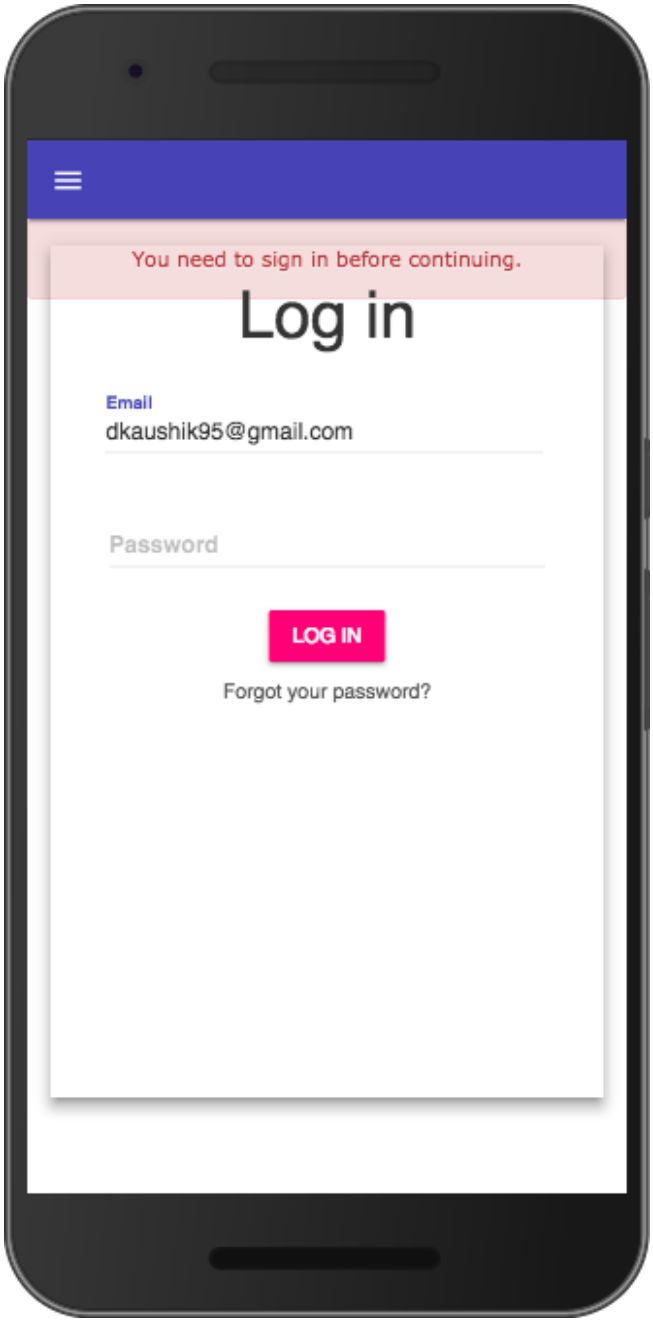
List the main features of your app. For example:

- Authenticated and encrypted by the device authentication
- New informations comes from the custom built API
- Saves the old announcements for the users to see when offline. A limit is kept to improve disk usage.
- Notifications are called everytime a new announcement is made.
- New users are created only by the class representatives.
- Deadline will help adding important events to the calendar using the calendar API.

User Interface Mocks

As I am learning progressive web apps, the mocks are from there and the android app will look similar.

Screen 1

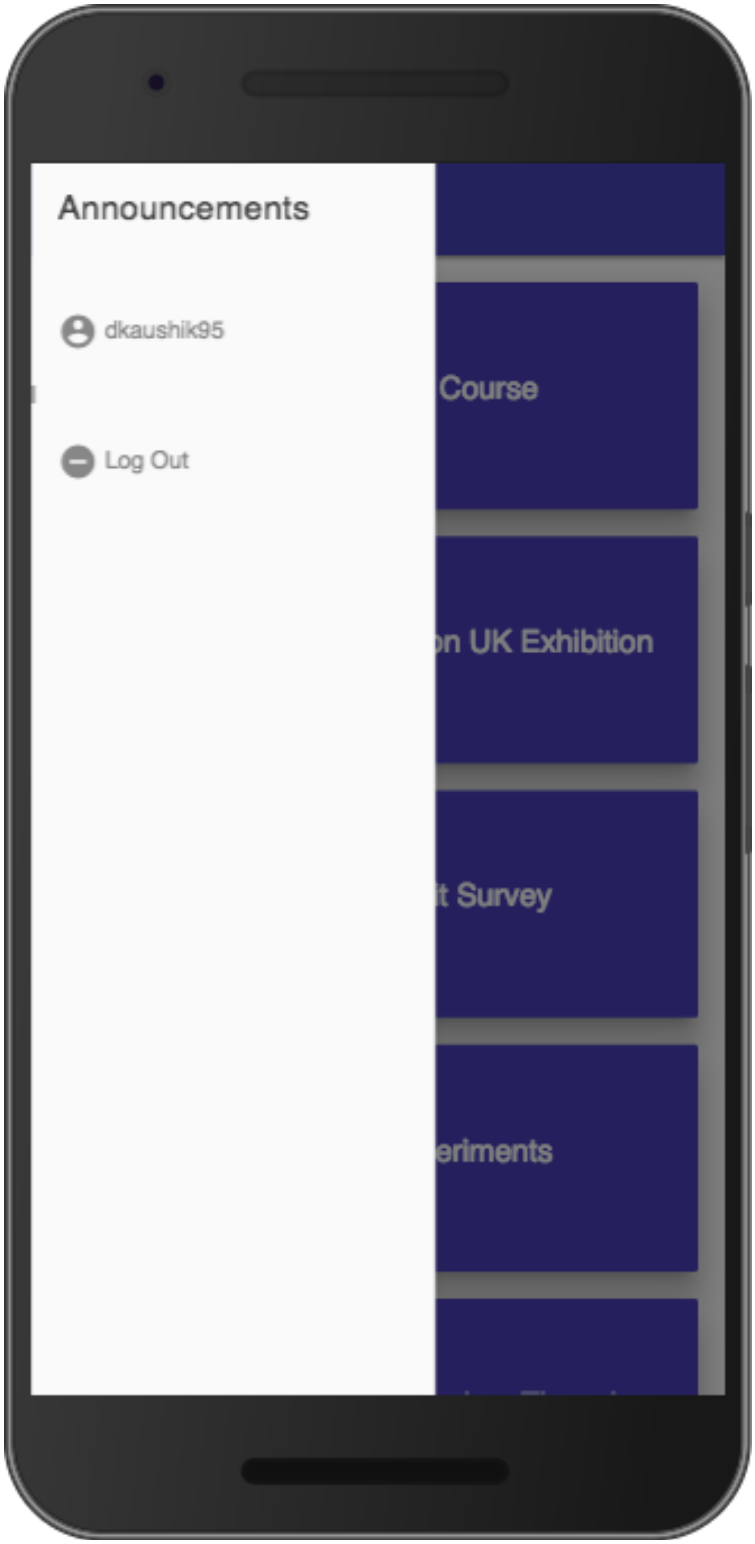


This is the login screen. Login will happen using the Devise gem in the rails backend.

Screen 2

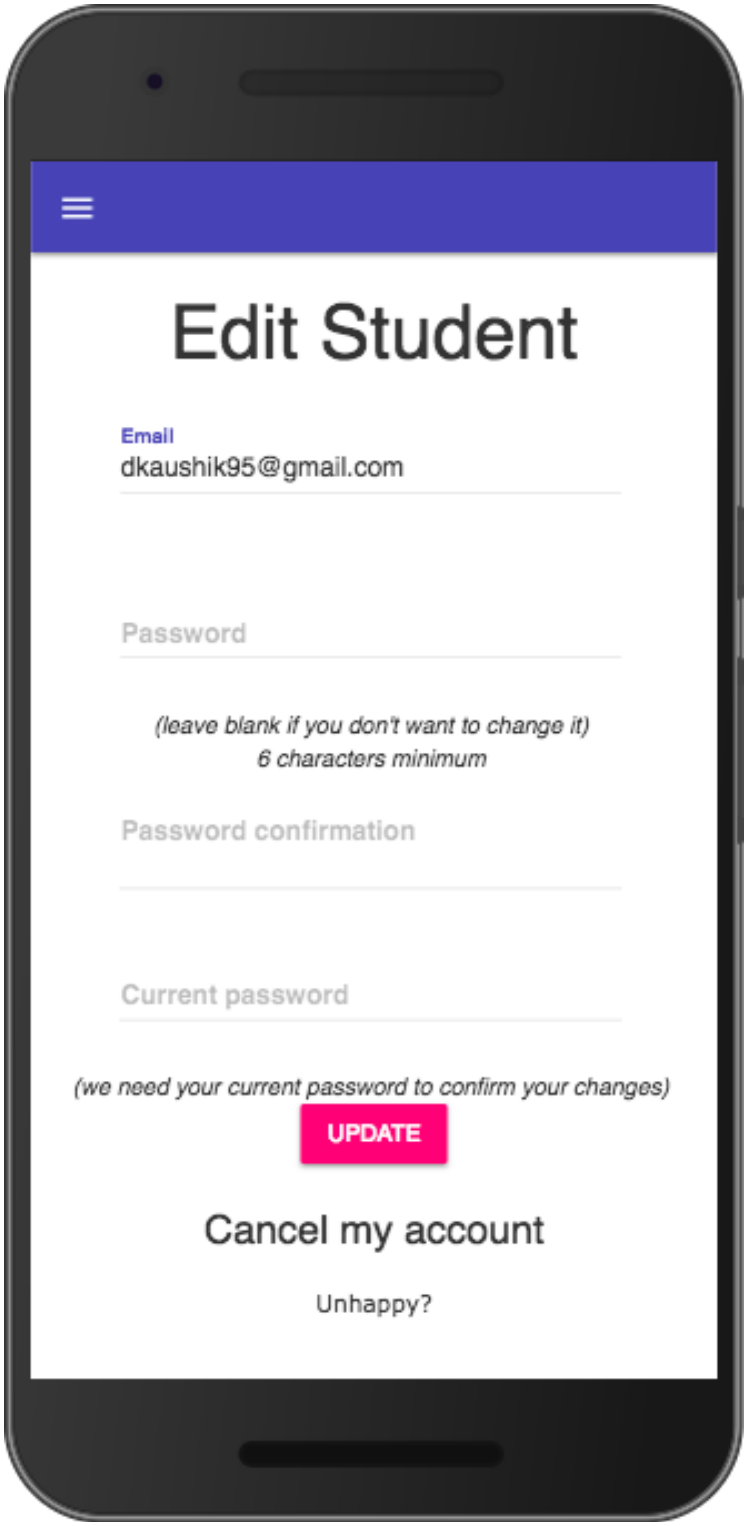
After the user logs in, the user will be presented with cards of announcement. These announcement are brought from the rails backend API and stored using content providers for offline use.

Screen 3



The user can edit his/her account and log out from the app if they prefer.

Screen 4



This is how they can edit the account.

Screen 5

Once they click an announcement, they will see this screen where the details of the announcement. If they click on the date(if available) they can save it to the calendar using the Calendar intent and the details will be passed as parameters.

Key Considerations

How will your app handle data persistence?

I already have a database set up online which the API will use. Once we interface with the API, we will get new data which we will store on local phone by building a Content Provider. The user information will be stored in the shared preference as there is only one key value pair.

Describe any corner cases in the UX.

I am trying to make custom transitions and using the material design principles for making the UI consistent and ideal for the users to use. I will try to implement the paper animation and shared transition for the announcement.

Describe any libraries you'll be using and share your reasoning for including them.

Server side:

- Rails GCM gem: for push notifications initiated by the server.
- Rails JSON parser to convert SQL queries to json responses and handle

Client side:

- AutoProvider: For creating content providers easily.
- AsyncTaskScheduler: For background processing of data by API calls and all other functions which doesn't work in UI tasks.
- FireCrasher: To handle the Uncaught Exceptions in your android application and helps to recover without exiting from the application.
- Welcome: to create custom splash screens.
- Jus: flexible and easy HTTP/REST client library for Java and Android.
- lg-json-parser: to parse JSON into content providers
- NightOwl: for switching day/night mode on Android.

Describe how you will implement Google Play Services.

My project will use Google Play services to set up GCM from the rails server to the android application so that when the Student representatives update something to the database, all the users should be notified about the change.

- Google Account Login: For login of the students
- Google Cloud Messaging: For push notifications

Next Steps: Required Tasks

Task 1: Project Setup

- The backend
    - Build the rails application and make it slim for API only usage so that I only have the Model (for database) and controller (for API request rules) and not the Views (HTML pages).
    - Create the database of Student representatives, Students and Announcements.
    - Use the controller to define the API requests and responses. They will also contain all the rules to authenticate before showing and handling all the error cases.
    - Test it and deploy the backend in Heroku.
- Front-end (The Android App)
    - Setup the android app and add all the libraries and resolve compatibility conflicts.
    - Make the Appropriate skeleton for all the activities that will be present
        - Login
        - Show All Announcements
        - Show Selected Announcement
    - Connecting the App to Google Account Login and wiring it up with the server.
    - Testing the API interface with Android and show simple JSON results to each activity.

Task 2: Implement UI for Each Activity and Fragment

- Login
    - Create the Login Interface in the XMLs
    - Create a shared preference for login information
    - Test out the login system and make it the first screen if the user is not logged in.
- Show All Announcements
    - Use the JSON Parser to show all Announcements in respective cards
    - Create content providers to save the incoming announcement.
    - Make this the first screen when the user is logged in.
    - Set up the GCM to get a notification when a new announcement is posted.
    - Create and endpoint intent to handle users taps to any announcement.
- Show selected Announcement
    - Use the intent to get the ID of the selected announcement
    - Show the information in detail in this view and create a Calendar Provider for adding a deadline oriented announcement to Google Calendar.

Task 3: Final touch up for the app

- Creating tests and handle cases when the API is not responding.
- Adding transitions and splash screen for the app.
- Adding material design for the app.

- Creating a widget for the announcement list.
- Adding content descriptions and RTL support for greater accessibility.
- Making a watch app to read announcement on the go.

Task 4: SR flavor for the app

- Adding a build variant for the Student Representative to use the app.
- The app will contain a similar interface and use, except the user will now have a FAB to create a new announcement.
- Using Firebase, once the announcement is created, the app will send a notification to all other users.