SOFTWARE DESIGN SPECIFICATION

1. INTRODUCTION

1.1) Purpose

This document intends to provide elaborative details and data flow for the software development and design process. The document lays down the design flow of how the software is being made through the use of class diagram, use case diagram and sequence diagram along with other required details.

1.2) Scope

The product being talked about in this document is a fest management android application, called as Festeve. The main purpose of this application is to build up a common space for both the fest organizers and the viewers or participants. The application is being developed due to an unorganized and incomplete details, in current apps, about a certain fest or its events to the students or audience. Therefore, there is a need for such a platform where the students can quickly get an access to details regarding these events going on around him/her. This way the problems faced by the students to actually dig out informations related to fests will just be cured, right in their hands, by downloading this simple and useful application. The fest organisers will also be benefited by the system as it would notify them instantly regarding new registrations, visitors, withdrawals etc and they will be able to manage accordingly.

1.3) Definitions, Acronyms, Abbreviations

- Admin : Fest Administrator (only 1 from a college)
- App : Application
- Back-end : Logic that makes a mere icon or system to work
- Fest-Calendar : Calendar showing dates reserved for various fests.
- Festeve : Fest-events
- Fest-Feeds : Coming up stories about fests in various colleges
- GUI : Graphical User Interfaces

1.4) References

- https://en.wikipedia.org/wiki/Software_architecture
- https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
- https://en.wikipedia.org/wiki/Component-based_software_engineering
- http://research.cs.queensu.ca/home/ahmed/home/teaching/CISC322/F10/files/documentingArchitecture.pdf
- http://www.cs.utah.edu/~jamesj/ayb2005/docs/SDS_v2.htm#2.0

1.5) Overview of documents

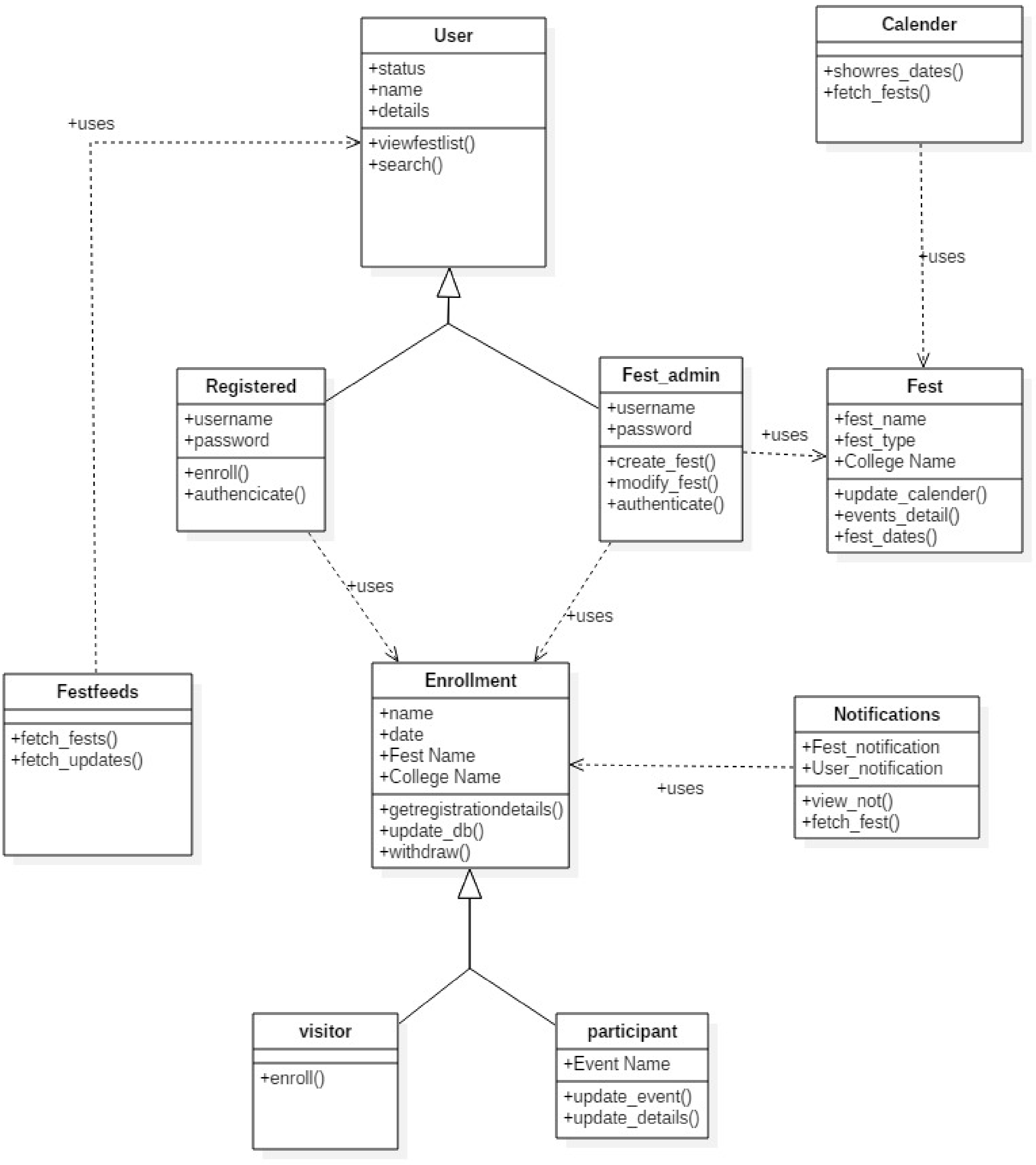
The document puts forward a description regarding application’s major components and it’s architecture, along with the specifications on the interaction between the user and the system.

- Section 1 provides the description of our project as well as outlines the importance of this documentation.
- Section 2 describes system architecture with the help of a component based architecture design diagram and a class diagram to point out the relationships and structure. This section also forcuses on UI design of the system.
- Section 3 gives proper detailing of the components or subsystems of the application which was briefly outlined by the architecture design diagram.
- Section 4 lays out the core decision of reusability of some work and marks its relationship with the other tools or products.
- Section 5 points out the major decisions we had to take while designing our software and explains the reasons behind undertaking and implementing those decisions.
- Section 6 is the pseudocode section which provides pseudocode to clarify the desired operation of certain components.

2) SYSTEM ARCHITECTURE DESCRIPTION

2.1) Overview of modules / components

Class Diagram



The flow of software design for our project is derived using class based architecture design wherein we have assigned a fully functioned subsystem having one main function and similar structure, appearance or coding structure into one component. Though there exists some dependency among the components but the overall functionality which each component performs is what distinguishes one another.
So here, we have summed all our features and functions of our app into different classes :

- User
  - Unregistered
  - Registered
  - Fest_admin

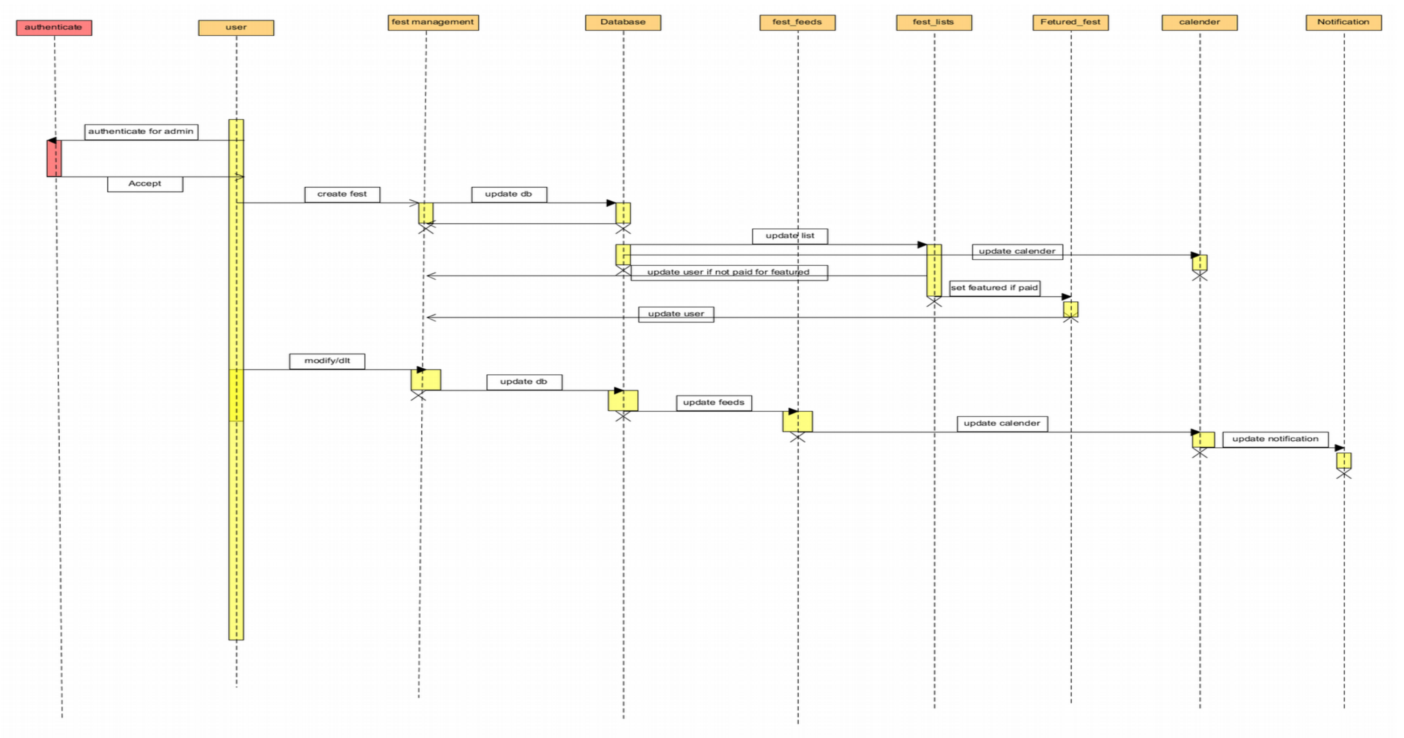
- Fest Registration
  - Visitor
  - Participant

- Fest Management – Create, modify or delete a fest or its details
- Fest-Feeds – Shows up latest changes in fests and fest calendar
- Calendar – Shows booked dates of fests
- Notifications – Notifies for updates in the registered fests

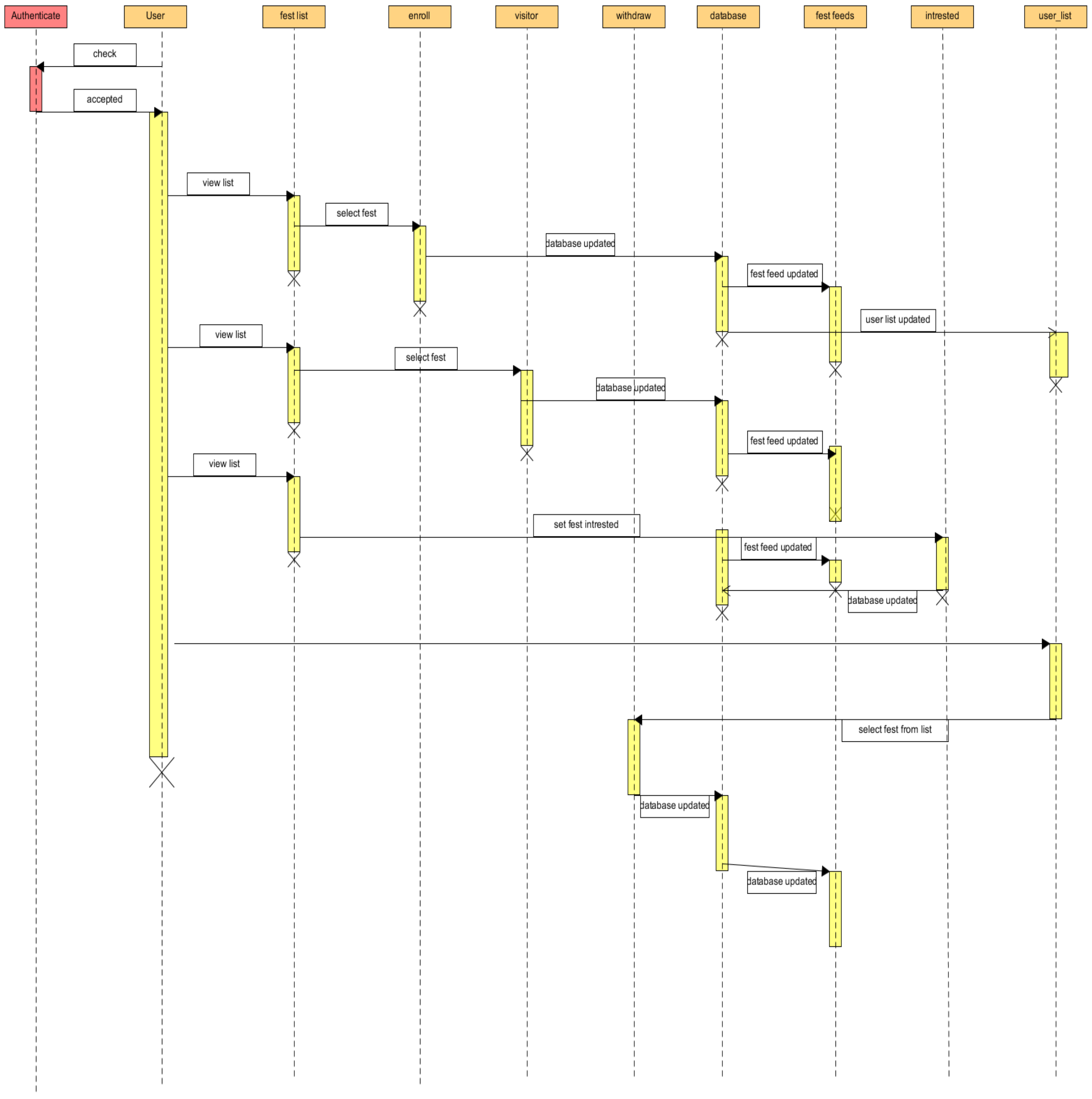
2.2) Structure and relationships

The structure of the class defines the cohesion of the system and the relationships among those classes determine the coupling in the system. So in order to achieve efficient system, we have come up with designing class in such a way as to have high cohesion and low coupling.

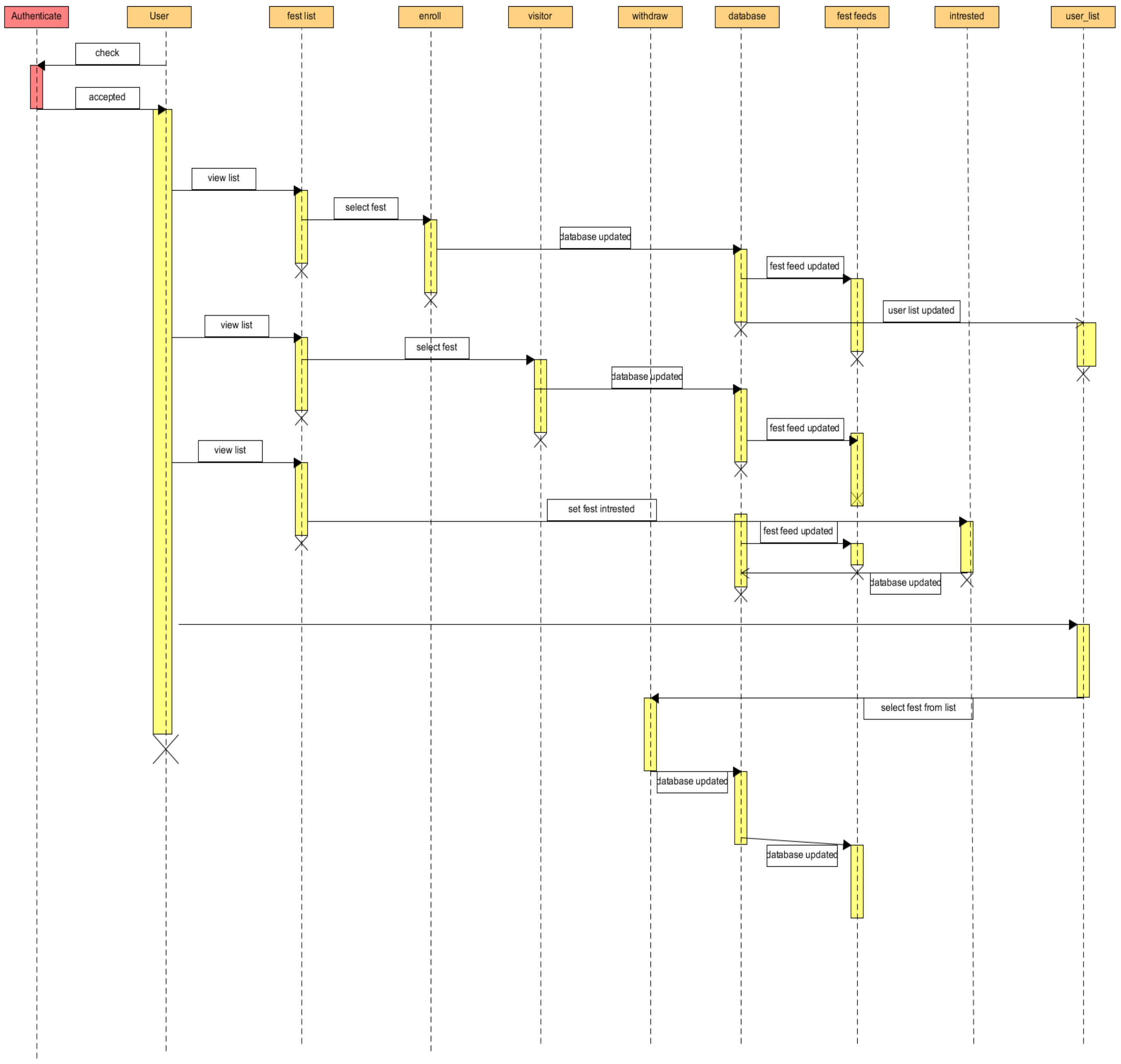
2.2.1) Sequence Diagram For Fest Admin



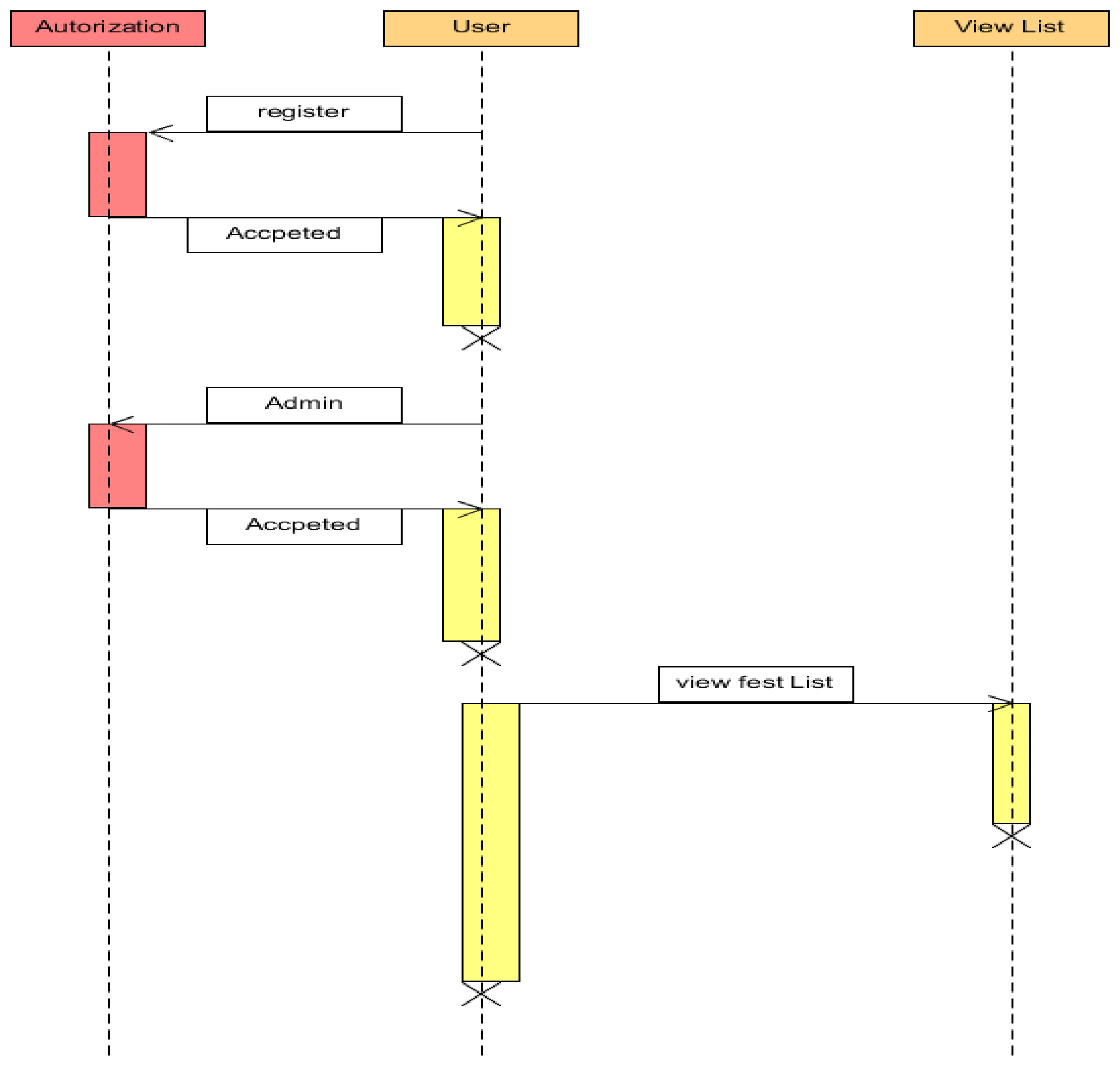
2.2.2) Sequence Diagram For Registered User



2.2.3) Sequence Diagram For Selecting Fest



2.2.4) Sequence Diagram For Unregistered User



2.3) User interface issues

The application carries a rather simple Graphical User Interface (GUI) which makes it more deliverable and compact in nature. The User Interface in terms of features and utility varies with the essence of the kind of user interacting with our application.

The first basic app page which will be the same for every type of user, is the looking of the application while the interaction may be different.

Basic GUI components :

1) ***Header*** **-** The header (blue in colour) contains the very own app name – Festeve, on the left hand side and an icon for menu on the right hand side. This menu opens up a side activity space having various features and setting options. This also displays some important

features which should not go unnoticed like notifications to registered users in the form of an icon on the right hand side at the top.

2) ***Tab*** – This consist of a search bar with a search icon on it along with query suggestions written on it to help the users to know about what to type in or more specifically what can they search for.

3) ***Content Area*** – It is divided into two parts :-

- The top portion showcases a featured event or fest in the form of a rectangular block.
- The remaining portion which is scrollable in nature is the Fest-Feeds (similar in nature to news feeds) where each information about fests anywhere in the colleges will be shown.

4) ***Icons*** – Dedicated icons are designed for catchy or specialized functions. Our app has icons for search option, notification system, menu option and fest calendar.

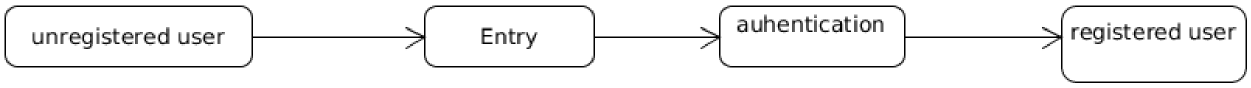
3. DETAILED DESCRIPTION OF COMPONENTS

3.1) User Class

Identification	User
Type	A class
Purpose	Defines the user base of the app. To differentiate between the type of users and classify each of them with their functions. This is done by using status property or attribute of the class
Function	The class performs functions like : • signup() - lets user to sign up • view_calendar() - lets user to view fest calendar • search() - lets users to search for a fest

3.1.1) Registered User Class

State Diagram : For Sign Up

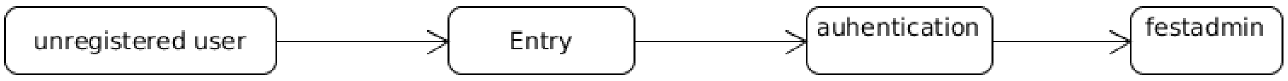


Identification	Registered User
Type	Subclass
Purpose	To provide increased functionalities including notification system and enrollment for fests.
Function	The class performs functions like : • enroll() - to enroll for a fest

	• withdraw() - to withdraw from a fest

3.1.2) Fest Admin Class

State Diagram : For sign Up



Identification	Fest Admin User
Type	Subclass
Purpose	To let colleges create fests and modify them.
Function	The class performs functions like : • create_fest() - lets user to create a fest • modify() - lets user to modify • delete() - lets user to delete a fest • check_reg() - lets user to check registrations

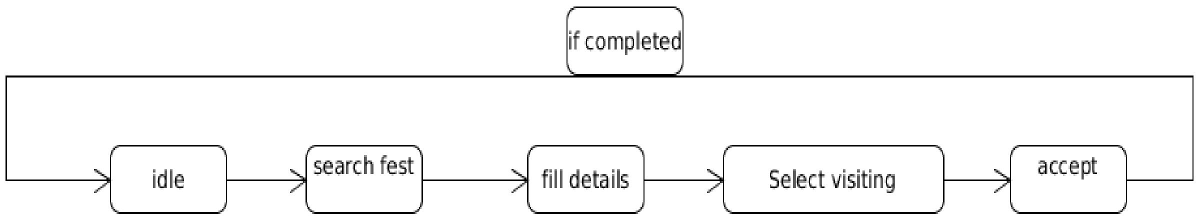
3.2) Enrollment Class

This class comprises of two sub class which are Participant class and Visitor class .
The Participant class deals with the users who want to enroll in an event of a fest whereas the Visitor class deals with the users who wants to attend the fest as audience**.**

Identification	Registration
Type	Super Class
Purpose	This class comprises of two sub class which are Participant class and Visitor class . The Participant class deals with the users who want to enroll in an event of a fest whereas the Visitor class deals with the users who wants to attend the fest as audience
Function	The class performs functions like : • fill() - lets user fill details • preview() - lets user preview filled details • save() - lets user save the registration

3.2.1) Visitor Class

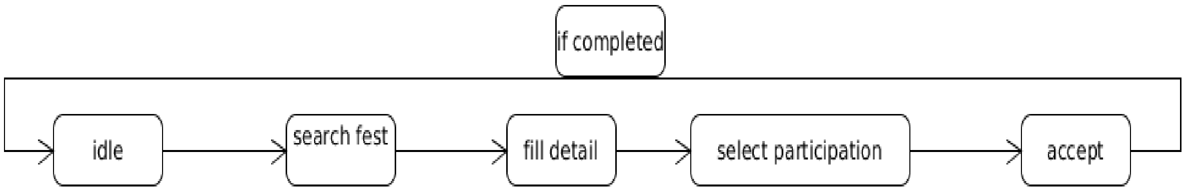
State Diagram : Visitor Enrollment



Identification	Visitor
Type	Sub class
Purpose	Visitor class deals with the users who wants to attend the fest as audience
Function	The class performs functions like : • Lets user to register as a visitor to a fest

3.2.2) Participant Class

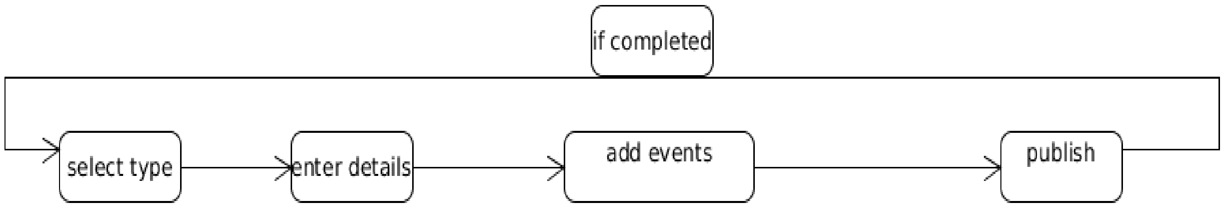
State Diagram 1 : Enrollment as participant



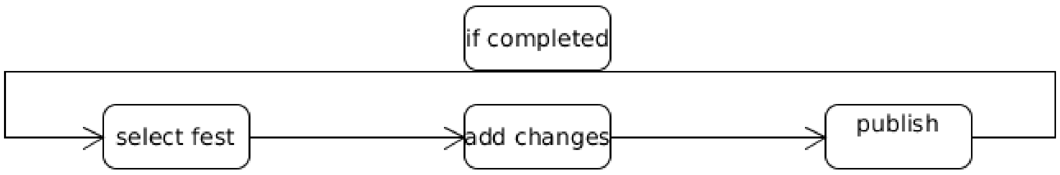
Identification	Participant
Type	Sub class
Purpose	The Participant class deals with the users who want to enroll in an event of a fest
Function	The class performs functions like : • Lets user enroll as participant to fest.

3.3) Fest – Management

3.3.1) State Diagram 1: Create Fest



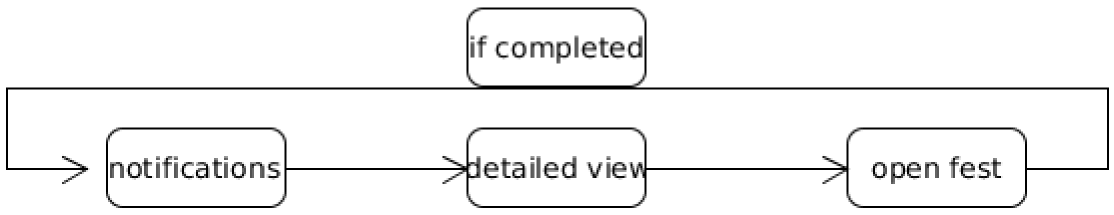
3.3.2) State Diagram 2: Modify Fest



Identification	Fest Management Class
Type	A Class
Purpose	This Class takes care of creating new fest and updating the related classes of the app. This class also manages the events , their dates and details
Function	The class performs functions like : • update_calender() – Update the date in the calender app so that the user can use the calender to view the fest dates in more friendly way • event_details() - fetch the event details and update the Database • fest_dates()- update the dates of events in database

3.4) Notification

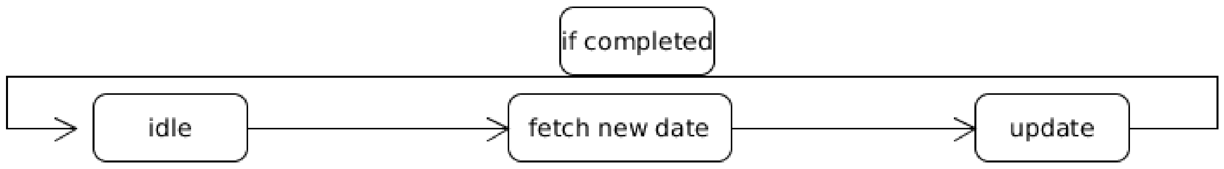
State Diagram: Notification



Identification	Notification
Type	A class
Purpose	This class manages the notification functionality of our app. This functionality gives the real time changes made in the interested fest by the user .
Function	The class performs functions like : • fetchfestst() - opens the app and fetch the fest when user tap the notification • view_notif() - shows the notification in more elaborated way

3.5) Calender

3.5.1) State Diagram: Calender



Identification	Calender
Type	A class
Purpose	This class auomatically update the calender in our app , each fest date will be marked in our app calender
Function	The class performs functions like : • fetchfestst() - opens the app and fetch the fest when user tap the notification • show_dates() - shows the fest dates in calender app

3.6) Festfeeds

Identification	Fest feeds
Type	A class
Purpose	This class updates the feeds section related with activities regarding changes in fest
Function	The class performs functions like : • fetchfestst() - opens the app and fetch the fest when user tap the notification • fetch_updates() - update the feeds list

4. <u>Reuse and Relationships to other Products</u>

The project that we are doing is to make a common platform to organise , and promote a fest the modules which we are using are new and can be used in future updates as the modules are desinged intelligently . Through app the user can create a fest , enroll in an event , track the current changes and updates in their interested fest and events . In future the app can be used by the sponsors to sponsor fest

5. <u>Design,Decision and Tradeoffs</u>

Certain design decisions regarding the working of app were taken considering the time factor. Main activities were given preferences like search activity,notification system(basic), fest_calendar, fest-feeds and registration and withdrawal and these were kept by taking an account of survey we conducted where these features were the most demanded. Many other features like linking Google Map services, sponsor details and advanced notification system will be added once the previous lot of activities are completed and will be available in upgrade version. These features will need more time to accomplish and so for the time being, we are keeping them aside but not discarding them completely.

6. Pseudocode for Components

1. User

```
        private void registerUser(){
         //getting email and password from edit texts
         String email = editTextEmail.getText().toString().trim();
         String password  = editTextPassword.getText().toString().trim();
         //checking if email and passwords are empty
         if(TextUtils.isEmpty(email)){
              Toast.makeText(this,"Please enter email",Toast.LENGTH_LONG).show();
              return;
         }
         if(TextUtils.isEmpty(password)){
              Toast.makeText(this,"Please enter password",Toast.LENGTH_LONG).show();
              return;
         }
         //if the email and password are not empty
         //displaying a progress dialog
         progressDialog.setMessage("Registering Please Wait...");
         progressDialog.show();
         //creating a new user
         firebaseAuth.createUserWithEmailAndPassword(email, password)
                   .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
                             //checking if success
                             if(task.isSuccessful()){
                                  //display some message here
                                  Toast.makeText(LoginActivity.this,"Successfully
registered",Toast.LENGTH_LONG).show();
                             }else{
                                  //display some message here
                                  Toast.makeText(LoginActivity.this,"Registration
Error",Toast.LENGTH_LONG).show();
                             }
                             progressDialog.dismiss();
                        }
                   });
     }
     @Override
     public void onClick(View view) {
          //calling register method on click
          registerUser();
     }
}
```

-

2. Registration

```
       enroll.setOnClickListener(new View.OnClickListener() {
     public  void onClick(View v)
     {
          final Firebase ref = new Firebase(Config.FIREBASE_URL);      //firebase url
          //Getting values to store
```

```
        String usr_name = user_name.getText().toString().trim();
        String usr_email = user_email.getText().toString().trim();
        String usr_number = user_number.getText().toString().trim();
        String usr_city = user_city.getText().toString().trim();
        String usr_univ = user_univ.getText().toString().trim();
        createEnrollclass Enroll = new createEnrollclass();   //getters and setters
        //Adding values
        Enroll.setusrname(usr_name);
        Enroll.setusremail(usr_email);
        Enroll.setusrnumber(usr_number);
        Enroll.setusrcity(usr_city);
        Enroll.setusruniv(usr_univ);
        ref.child("Enrollment").setValue(Enroll);    //storing into firebase
        if (v.getId() == R.id.enroll) {
            Toast.makeText(getApplicationContext(), "Enrollment Succesfull",
Toast.LENGTH_SHORT).show();
            Intent q = new Intent(getApplicationContext(), SelectFestActivity.class);
            startActivity(q);
        }
    }
    });
```

3. Create Fest

```
button4.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Creating firebase object
        /**  final Dialog dialog = new Dialog(create_fest.this);
         dialog.setContentView(R.layout.dialog);**/
        //  button4=(Button)dialog.findViewById(R.id.button4);
        final Firebase ref = new Firebase(Config.FIREBASE_URL);
        //Getting values to store
        String fest_name = nametext.getText().toString().trim();
        String college = collegetext.getText().toString().trim();
        String city = citytext.getText().toString().trim();
        String s_date = date.getText().toString().trim();
        //Creating Person object
        createFestclass fest = new createFestclass();
        //Adding values
        fest.setFestName(fest_name);
        fest.setCollege(college);
        fest.setCity(city);
        fest.setDate(s_date);
        //Storing values to firebase
        ref.child("create_fest").setValue(fest);
        if(v.getId()==R.id.button4)
        {
            Toast.makeText(getApplicationContext(), "Fest Created
Successfully",Toast.LENGTH_SHORT).show();
            Intent q=new Intent(getApplicationContext(),SelectFestActivity.class);
            startActivity(q);
        }
```

4. Fest feeds

```
public class fest_feeds extends AppCompatActivity {
    RecyclerView recyclerView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fest_feeds);
        recyclerView= (RecyclerView) findViewById(R.id.recyclerview);
        ReadRss readrss=new ReadRss(this,recyclerView);
        readrss.execute();
    }
}
```

5. Notifications

```
     Intent resultIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity, there's
// no need to create an artificial back stack.
PendingIntent resultPendingIntent =
 PendingIntent.getActivity(
this,
0,
resultIntent,
PendingIntent.FLAG_UPDATE_CURRENT
);

NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

--**********--