Campus Tribune

Mobile based Localized News Platform with User Generated Content

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree

Master of Science in Software Engineering

By
Divya Bitragunta, Sandyarathi Das, Shruthi Narayanan, Aditi Rajawat
August, 2016

APPROVED

Charles Zhang Digitally signed by Charles Zhang
Date: 2016.08.08 22:18:24 -07'00'

Prof. Charles Zhang, Project Advisor

Prof. Dan Harkey, Director, MS Software Engineering

Dr. Xiao Su, Department Chair

ABSTRACT

Campus Tribune

-Mobile based Localized News Platform with User Generated Content

By Divya Bitragunta, Sandyarathi Das, Shruthi Narayanan, Aditi Rajawat

Enablement of User Generated Content (UGC) in online media has been one of the most transformative innovations of Web 2.0. The Web 2.0 architecture breaks down the distinction between online media producers and consumers. The concept of UGC has paved way to applications like Reddit, Twitter and Quora, which are being used as platforms for open discussions and information sharing by the public and academic fraternity in equal measures.

The existing social media platforms are used on a global scale and thus posts and discussions related to a university become a very small part of the entire system. On the other hand, there is a genuine need for a mobile news application with UGC that caters to the interest and benefits of a local community like a university, which can significantly differ from those of other such communities. The most prominent challenges faced today by the UGC platforms are spam-control, front-page algorithms for the provision of user-specific news feeds and a comprehensive voting system to appreciate genuineness and significance of content.

We propose a virtual campus mobile platform for universities and colleges, where users can post and view campus news, subscribe to topics of their interest, and share information. We intend to address the issues like spam control with filtering techniques, provide a comprehensive voting mechanism based on reputation algorithms, incorporate security enabling measures and implement a strong front-page algorithm to improve user-experience. The ultimate goal of the project is to develop a highly performant, secure and scalable UGC mobile platform for universities, providing them with innovative, instant, localized, and reliable community news and information sharing service.

Acknowledgments

The authors are deeply indebted to Professor Charles Zhang for his valuable comments and assistance in the preparation of this study.

Table of Contents

List of Figures

List of Tables

Chapter 1. Project Overview

Introduction

Campus Tribune, a localized news platform is a native Android mobile application that leverages the UGC platform to allow academic fraternities to create and distribute campus news. With the advent of Web 2.0, creation of user-generated content started becoming quite common in the Internet. User generated content(UGC) includes, but not limited to, blogs, news reporting, information sharing and multimedia sharing. UGC platform paved way for users to build and share text or media content with other users which resulted in free flow of information over the Internet.

Our application allows registered users publish content related to their campus thereby creating a virtual campus community platform for each university or college. The application provides functionalities to initiate discussions on specific topics of interest and allows users to respond to the content in the form of comments, subscription or votes. The application can be broadly divided into four modules - user module, post module, events module and recommendation and search module. The user module has functionalities for a user to register and login to the application. The post module allows users to publish and manage a content. The post module also provides functionalities to comment on a content, up-vote or down-vote a content, follow or un-follow a content. The event module permits a user to publish and manage information about any event. The recommendation and search module employs algorithms to recommend relevant posts or events to users and provide effective search mechanism.

Since our application is driven by user generated content, it is very crucial to validate the quality of the information that is being shared. The spam filtering algorithm designed for the system makes sure that any published content is checked for the possible presence of abusive or offensive content. One of the most important goals of our application design is to provide custom news feeds and suggest news topics to users. Our recommendation system tracks a user's interests and activities to deliver relevant recommendations through emails and also enables in customizing the front page view for the user. The rich user interface and a robust front-page algorithm contribute to seamless reading experience and make the application more appealing to users. The availability of the application through the mobile platform along with the above mentioned features gives flexibility to the users to know what is happening in and around the campus at any time and from anywhere.

Proposed Areas of Study and Academic Contribution

- Mobile User Experience Design
- High-availability server architecture
- Machine Learning & Recommendation Engine design

Mobile app UI design is a major part of any mobile application. Creating a fluid user experience while interacting with the application should be paid primary attention during development phase to increase conversion performance. The key objectives of building a brand image, generating application traffic and there by revenues are achieved by creatively designing apps to deliver seamless user experience. User Interface (UI) is the medium through which a user interacts with the application, whereas User Experience (UX) defines how the app appeals to the user upon use.

The following points are a few of the design considerations that would be studied for Campus Tribune app:

- Responsiveness ⇒ Offload computation intensive operations to child thread
- Store & Forward Synchronization ⇒ to handle connection limitations
- Dynamic grid layouts for front page design
- Use of Action Bar

The ability of the server infrastructure to expand in response to the increase in workload is termed as being highly scalable. To incorporate this property, the server infrastructure should be designed to either scale horizontally or vertically. For the implementation of Campus Tribune, we will explore scaling the application by adding new nodes with identical functionality to the existing nodes in the server. This can be easily achieved by deploying the application in the cloud. In this design, to minimize response time and maximizing throughput, load balancing is employed to spread the incoming requests among the multiple available server nodes.

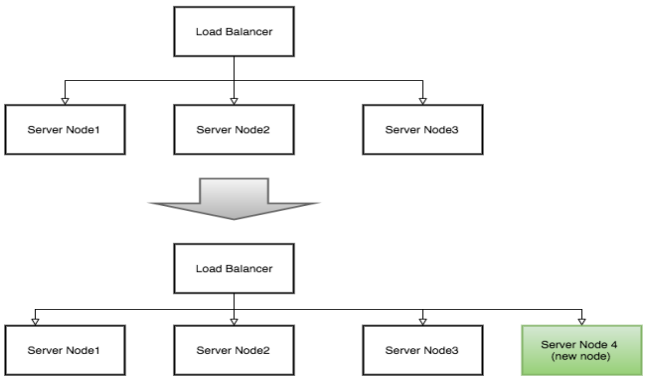


Figure 1: Horizontal Scaling

Here the Load-Balancing Cluster is employed in an (Active/Active) mode wherein the load is distributed among multiple backend redundant nodes that offer full-service capabilities and are active at the same time.

An important feature in Campus Tribune would be building a recommendation engine by collecting metrics about user, posts & events. Once there is sufficient amount of

metrics collected, user-post-category and user-event preferences are deduced by applying Regression techniques. Collaborative filtering techniques in the field of Machine Learning is employed to train the system to learn the pattern in which users make choices of posts and events. Once the system is trained using the data collected, predictions are made whether a particular post or event would be in the interest of the user, based on which recommendations are sent as emails or push notifications to the user. The computation of these recommendations also form the primary feed to generate a user's front page. Machine Learning methodologies like clustering, regression, classification using Naive- Bayes algorithm form the core of a collaborative filtering based Recommendation Engine.

Current State of the Art

The biggest challenges faced by the user-generated content based collaborative platforms are spam control and filtering, customization using the front-page algorithm and deriving the reputation system to provide incentives for constructive behavior. Much of the research has being done in these areas by both academia and industry leading to development of some of the novel approaches and techniques.

The existing spam-control techniques can be broadly classified into three main categories: content-based, link-based and hiding methods. The content-based methods analyze the content of the page and identify the deliberate modifications made in order to attain false high rate that causes the search engine to retrieve them as the genuine content. Various content properties of a spam page are used along with the machine learning models for improving the quality of spam detection techniques (Almeida, & Yamakami, 2010). The link-based methods identify the inappropriate links between different pages, which creates a network of misleading pages densely connected to each other and manipulates the search engine ranking results. The hiding methods identify the hidden high quality information, which is generally concealed by the spammers by cloaking and redirection. Fdez-Glez et al. have proposed WSF2 framework for spam detection and filtering, which uses all the three techniques in a simple combination. They have been able to demonstrate that WSF2 framework has better accuracy than using single filters (or classifiers) for spam detection (Fdez-Glez et al., 2016).

The front-page algorithm is essentially the recommendation algorithm used to customize the front page of an application. The prominent traditional recommendation techniques are based on collaborative filtering and node clustering. The user-based collaborative filtering techniques represent each user as a vector of all items, where each item has a positive or negative value depending on the user rating or a zero value if the user has no opinion about it. Some variant factors may also be considered to increase the significance of an item. The algorithm then calculates the similarity between the vectors of the current customer and all other customers in order to

recommend items. The collaborative filtering technique can also be based on the item-to-item relationships, which builds a list of similar items based on the customer's current items. The node clustering technique prepares a base of the customer's clusters depending upon their interests. The new customer is assigned to one of the clusters and the recommendations come from other customers of the allotted cluster. Zhao et al. have proved through their experiments that considering the asymmetric user influence and the global influence values in traditional user-based collaborative algorithms result in improved accuracy of the algorithm (Zhao, Wang, & Lai, 2016). Chen et al. have developed a framework to study the heterogeneous dynamics of the online users and proved that different update frequency of the recommendation list based on the user's activity results in improved performance of the traditional algorithms (Chen, Zeng, & Chen, 2015)

The most common types of reputation systems are – content-driven and user-driven. The content-driven reputation systems like Wikipedia and Maps derive the user and the content reputation based on the analysis of the user's interactions. On the other hand, the user-driven systems like eBay and Amazon derive reputations based on the explicit user feedback and ratings but often suffer from selection bias as users who are particularly happy or unhappy, are more likely to provide feedback or ratings (Alfaro et al., 2011).

The survey of the traditional techniques for spam-control, front-page recommendation and reputation system has revealed that leveraging their simple combinations and considering the effect of variant factors can improve the performance and accuracy of UGC-based collaborative platform.

Chapter 2. Project Architecture

Introduction

The motivation of the implementation architecture is taken from the concept of Micro-Services. A Micro-service architecture consists of several service components. Each service component is developed to represents a single, functional feature of the application and is deployed on single or multiple servers. Thus each functional component could be scaled independent of the other service components. In a traditional monolithic application, the main thread communicates with a single system/engine to respond to the client requests. But in the micro-services approach, the functionality is scaled out across a distributed topology to increase availability. In this architecture, the main advantage is that each component could be built and deployed independent of other service modules. Here the server is designed in a way to balance performance by choosing reduced or eventual consistency. Here each of the components shares some information either through a message queue or rests calls, but largely have their own distinct parts that they own. Such an application may have latency issues, but this could be addressed using Caching techniques. Given the timeframe to develop a working prototype of Campus Tribune, the packaging of each component as separate jars is deferred to a later stage, and instead the focus now is to build separate functional components in a single jar. The monolithic application thus built could be later decomposed into multiple jars and deployed to scale independently. The aim of this architecture design is to achieve optimal application performance & user-experience with seamless development and deployment methodologies and to understand the modalities of knitting various application frameworks together in building a whole some application.

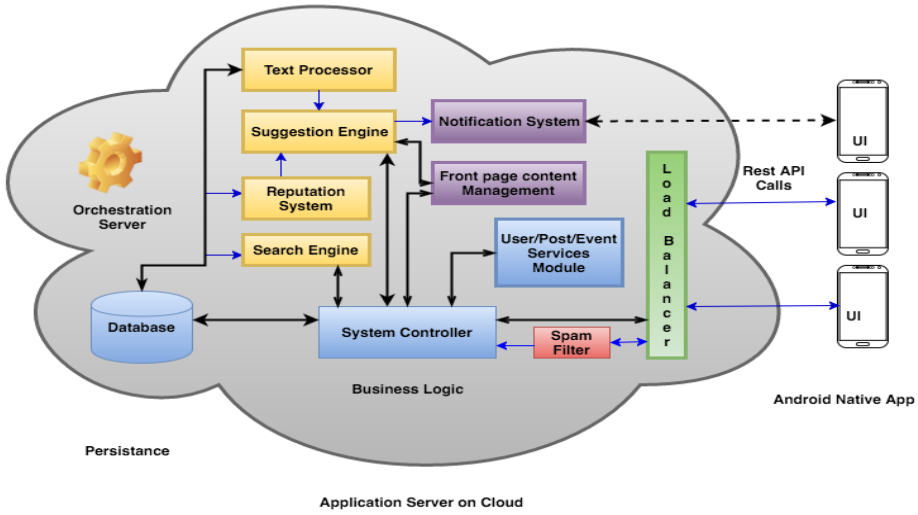


Figure 2: High-Level Architecture

Architecture Sub-systems

A. Database:

A NoSQL DB like MongoDB will be used to store the posts, events and user data in a semi structured JSON format. Mongo DB is a document store and provides the advantage of being schema-less. It has inherent features like text-search and full-text search features that would be useful to implement the search functionality in the application and MongoDB easily connects to analytic engines like Apache Spark/ Mahout for text processing and classification. MongoDB architecture provides features such as replication and authentication, which would aid to data security and as well availability. By deploying the Database server on the cloud, we can configure the auto-sharding feature, which would load balance the data requests.

B. User-Interface:

The user interface will provide the user access to login to the application, view & post content, vote on existing posts, etc. The user interface will be developed as a mobile application for Android based devices. The UI would be developed using the Android Java SDK by incorporating various screens serving distinct functionalities. The UI app will be built to engage the user by providing effective navigations, dynamic loading of the screens and most importantly a user-appealing layout. The UI will receive response in the HTTP- JSON format, and the client logic will parse the JSON and display the results on to the screen.

C. Cloud Application Server:

The cloud infrastructure provides a high performance, horizontally scaling and easily expandable infrastructure for the application. In addition, it provides system monitoring, load balancing and data sharding capabilities. The application will be hosted on Amazon Web Services (AWS) Elastic Compute Cloud (EC2). The server application would be packaged and deployed on multiple instances employing the auto-scaling feature to handle request loads without any service interruption due to the extra load.

D. System Controller:

The business logic component will provide a number of RESTful APIs, which the clients can invoke to interact with the application and also authenticating a user, checking permissions etc. The business logic will be developed using Java and Spring MVC Framework.

E. Reputation System

The Reputation System maintains a repository of user votes, most viewed post ids, most commented post ids and measures popularity. This reputation system will be responsible for the ranking of the pages based on user actions as stated earlier and will also be fed as an input to the front-page algorithm. The reputation system will also hold the logic for auto-deleting posts that are voted as spam by a threshold level of users.

F. Text Processor

A natural language processing (NLP) engine will be integrated into the system to analyze all the text posts submitted to detect topics, trends and to discover related posts. The main functionality of the text processor would be to extract tags from given document to classify them into certain categories, so that posts and events could be recommended to users based on their reading behavior.

G. Suggestion Engine

A suggestion engine will combine information from both the natural language processor and reputation system in order to suggest new posts for each individual user. The suggestion engine is a meaningful content discovery tool for users. The Suggestion engine would be implemented on the Apache Spark framework to provide recommendations based on a word-weighted machine-learning algorithm

H. Notification System

The notification system uses input from the suggestion engine and provides notifications in the form of push notifications or emails to recommend posts/events that the user may be interested in or opt to be notified about. The Notification framework will be implemented using Cloud Message Bus/Amazon Simple Notification Service (SNS).

I. Front-page Content Management system

The Front-page management system combines the user-interest profiles and recommendations from the recommendation engine to deliver personalized front-page content to the user’s home page.

J. Spam Filter

Every post and comment made by any user must pass through the spam filter before it can be persisted in the database. The spam filter will be implemented to filter out spam from URLs and text content generated by the user.

K. Search Engine

The Search engine will enable the user to search for posts, articles or events based on keywords or tokens. By utilizing the MongoDB auto indexing and text search feature, the application would provide a look-up feature to filter and fetch post/event documents based on keywords.

Chapter 3. Technology Descriptions

Campus Tribune architecture consists of the Client, Middle and Data-Tier technologies. The user interface is designed using Android and the communication with the MongoDB database is provided using REST API calls.

Client Technologies

User Interface plays an important role for an application as the Client interaction is done through it. Campus Tribune is a mobile application developed using Android.

Android

Android is a mobile OS developed by Google, designed primarily for use on mobile devices, such as smartphones and tablets. The UI is based on the touch inputs received from the user such as swiping or tapping on the mobile screen. Any API version from Android 5.0(Lollipop) is supported by the application. Android studio is being used for the development. An Android Virtual Device (AVD) can be defined in it with the required characteristics of an Android phone, tablet etc., that we want to simulate in the Android Emulator. The AVD Manager helps in creating and managing AVDs to effectively test the application.

Third Party APIs

Google provides APIs for the integration of Maps and Google Cloud Messaging Service (GCM). They are being used for the Events and Notification modules of the application. Maps are used for providing the location of an event and GCM is used for receiving the notifications.

Middle-Tier Technologies

The communication with the database is accomplished using Rest Service API's written in the Spring Framework. Spring Tool Suite is being used for the development. The Spring Tool Suite is an Eclipse-based development environment that is customized for developing Spring applications.

Spring Framework

The Spring Framework is an application framework whose core features can be used by any Java application. Spring provides core support for dependency injection, transaction management, web applications, data access, messaging, testing and many

other features. It helps in building simple, portable, fast and flexible JVM-based systems and applications.

RESTful Services

REST (Representational State Transfer) is an architectural style which is often used in the development of Web services for mobile applications, websites, mashup tools etc., The interactions between clients and services have a limited number of operations (GET, POST, PUT and DELETE). As light weight communication is provided between producer and consumer, REST is popularly used for building cloud-based APIs (Application Programming Interfaces). When Web services use the REST architecture, they are called RESTful APIs or REST APIs.

JMS

The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API used for sending messages between clients. JMS is used to send the Welcome mail when a user signs up for the application. It is also used to send out the email recommendations for the users.

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It provides computing resources with complete control and allows scaling of capacity as the requirements change.

Data-Tier Technologies

The database used is MongoDB and the server is deployed onto Amazon EC2. S3 buckets are used to store and retrieve the images.

MongoDB

MongoDB is a NoSQL database which uses document oriented data model. The architecture consists of collections and documents. Documents are the key-value pairs which is the basic unit for data. A collection is a set of documents. It supports dynamic schema design which allows documents to have different fields and structures. JSON documents can be produced as output which can be parsed for usage. Automatic sharding is provided enabling horizontal scalability.

Amazon S3

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It provides computing resources with complete control and allows scaling of capacity as the requirements change. Amazon S3 is a cloud storage provided in the Internet. We can create a bucket in one of the AWS regions to upload data (photos, videos, documents etc.). Any number of objects can be added to a bucket.

Chapter 4. Project Design

Client Design

Use-case Diagram

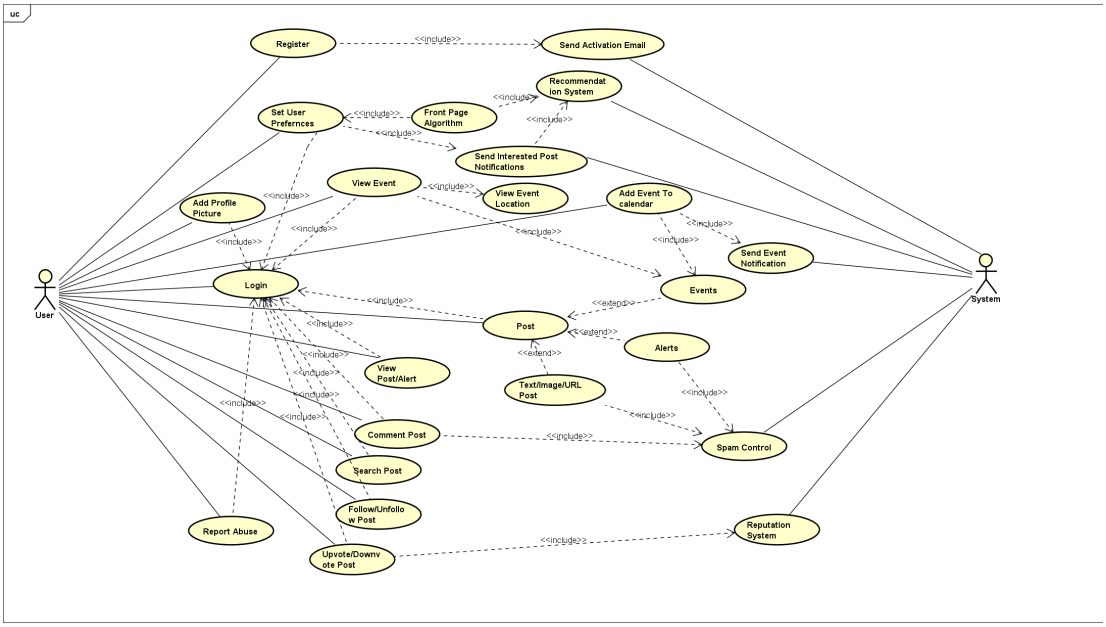


Figure 3: Use-case Diagram

Use Cases

The use cases of the application are broadly divided into four modules: user module, posts module, events module and search and recommendation module.

Use Case Name:	Sign Up for the application.
Description:	Registration by Signing Up for the application.
Trigger:	User will click the Sign Up button.
Preconditions:	User should have installed the mobile application. User needs to have an .edu id for the registration.
Post conditions:	User will be registered on the application. User is sent an activation code for verification.
Normal Flow:	User will click on the Sign Up button after entering the details. User receives an activation code for the verification.
Alternative Flows:	If the user does not have an .edu id or enters the activation code incorrectly registration will not be successful.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.

Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 1: Use Case 1

Use Case Name:	Sign in to the application.
Description:	Signing into the application.
Trigger:	User will click the Sign In button after entering the credentials.
Preconditions:	User should have installed the mobile application. User needs to be registered and verified.
Post conditions:	User is signed into the application. For the first time user is given a set of categories to select from. Subsequent Sign in's will get the front page of the application to appear.
Normal Flow:	User will click on the Sign in button after entering the credentials. User successfully opens the application.
Alternative Flows:	If the user enters wrong credentials, a message is displayed that wrong User Id /password was entered.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 2: Use Case 2

Use Case Name:	Enable/Disable notifications.
Description:	Enable/Disable notifications.
Trigger:	User will click on the enable/disable notifications button.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. User needs to open the settings.
Post conditions:	User will be able to enable/disable notifications. Notifications are sent as per the user selection.
Normal Flow:	User will click on the enable/disable notifications button. User receives notifications accordingly.
Alternative Flows:	If the user has already enabled / disabled the notifications and no change is needed.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 3: Use Case 3

Use Case Name:	Check for the posts and events in different categories.
Description:	Checking for the posts and events in different categories.
Trigger:	User will select the category of posts/events that the user is interested in.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application.
Post conditions:	User will be able to select the interested category. User is able to check the posts/events according to the selection.
Normal Flow:	User will click on the category he/she is interested in. User can view the posts/events in the selected category.
Alternative Flows:	If the user does not select any category then the front-page posts remain.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 4: Use Case 4

Use Case Name:	Create, Edit or Delete a post.
Description:	Create, Edit or delete a post.
Trigger:	User will click on the create post for creating the posts. For editing or deleting the user clicks on the edit or delete options of the post.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. For editing or deleting a post it should be created by the same user.
Post conditions:	User will be able to create a post. User is able to edit or delete a post.
Normal Flow:	User will click on the create post button. User enters the details required to create the post. User can edit or delete the post.
Alternative Flows:	If the user has not created the post, editing and deleting the post are not possible.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 5: Use Case 5

Use Case Name:	Create, Edit or Delete an event.
Description:	Create, Edit or delete an event.
Trigger:	User will click on the create event for creating the event. For editing or deleting the user clicks on the edit or delete options of

Preconditions:	User should have installed the mobile application. User needs to be signed into the application. For editing or deleting an event it should be created by the same
Post conditions:	User will be able to create an event. User is able to edit or delete an event.
Normal Flow:	User will Click on the create event button. User enters the details required to create the event. User can edit or delete the event.
Alternative Flows:	If the user has not created the event, editing and deleting the event are not possible.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 6: Use Case 6

Use Case Name:	Follow and receive notifications for Posts and Events.
Description:	Follow and receive notifications for Posts and Events.
Trigger:	User will click on the Follow button.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. User should have an event or post open that he/she wants to follow.
Post conditions:	User will be able to follow the post/event. User receives notifications for the followed posts/events.
Normal Flow:	User will Click on the follow button. User receives notifications for the posts/events that the user is following
Alternative Flows:	If the user is already following the post/event, an unfollow can be done.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 7: Use Case 7

Use Case Name:	Comment or Delete the comment for Posts and Events.
Description:	Comment or delete the comments for the posts/events.
Trigger:	User will enter text in the comment region and press enter. User can delete the comment by clicking the delete button.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. User needs to have a post/event open to comment on. User can delete the comment if it is posted by the same user.

Post conditions:	User will be able to comment on the post/event. User deletes a comment.
Normal Flow:	User will enter the comment in the text area and the comment is posted. User can delete a comment he/she previously posted by clicking on the delete button.
Alternative Flows:	If the user did not comment they cannot delete the comment.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 8: Use Case 8

Use Case Name:	Promote or Demote a post/event by upvoting or downvoting it.
Actors:	Users
Description:	Promote or Demote a post/event by upvoting or downvoting it.
Trigger:	User will click on the upvote/downvote button.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. User needs to have an event/post open.
Post conditions:	User will be able to upvote and downvote a post/event.
Normal Flow:	User will click on the upvote/downvote button. User response is registered.
Alternative Flows:	If the user has already cast a vote then can change the vote.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 9: Use Case 9

Use Case Name:	Report a post as Spam.
Description:	Report a post as Spam.
Trigger:	User will click on the Report Spam button.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application. User needs to have a post opened.
Post conditions:	User will be able to report the post as spam. The post is marked as spam.
Normal Flow:	User will click on the Report Spam button. The Post is then marked as spam.

Alternative Flows:	If the user already reported the post as spam then the option is not available.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 10: Use Case 10

Use Case Name:	Search for news or events.
Description:	Searching for news or events.
Trigger:	User will enter any text and click on the search icon.
Preconditions:	User should have installed the mobile application. User needs to be signed into the application.
Post conditions:	User will get the search results accordingly for posts and events as required.
Normal Flow:	User enters text in the search box and clicks on the search button. User gets the search results for posts/ events as desired.
Alternative Flows:	If the user does not enter any value a validation message to enter text is displayed.
Exceptions:	No Internet connection and not able to connect to the database. Option to retry or to cancel operation will appear.
Assumptions:	User understands English language Internet connection is present and the cloud is accessible.

Table 11: Use Case 11

Client Activity UML Diagrams

The primary client activities are deduced from the use cases related to four modules of the application. These are – signup and login, create posts (or events), view posts (or events), react on a post (or an event), receive notifications and search for posts (or events). The react on a post (or event) activity includes various actions that the user can perform on a post or an event.

Sign-up and Login

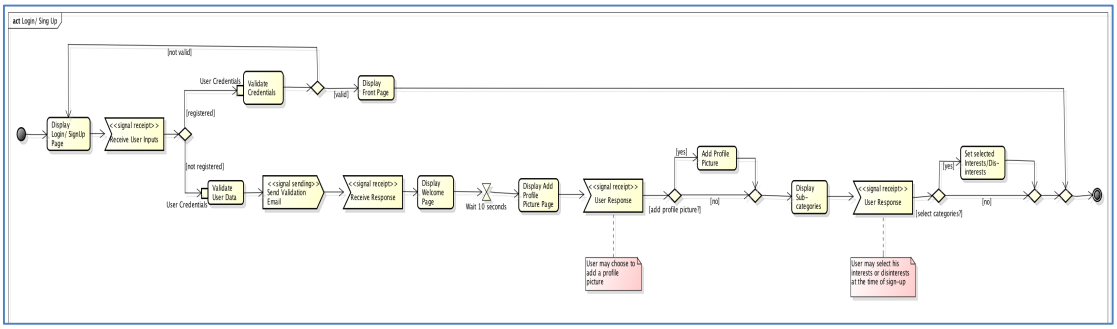


Figure 4: Sign up and Login

Create Posts (or Events)

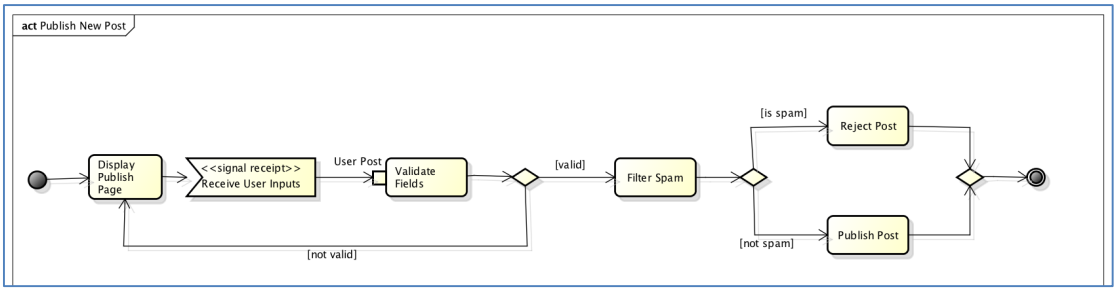


Figure 5: Create Posts (or Events)

View Posts (or Events)

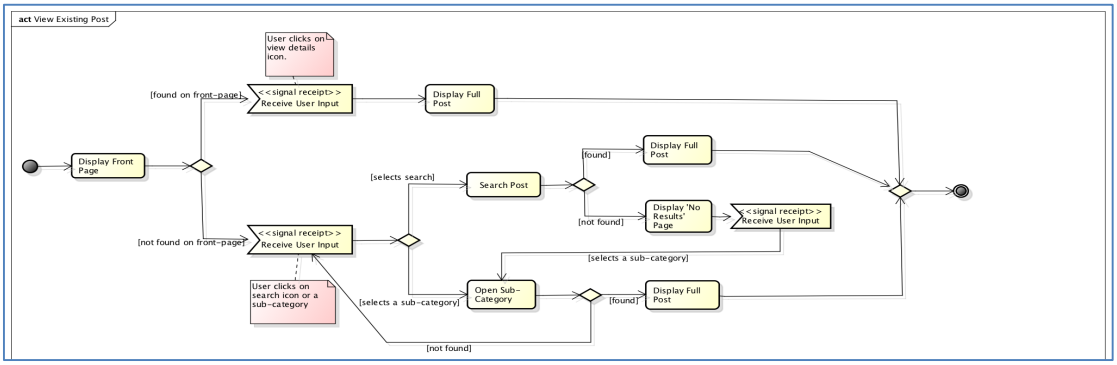


Figure 6: View Posts (or Events)

React on a Post (or an Event)

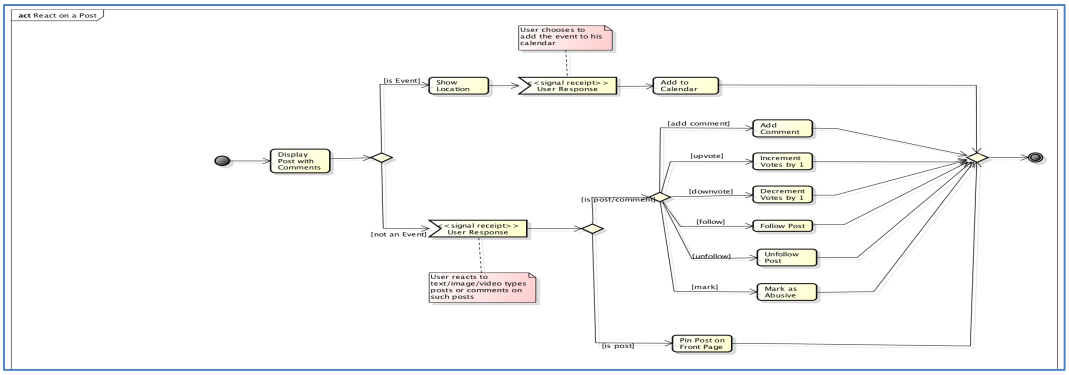


Figure 7: React on a Post (or an Event)

Receive Notifications

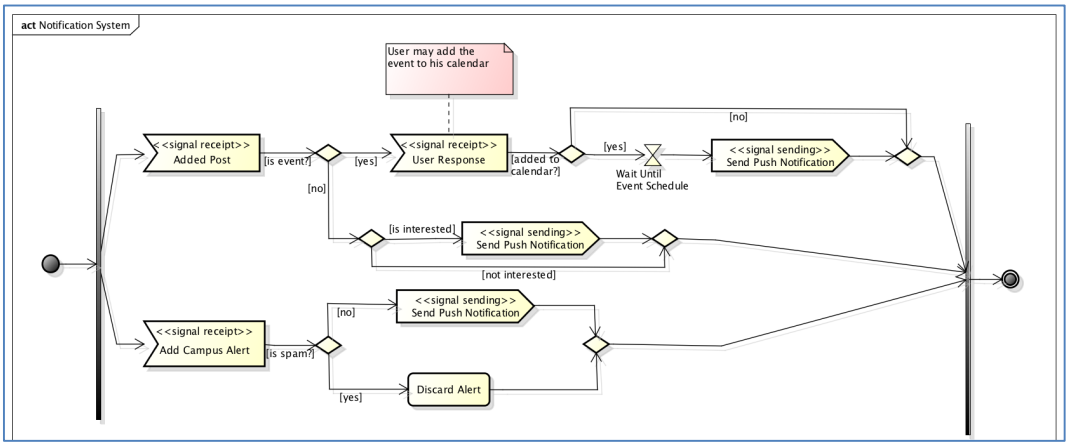


Figure 8: Receive Notifications

Search for Posts (or Events)

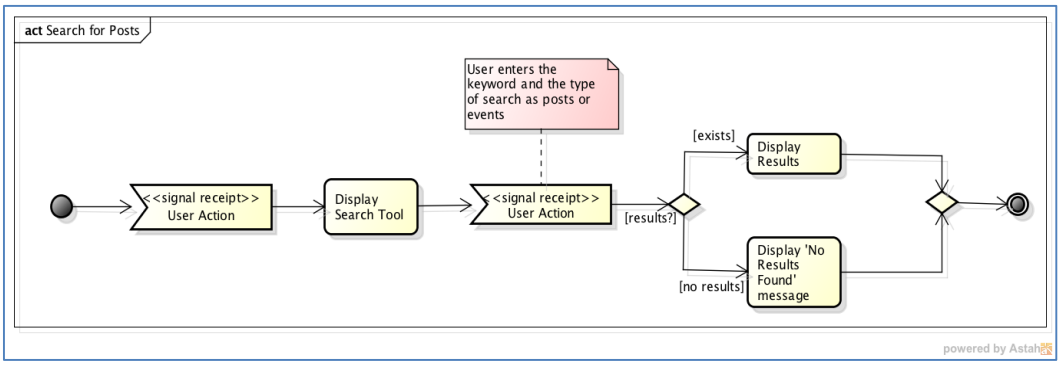


Figure 9: Search for Posts (or Events)

User Interface Design

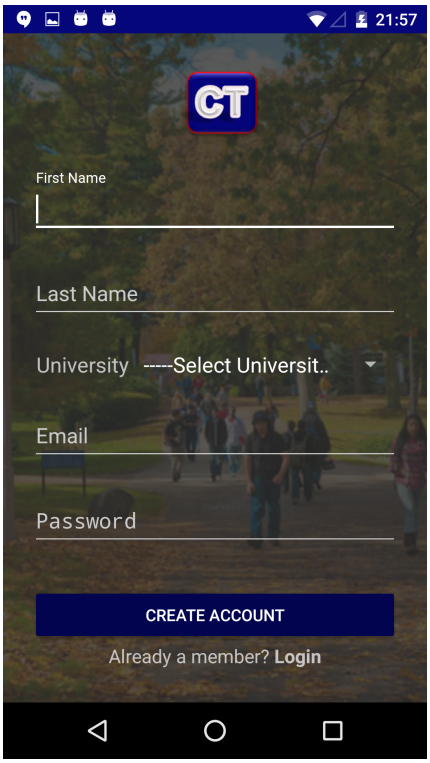


Figure 10: Login Screen

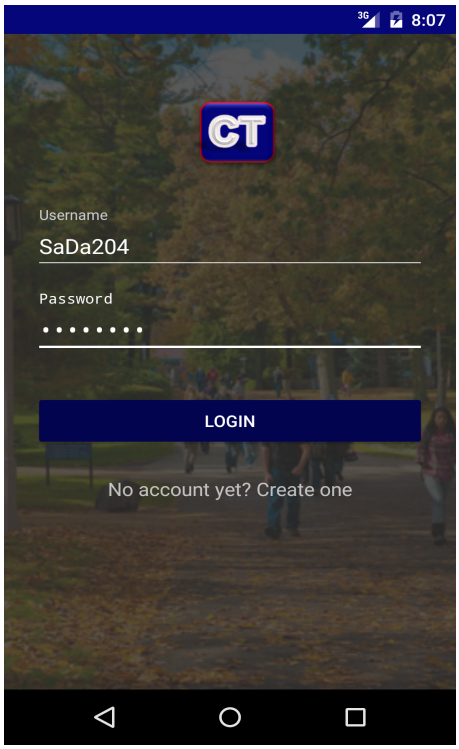


Figure 11: Sign up screen

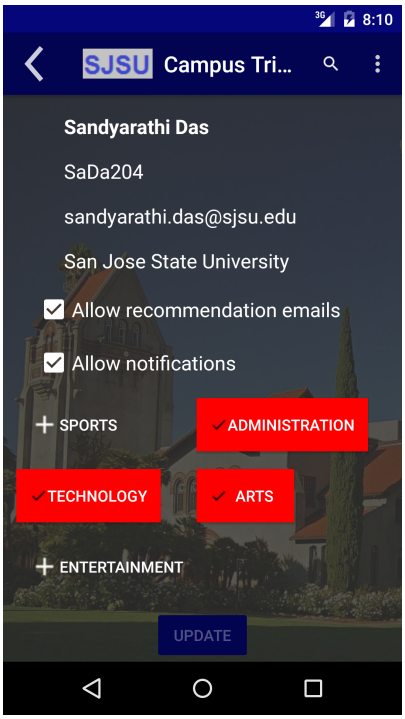


Figure 12: SJSU User Profile Screen

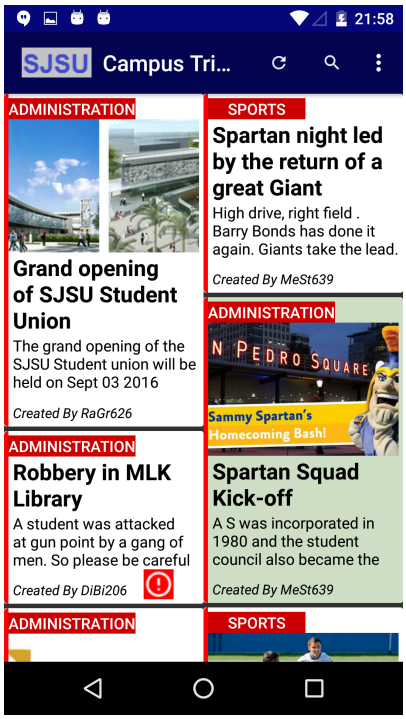


Figure 13: SJSU Front page Screen

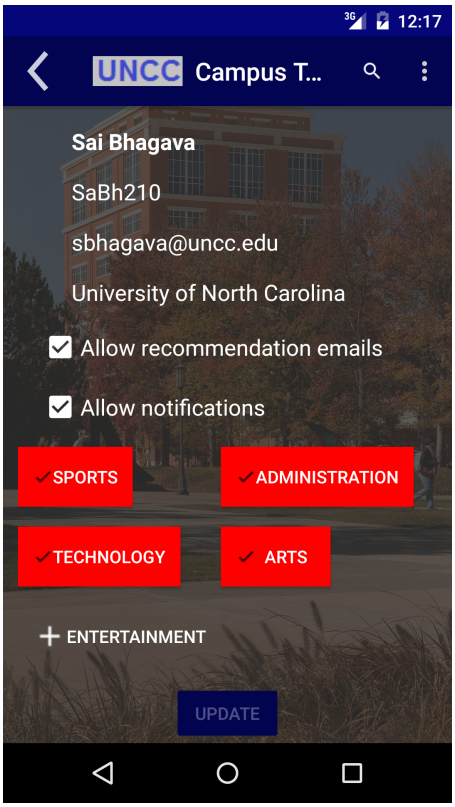


Figure 14: UNCC User Profile Screen

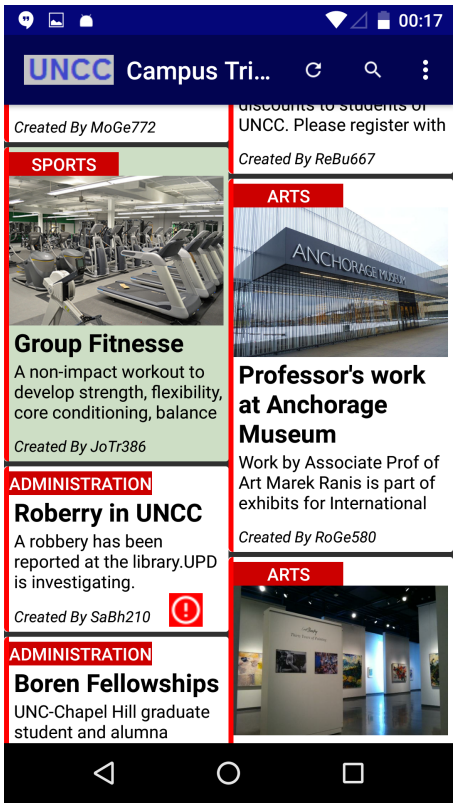


Figure 15: UNCC Front Page Screen

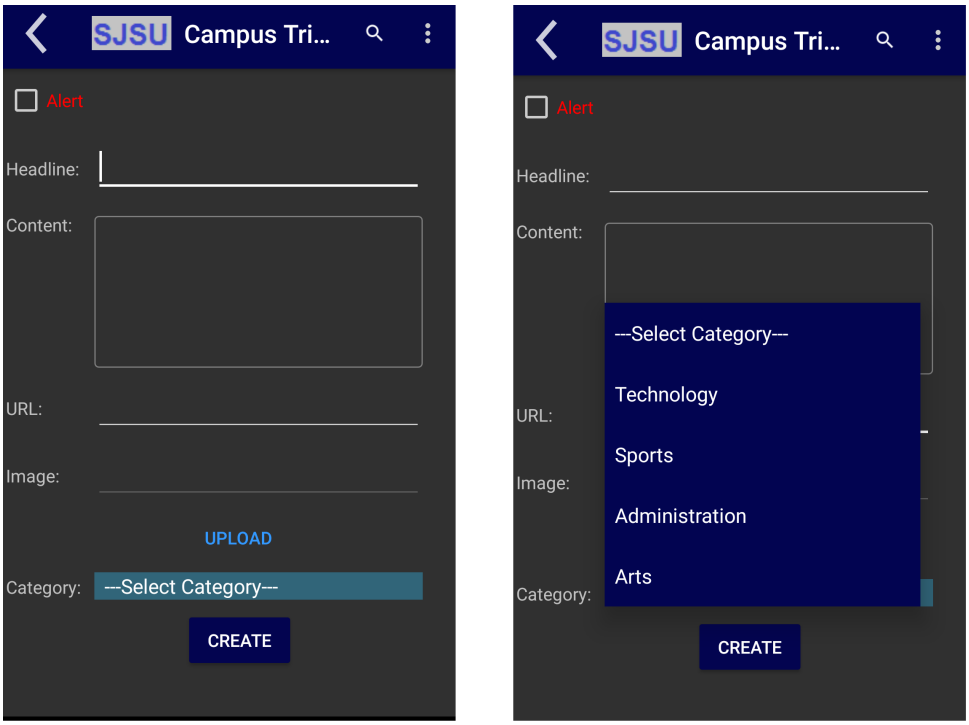
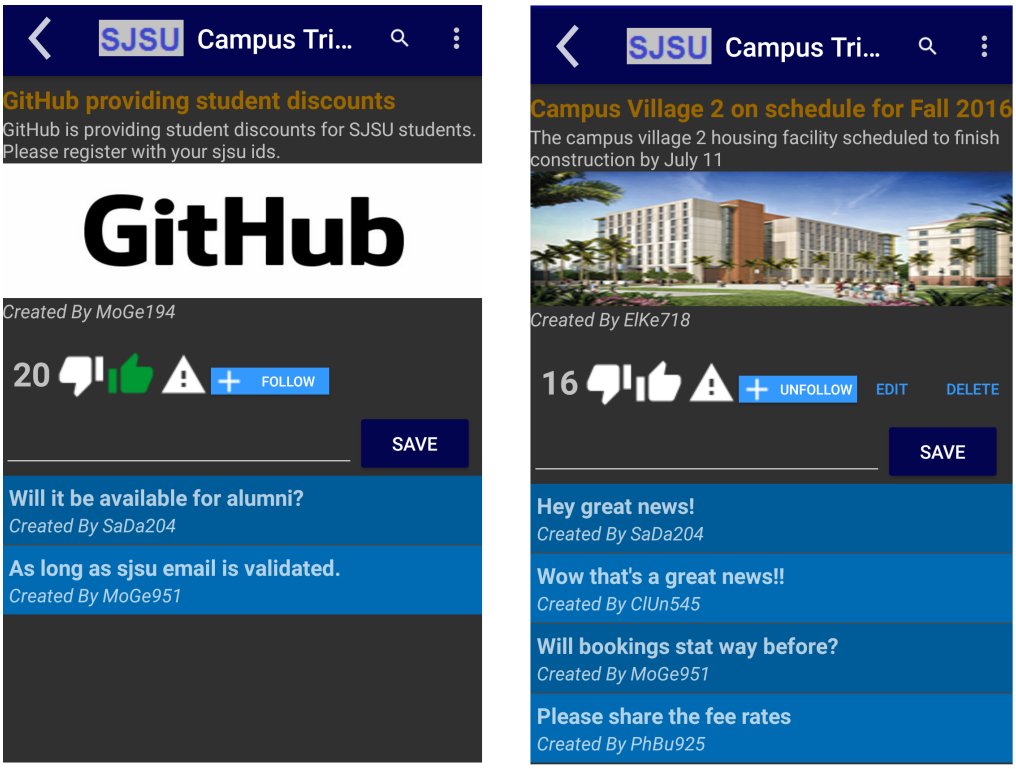
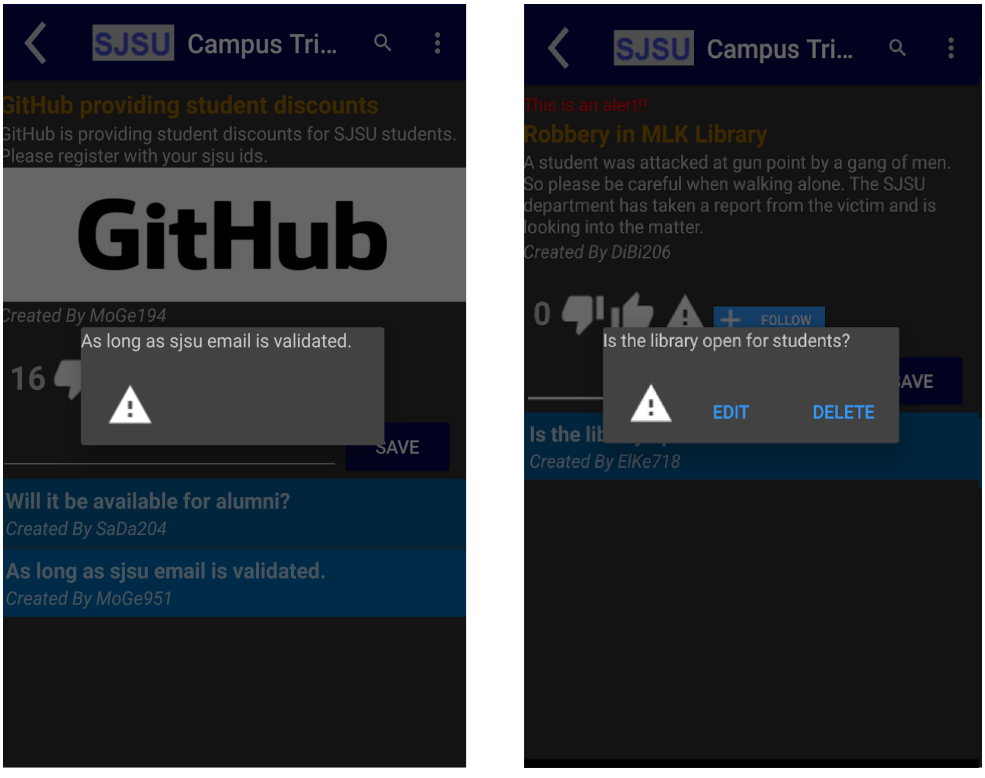


Figure 16: Create Post Screen



(as shown for post owner)

Figure 17: View Post Screen



(as shown for comment owner)

Figure 18: View Comment on Post Screen

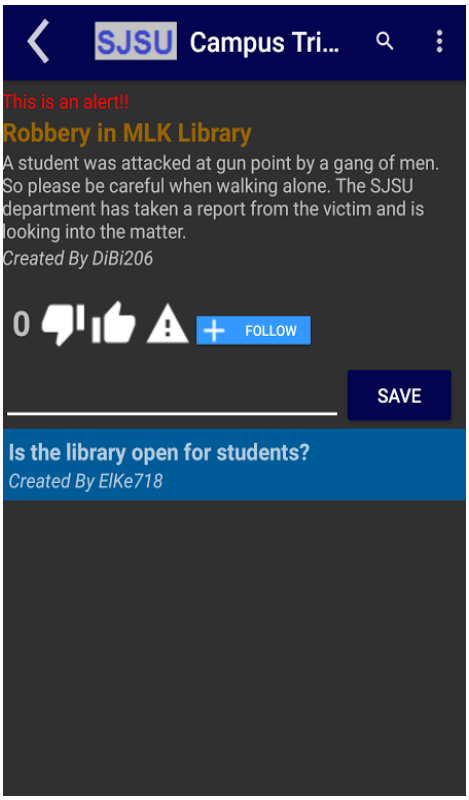


Figure 19: View Alert Screen

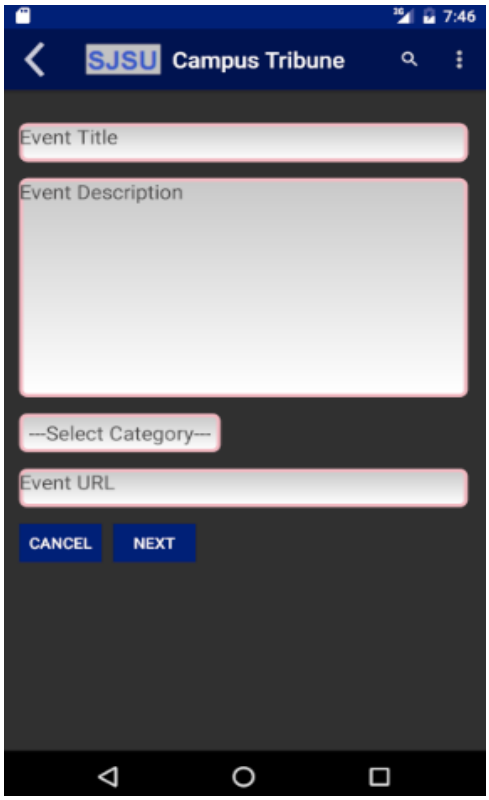


Figure 20: Create Event Screen

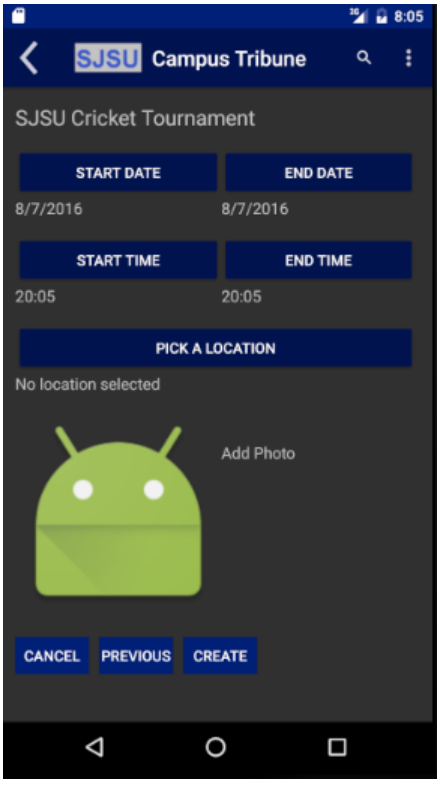


Figure 21: Enter Event Details Screen

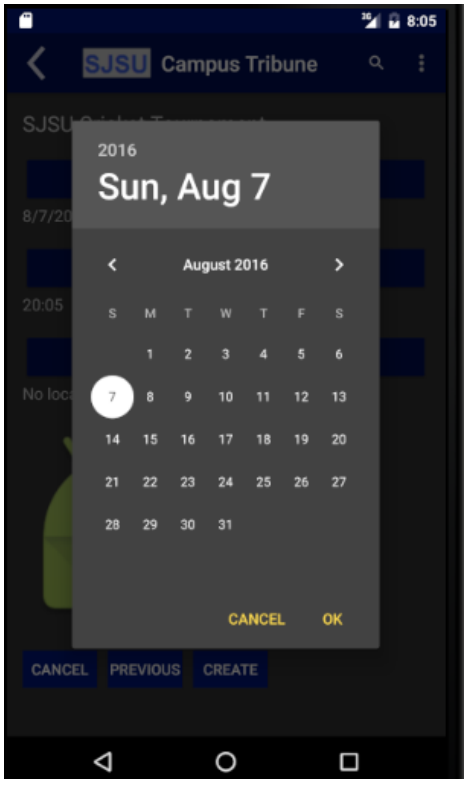


Figure 22: Add Event Date Screen

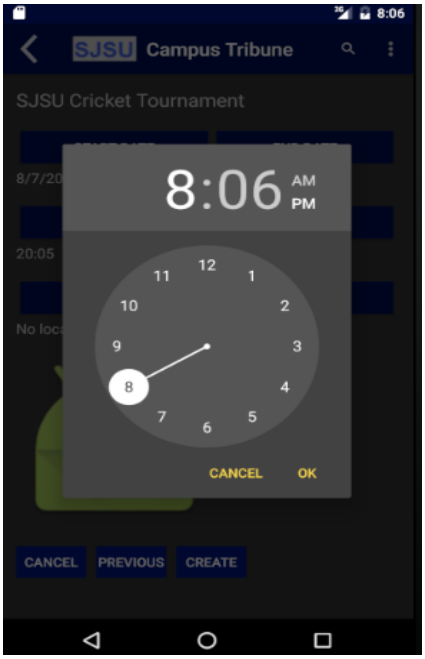


Figure 23: Enter Event Time Screen

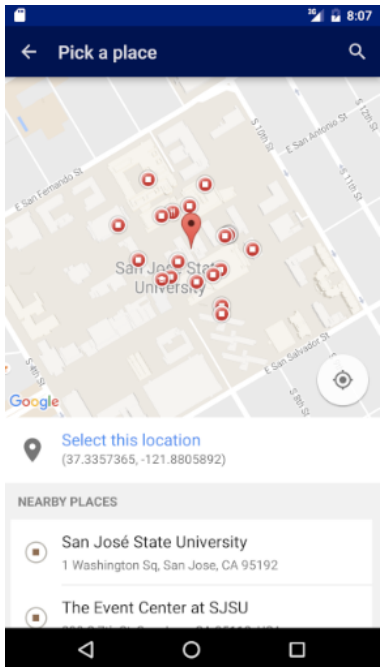


Figure 24: Place Picker Screen

Figure 25: View All Events Screen

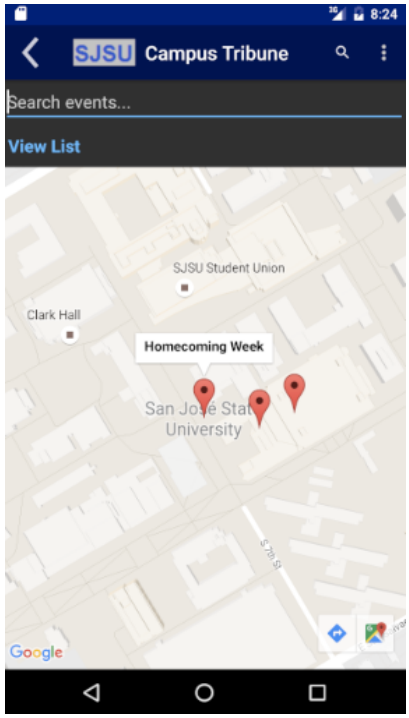


Figure 26: View All Events on Map Screen

Figure 27: View Event Details Screen

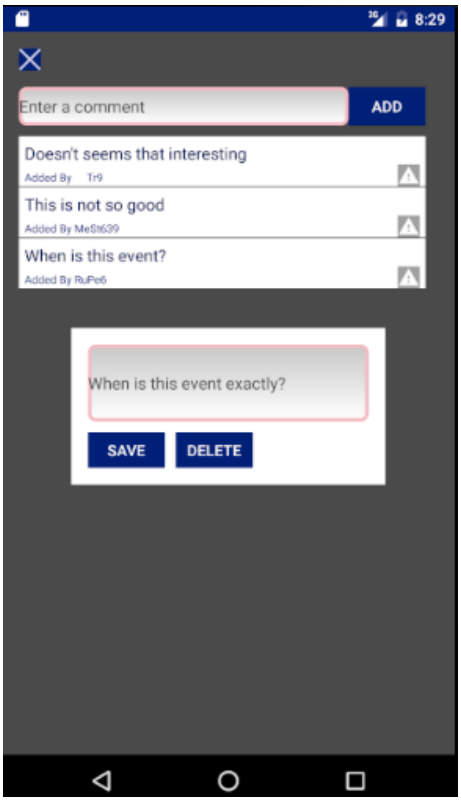


Figure 28: Comment on Event Screen

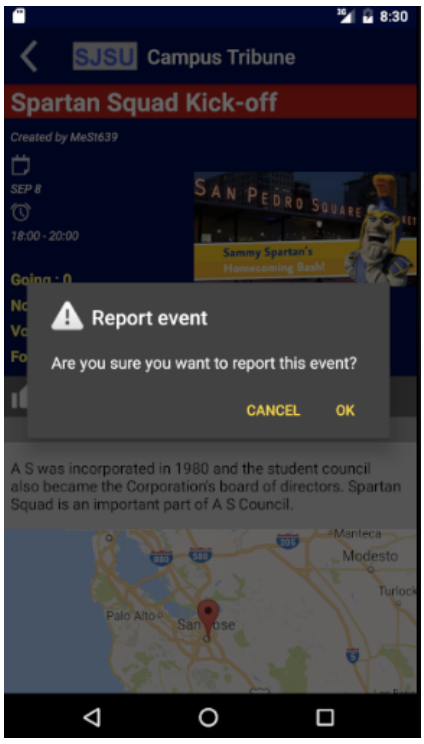


Figure 29: Report Event Screen

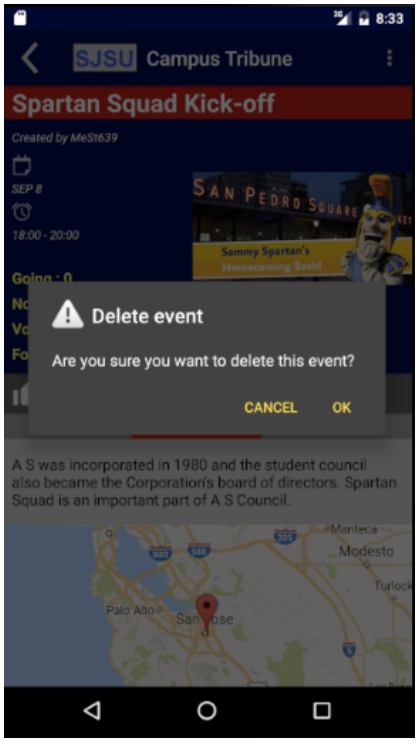


Figure 30: Delete Event Screen

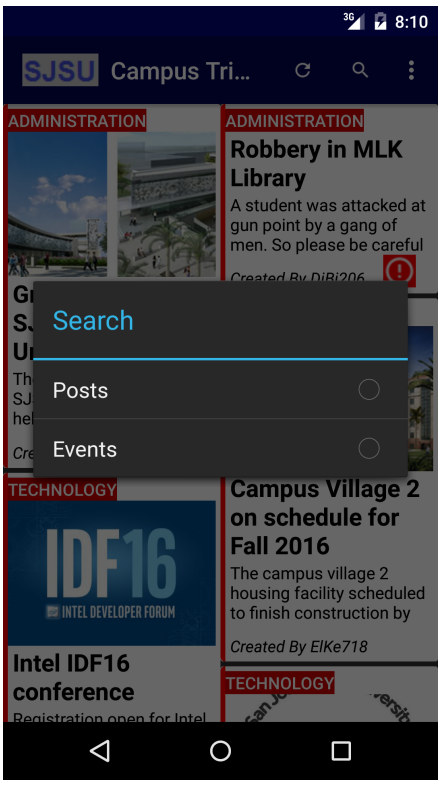


Figure 31: Search Icon Dialog Box

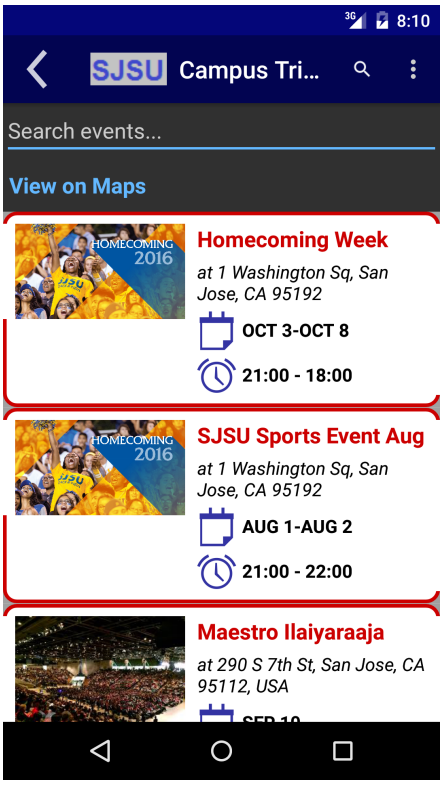


Figure 32: Search Events Screen

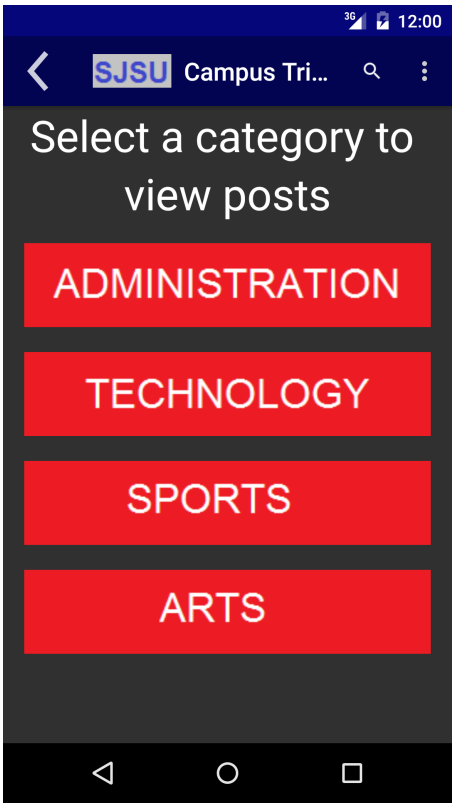


Figure 33: View Post by Category Screen

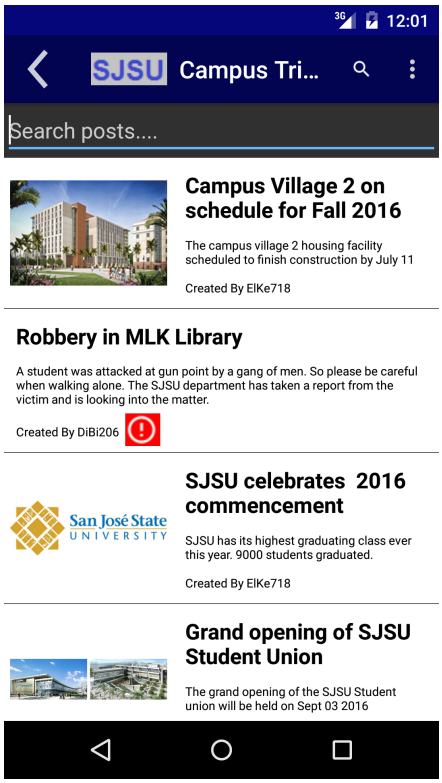


Figure 34: View All Posts By Category

Client Application Design Pattern

The four modules of the client application are designed to support a collection of Android activities according to the functionalities they provide. The various activities are interconnected according to the logical flow of the application. The table below shows the list of activities along with a brief description and the module name they belong to.

Activity Name	Description	Module
SignUpActivity	Application launcher activity and allows the user to register for the application.	User Module
LoginActivity	Started by SignUpActivity and allows the user to login using the registered credentials.	User Module
FrontPageActivity	Started by LoginActivity on authenticate user login and provides recommended news feeds to the user according to his interests. The application resumes from this activity every time the user returns to the application after first successful login.	Search and Recommendation Module
CreatePostActivity	Started when the user choose to create a new post from the application menu.	Posts Module
ViewPostActivity	Displays the details of a post. It can be started by multiple activities, as depicted in the diagram below.	Posts Module
CreateEventActivity	Started when the user choose to create a new event from the application menu.	Events Module
ViewEventActivity	Displays the details of an event. It can be started by multiple activities, as depicted in the diagram below.	Events Module
ViewAllEventsActivity	Displays the list of the events according to their start date or the search keyword.	Events Module
UserProfileActivity	Allows the user to make changes to his profile and enable/disable notifications.	User Module

SearchActivity	Allows the user to search for a post or an event by a keyword.	Search and Recommendation Module

Table 12: Activity Information

The diagram below depicts the client application logical flow.

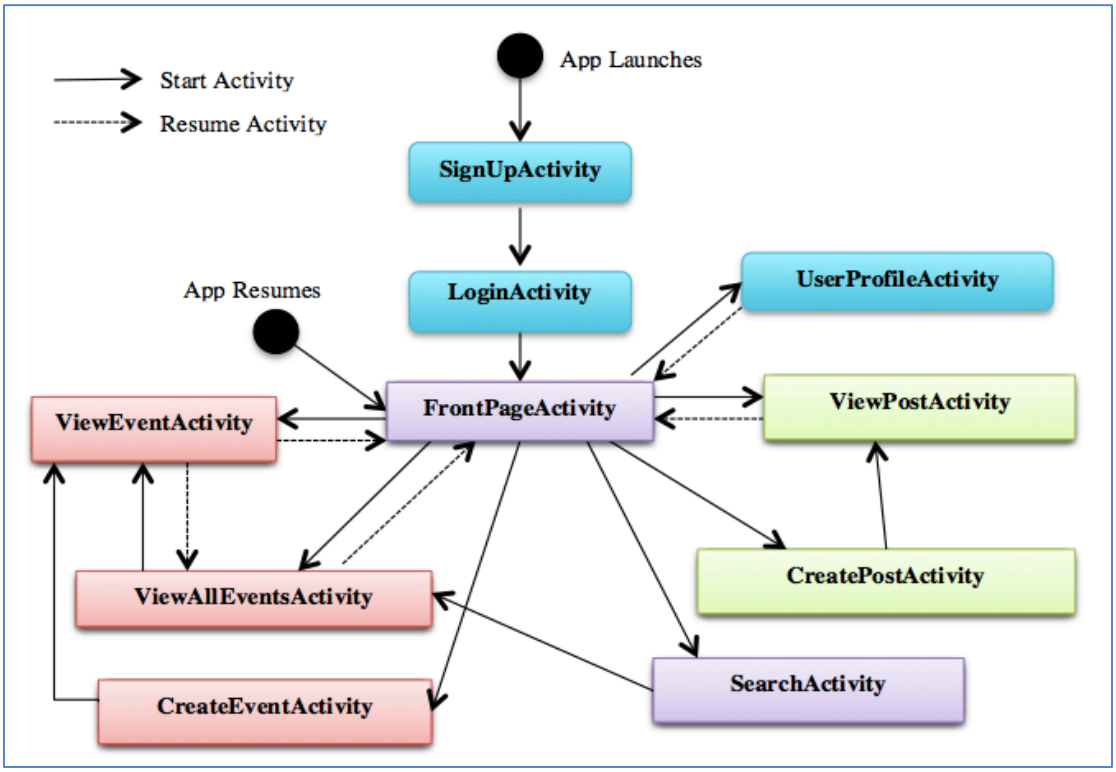


Figure 35: Client Application Logical Flow

Middle-Tier Design

The application middle tier consists of an application server that provides a REST interface to respond to http requests made by the client. The middle tier is responsible for accepting requests, authentication and authorization of users and user requests, fetching of resources(data) requested by the client. The request payload and response body received and sent by the application server is in the form of a JSON document embedded with a HTTP response status code.

System Sequence Diagram

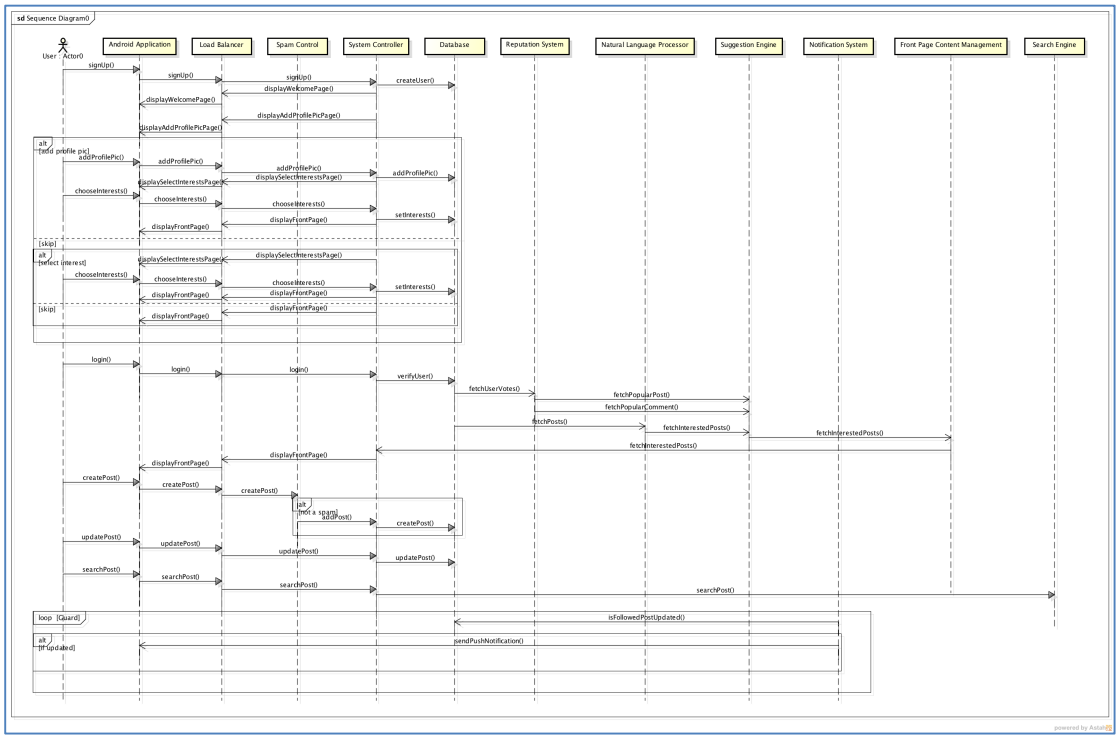


Figure 36: System Sequence Diagram

Logical Flow architecture

Users interact with the application using the Android native client and every action taken by the user stimulates a REST call, which reaches the server via a load balancer. The load balancer evenly distributes the client calls to the multiple instances of the server deployed on AWS EC2. Once the client call reaches a given server, it is first handled by the System controller, which acts as a medium through which the server responds to the client requests. It also enables caching of the response content. The System Controller triggers a method call to the appropriate service i.e. Post service or User service or Event Service, which internally perform the required computation based on the client request and sends the required data to the System Controller.

System Controller: The system controller will be a spring boot jar deployed on AWS with auto scaling feature. The client Rest API calls pass through a load balancer and are equally distributed amongst the multiples instances of the System Controller. The System Controller is also the medium through which responses are sent back to the client. The functionalities of the System Controller are:

1. Check for Spam in the content-body of the rest calls
2. Caching of response content (MemCache/ Redis)
3. Authentication

4. Front-Page Algorithm
5. Reputation Mechanism
6. Forwarding of requests to appropriate service modules through method Calls
    a. /event
    b. /post
    c. /user
    d. /search & /recommend
7. Provide appropriate response to the Client Requests

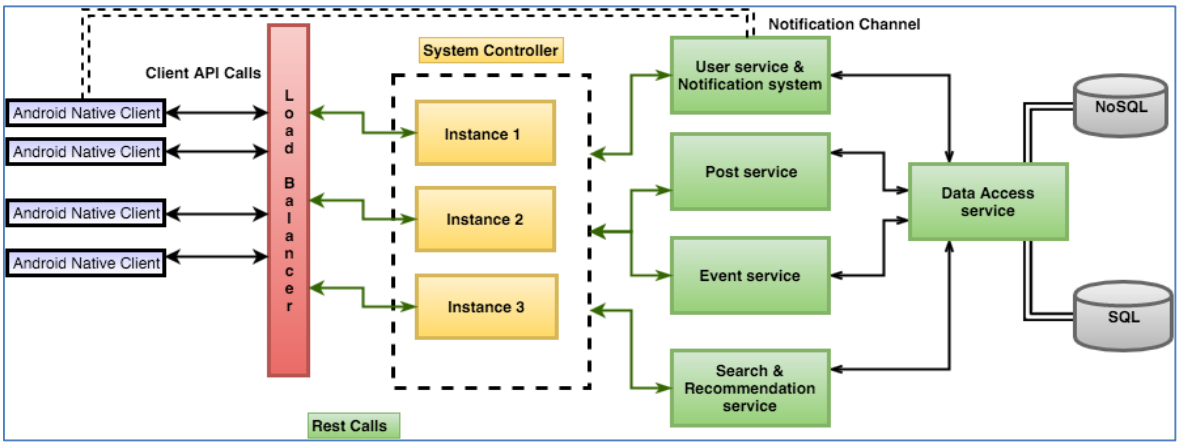


Figure 37: Logical Flow Architecture

Services Modules: The services module lies embedded into the application jar even though they are functionally distinct and are responsible only for services pertaining to their entity module.

- Event Services API: Business logic for Events Module
- Post Services API: Business logic for Post Module
- User Service API: Business logic for User Module
- Search & Recommendation Service API: Business logic for Search & Recommendation services
- Data Service API: The data service API is used by the other modules to communicate with the database through (data access objects or repository modules)

The middle tier employs the following technologies Spring-boot Framework, Spring-boot Rest Data Framework, Cache store for Auth token validation.

Component APIs Design

User Authentication and Authorization Service

To secure the application from unauthorized users, The Campus Tribune app employs a Token based authentication using Spring-boot Security framework.

User Signup

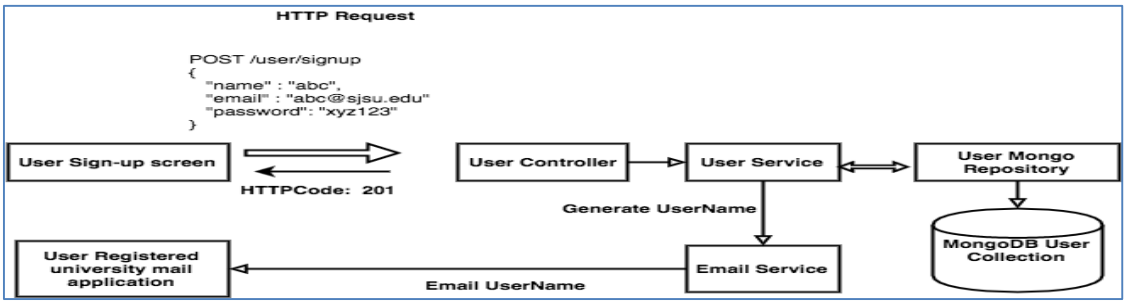


Figure 38: User Signup

User Login

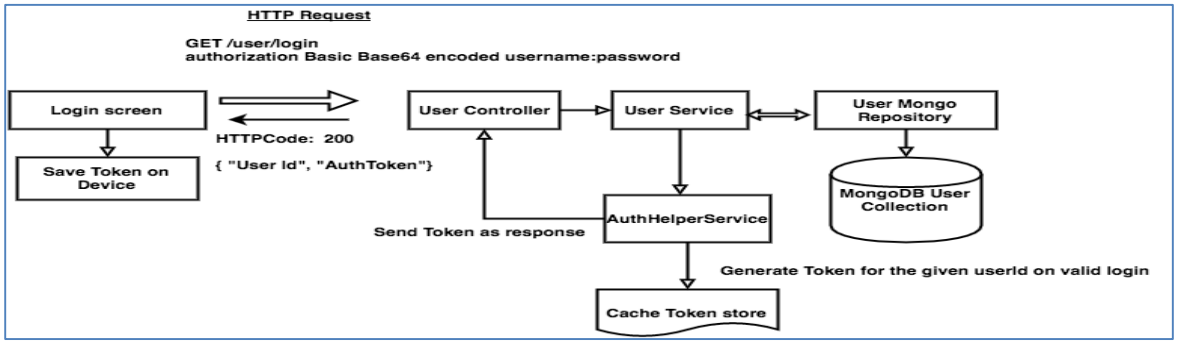


Figure 39: User Login

Front Page

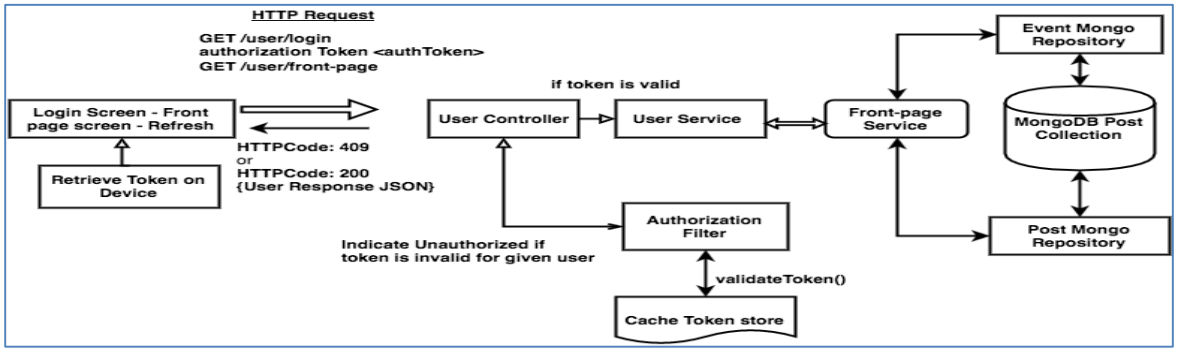


Figure 40: Front-page

The following table shows all the APIs for the user module

API	Method Type	HTTP Response Code	JSON Response
/user/signup	POST	201	User JSON
/user/login	GET	200 or 401/404	User JSON with Front Page Data
/user/front-page	GET	200 or 401/404	User JSON with refreshed Front Page Data
/user/user-profile	POST	200 or 401/404	User JSON with updated user subscription

Table 13: UserModule API

Post Service

The module's middle tier design mainly involves two functionalities - posts and comments.

Post Functionality

The post functionality allows a user to create post and view/edit/delete posts. A user can also vote/report/follow the post. The main java classes are PostController.java, PostService.java, PostDAO.java and PostUserDAO.java

The following is a sample screenshot showing the flow of data in the middle tier for Create Post feature.

Create Post

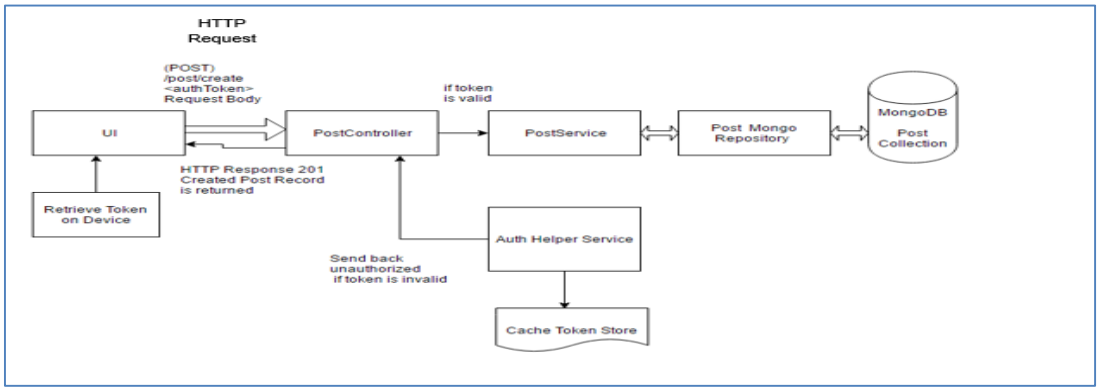


Figure 41: Create Post

View Post

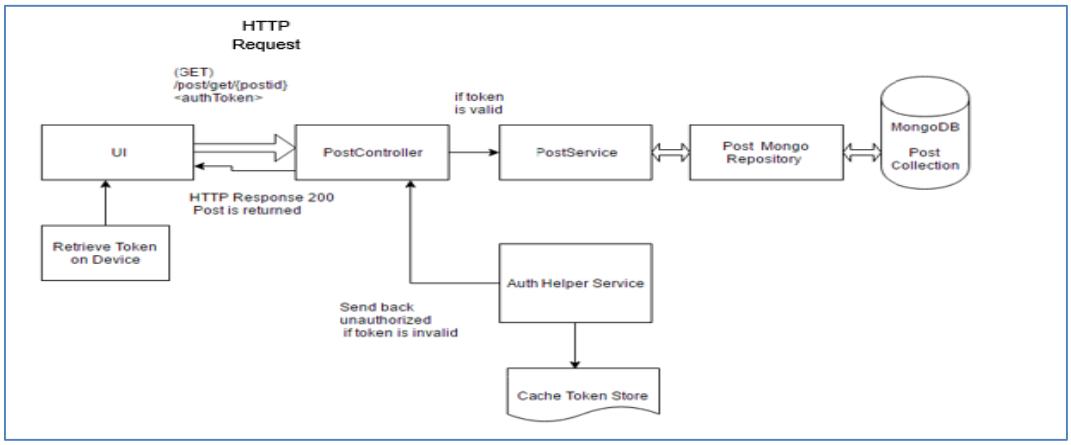


Figure 42: View Post

Comments Functionality

The comments functionality allows a user to create comments and view/edit/delete comments. A user can also report the comment. The main java classes are PostCommentsController.java, PostCommentsService.java and PostCommentDAO.java

The following table shows all the APIs for the post module

API	Method Type	Request	HTTP Response Code	Response
/post/create	POST	Post JSON	201 or 500	Post JSON
/post/view/{post_id}	GET		200 or 500	Post JSON
/post/update	PUT	Post JSON	200 or 500	Post JSON
/post/remove/{post_id}	DELETE		200 or 500	
/post//vote/{voteType}/by User/{user_id}	POST	Post JSON	200 or 500	Post JSON
/post//report/byUser/{user _id}	POST	Post JSON	200 or 500	Post JSON
/post//follow/{post_id}/by User/{user_id}	POST		200 or 500	
/post/getByCategory/{cate gory}	GET		200 or 500	Post JSON array
/comment/create	POST	Comment JSON	201 or 500	Comment JSON
/comment/view/{comment _id}/forPost/{post_id}	GET		200 or 500	Comment JSON
/comment/viewAll/forPost /{post_id}	GET		200 or 500	Comment JSON array

/comment/update	PUT	Comment JSON	200 or 500	Comment JSON
/comment/remove/{comment_id}/forPost/{post_id}	DELETE		200 or 500	
/comment//report/byUser/{user_id}	POST	Comment JSON	200 or 500	Comment JSON
/post/getUserActions/{user_id}	GET			PostUser JSON

Table 14: Post Module APIs

Event Service

Event service is used for creating an event, fetching details of upcoming events, updating an event's details and deleting an event. The service also creates, update and delete comments on events and record the lists of events a user has reacted on. All the details of an event are stored in events collections and comments are stored in eventComments collection. The list of events reacted on by a particular user is managed in eventuser collection. The event service provides the following APIs.

API	Request	Http Response Code	Response
POST /events/	Event JSON	201 or 500	Event JSON
GET /events/	-	200 or 500	ArrayList<Event> JSON
PUT /events/{event_id}	Event JSON	204 or 500	-
DELETE /events/{event_id}	-	204 or 500	-
GET /eventusers/{username}	-	200 or 500	EventUserDAO JSON

Table 15: Event Module APIs

The event service is composed of three primary components: EventController to receive all the API requests, EventService to process the request and Repository Interfaces to retrieve and update the data stored in the collections. The diagram below demonstrates an example of the logical flow of the event service APIs.

View All Upcoming Events

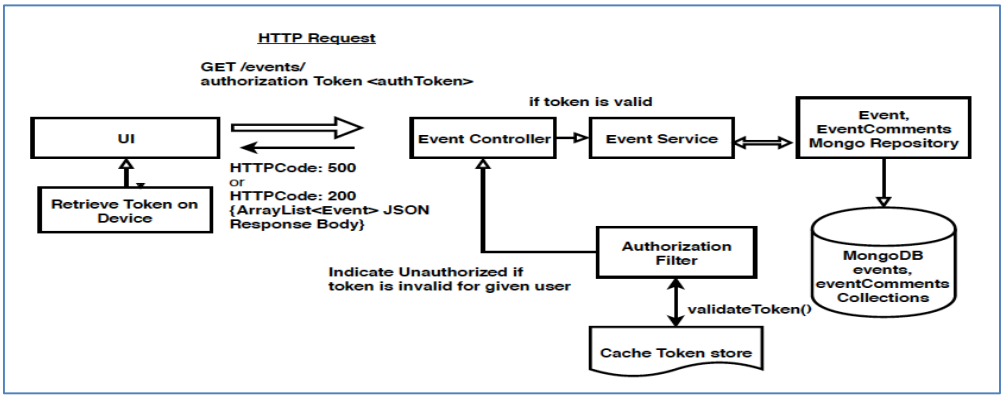


Figure 43: View All Upcoming Events

Notification Services

A notification system is implemented to send push notifications to the user based on the posts and events which the user follows. An email notification system is also implemented for sending recommendation mails according to the user interests. Notification Manager is created and used to notify the user when any event happens. The notifications appear in the status bar of the phone. They appear when the application is not opened in the screen.

Notifications take different forms to let the user know that something is happening. It might take any form of the following:

- A persistent icon that goes in the status bar and is accessible through the launcher, (when the user selects it, a designated Intent is launched),
- Turning on or flashing LEDs on the device, or
- Alerting the user by flashing the backlight, playing a sound, or vibrating.

This is according to the user notifications settings on the mobile phone.

Android Push Notifications are done using Google Cloud Messaging (GCM). It is used as a notification engine. The Campus Tribune App receives the message from Google cloud messaging server (GCM).

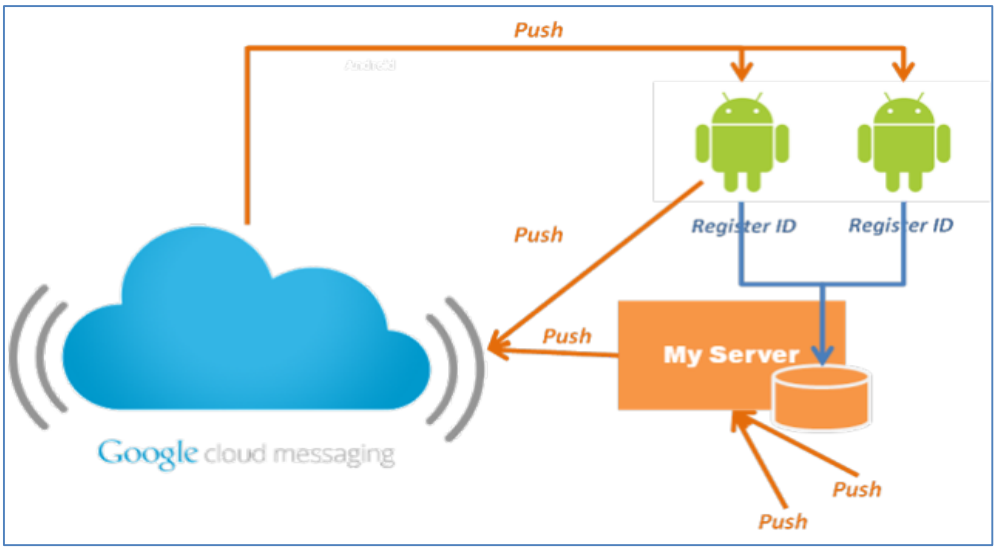


Figure 44: GCM Architecture

GCM is implemented in the project in two parts.

- A GCM server application is implemented to push the notifications.
- A GCM client application is implemented to receive and process Notifications from GCM server.

We switch on the GCM for android and create a key for using it specifying the package name and security details. A broadcast service and receiver is implemented. It is then combined with the Notification Manager in the android. A registration token is sent to the mobile device after which it is ready to receive notifications. Mailing notifications for recommendations is done using JMS.

Component Class Diagrams

Class Diagram for User Module

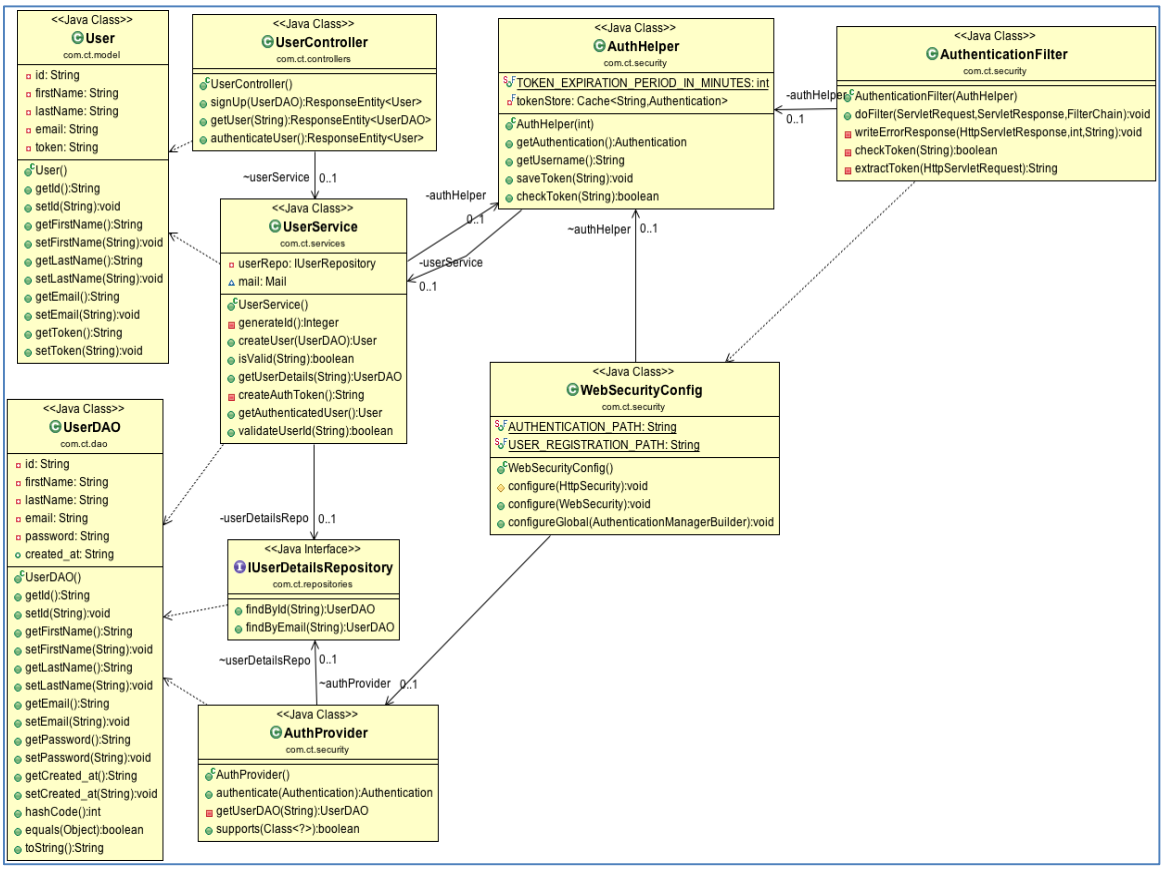


Figure 45: Class Diagram for User Module

Class Diagram for Post Module

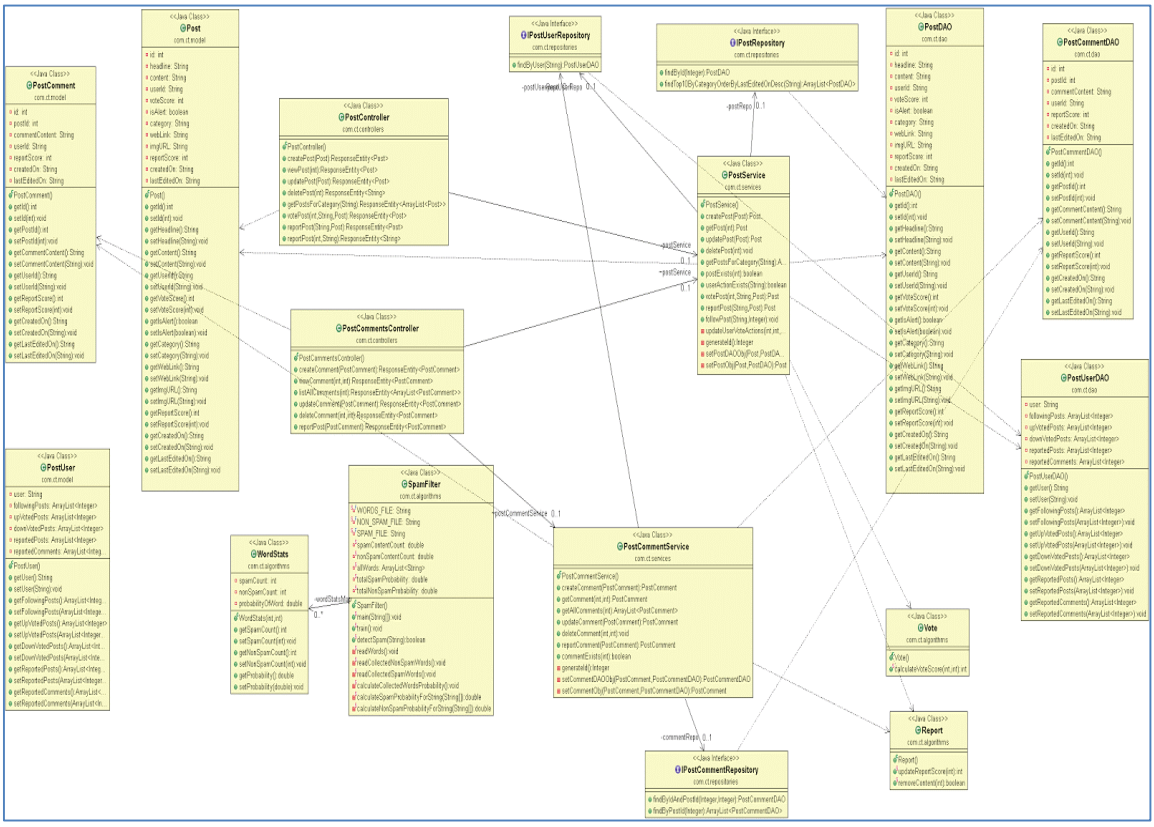


Figure 46: Class Diagram for Post Module

Class diagram for Events Module

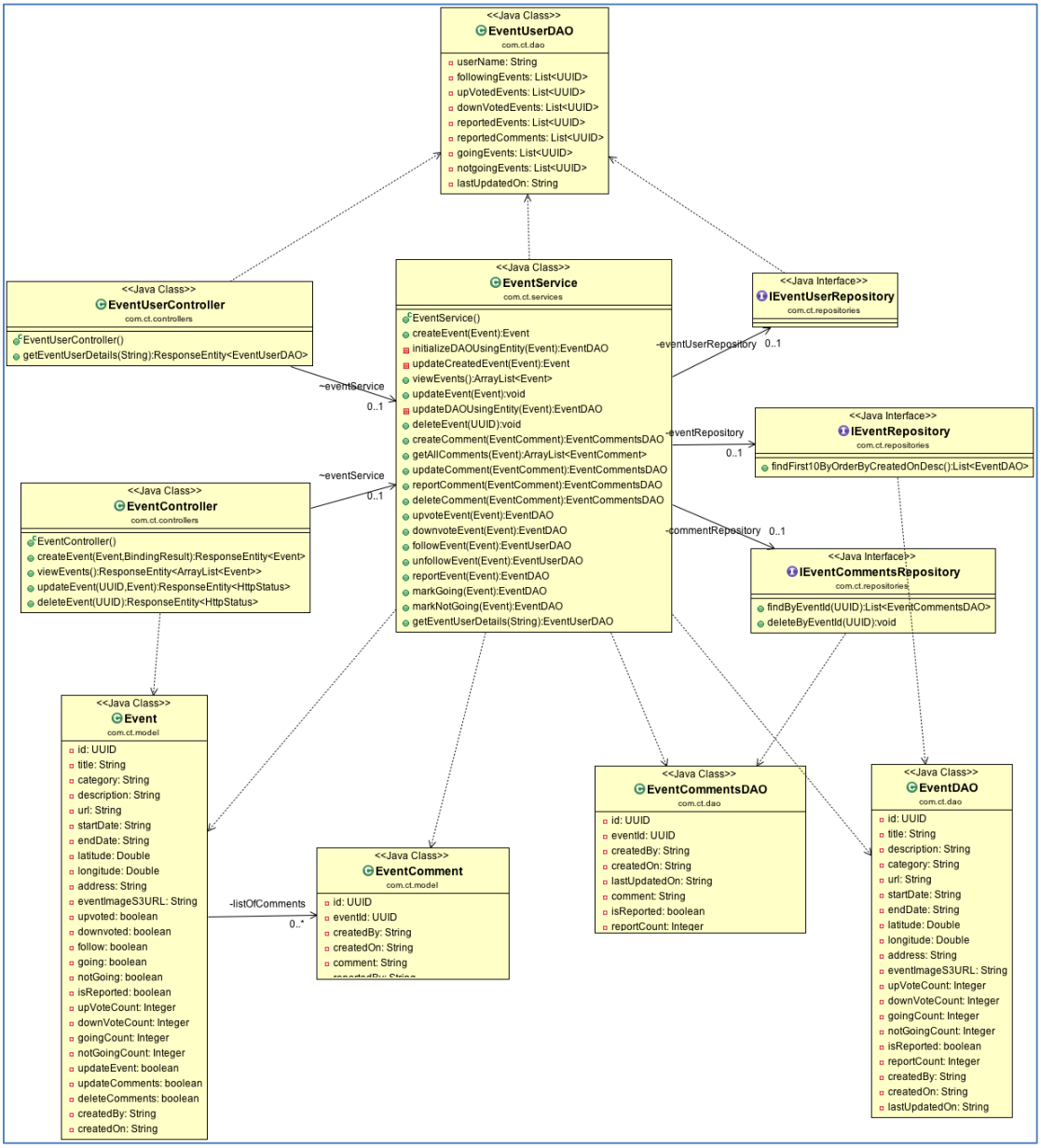


Figure 47: Class diagram for Events Module

Data-Tier Design

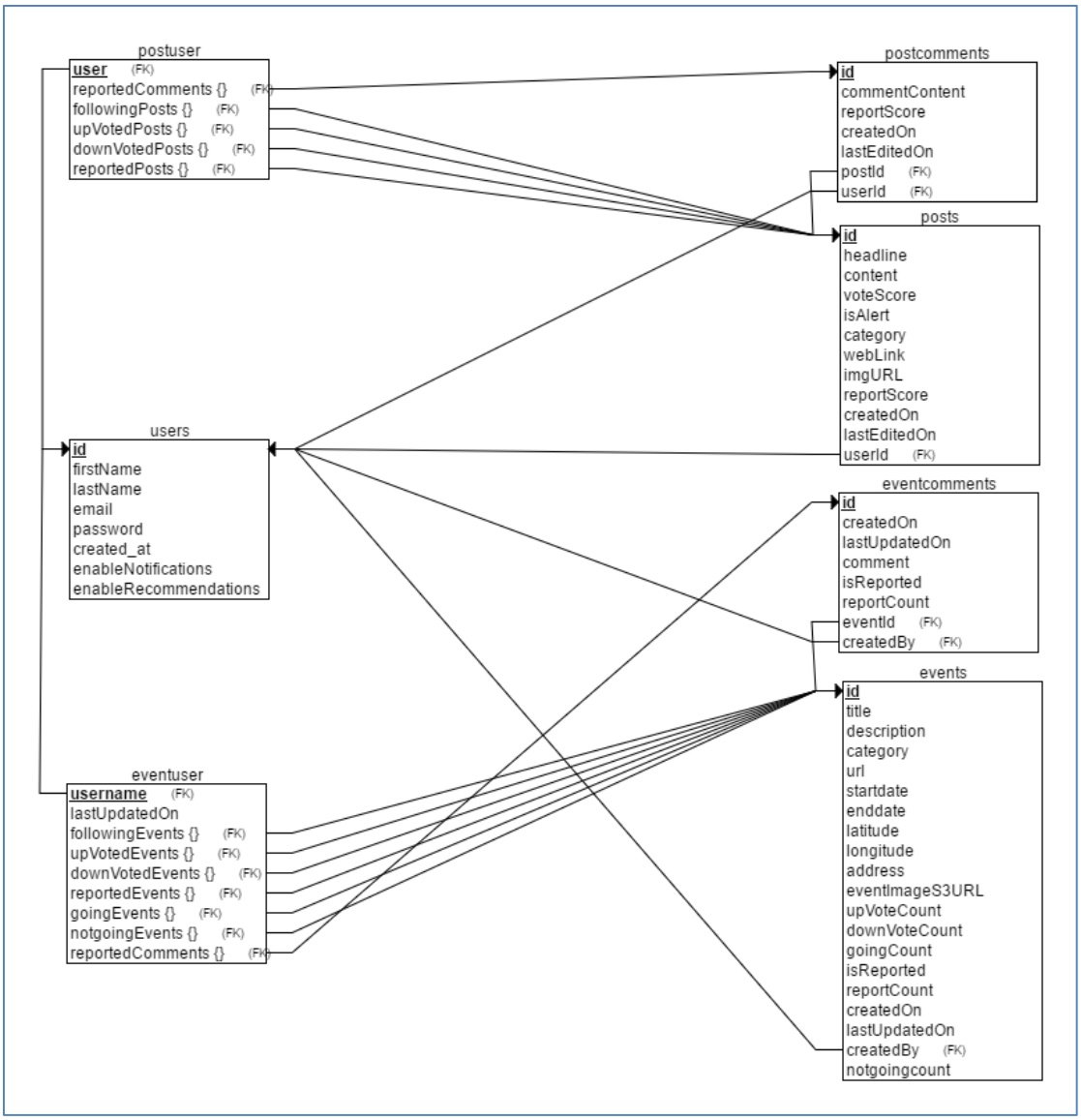


Figure 48: Data-Tier Design

Chapter 5. Project Implementation

Client Implementation

The user interface can be separated into five main modules - the user module, front-page module, post module, events module.

User Module UI

The User Interface Module has four functionalities, namely user sign-up, user login and authentication token persistence and display of user profile. The layout XMLs (Extensible Markup Language) of the user module have been created according to the Material Design Specifications using widgets from the android design support library version 7. Floating Labels, Buttons, Switch widgets from the design library have been used via TextInputLayout, AppCompatButton and SwitchCompat tags to enhance the user experience. Custom progress bar has been used to signify loading state. Each of the xml layouts are supported by a java activity class as given in the below table

XML Document	Activity Class
activity_signup.xml	SignUpActivity.java
activity_login.xml	LoginActivity.java
activity_userprofile.xml	UserProfileActivity.java

Table 16: User Module UI

The activity classes perform form user input validations for sign-up and login. And also the activity classes interact with the server using Asynchronous Http Client. The AsyncHttpClient library provides mechanisms to make asynchronous HTTP requests, handle responses in callback functions. The HTTP requests are made outside the UI thread. The invokeWebService() method in the activity classes is used to construct the http client request url by sending the Request Parameters via method parameters. The client is pre-set with any authorization headers if required. The HTTP Client Request method(POST/GET) is then invoked with the request url, Response handler.

As the server is set-up to accept and send only JSON payloads, the HTTP request call embeds a JSONHttpResponseHandler which overrides the onSuccess() and onFailure() methods.Depending on the success or failure of the client request call, the response HTTP status code and the response body are passed as parameters to either the OnSuccess() or OnFailureMethod. The OnSuccess() method performs the required operations on the response object and invokes the method call to perform the next

activity. The OnFailure() method gives out an appropriate error message depending on the error HTTP status code received and resets the current intent to accept user input again and make a new call.

Sign-up UI flow

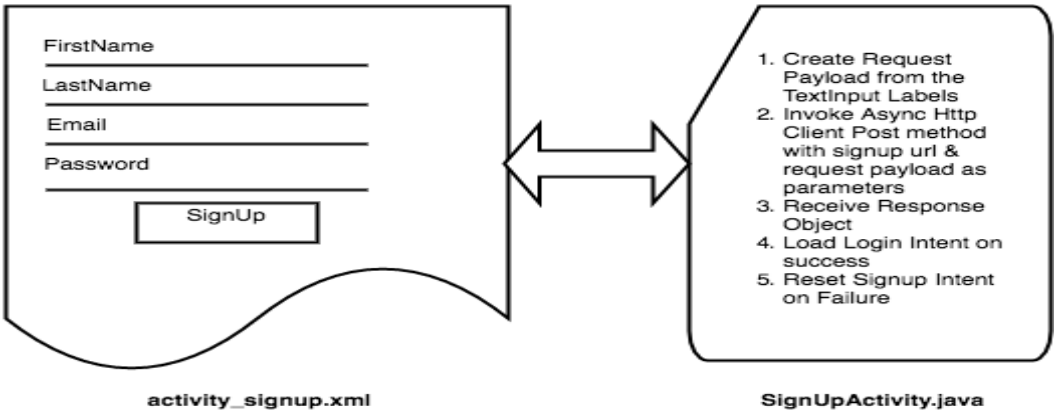


Figure 49: Sign-up UI Flow

Login Flow

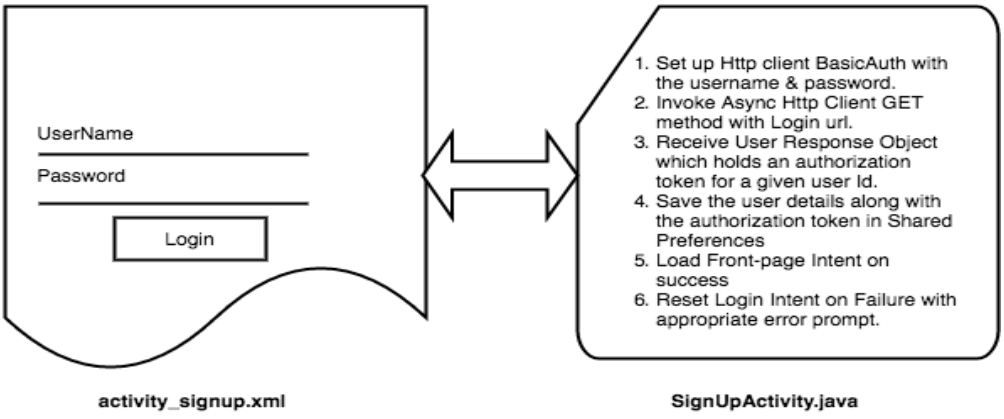


Figure 50: Login Flow

User Profile Flow

The User Profile screen is given as an action bar menu option on the front page. The action bar menu is set by overriding the onCreateOptionsMenu() method in the FrontPageActivity class. The click on the menu is registered by the OnOptionsItemSelected() listener method. Once the menu id for user-profile is clicked the intent for the UserProfile activity is invoked and the User Profile screen is displayed.

The User Profile screen also displays switch widgets imported from the design library along with the user details. These switches provide the mechanism through which the user allows notifications & recommendation emails to be sent.

The UserProfileActivity class implements the setOnCheckedChangeListener() method of the CompoundButton's class as switch inherits its attributes.

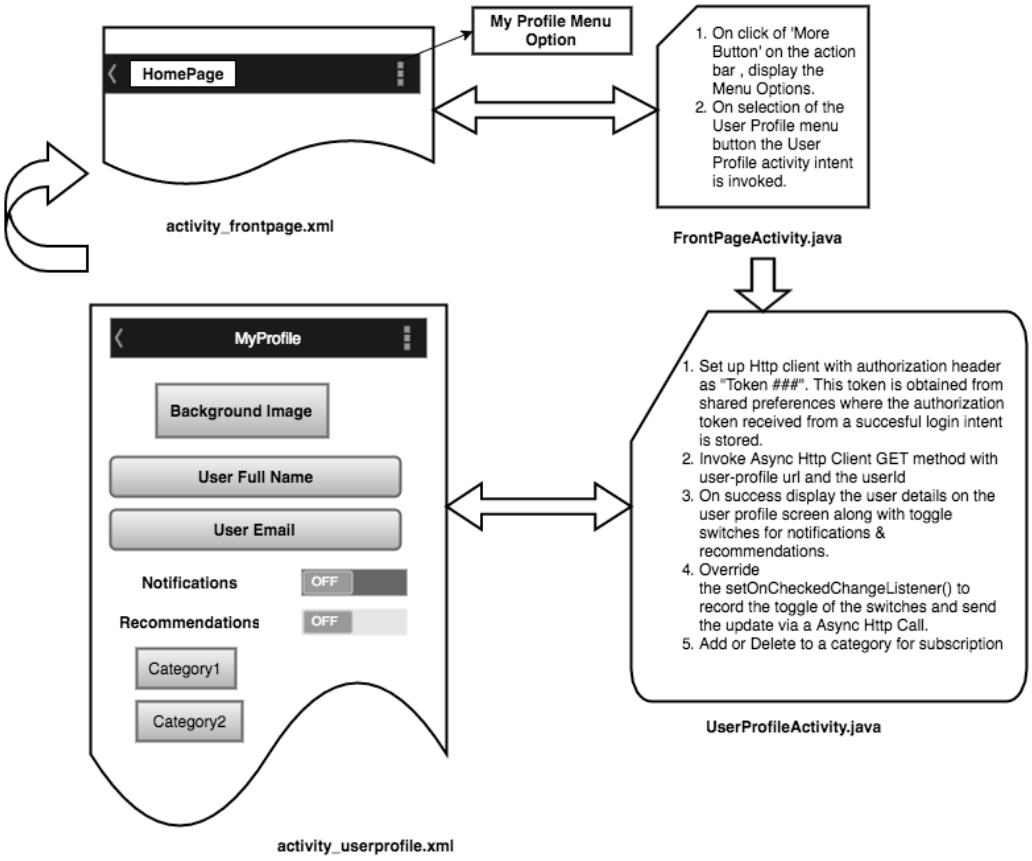


Figure 51: User Profile Flow

Front-page Flow

The front page UI has been implemented using the RecyclerView with a Staggered Grid Layout design to achieve the newspaper layout design. Each of the grid represents a data item i.e. either a post or an event. The data is loaded onto the View by means of a RecyclerView.Adapter class. The recycler view adapter requires a custom implementation of the data adapter. The adapter has three methods namely onCreateViewHolder() which initializes the View Holder and the other is onBindViewHolder() which is used to populate the current card view with data. The

images are loaded from the given s3 url using the Picasso library. Thus avoiding an async task to load the bitmap image after downloading it.

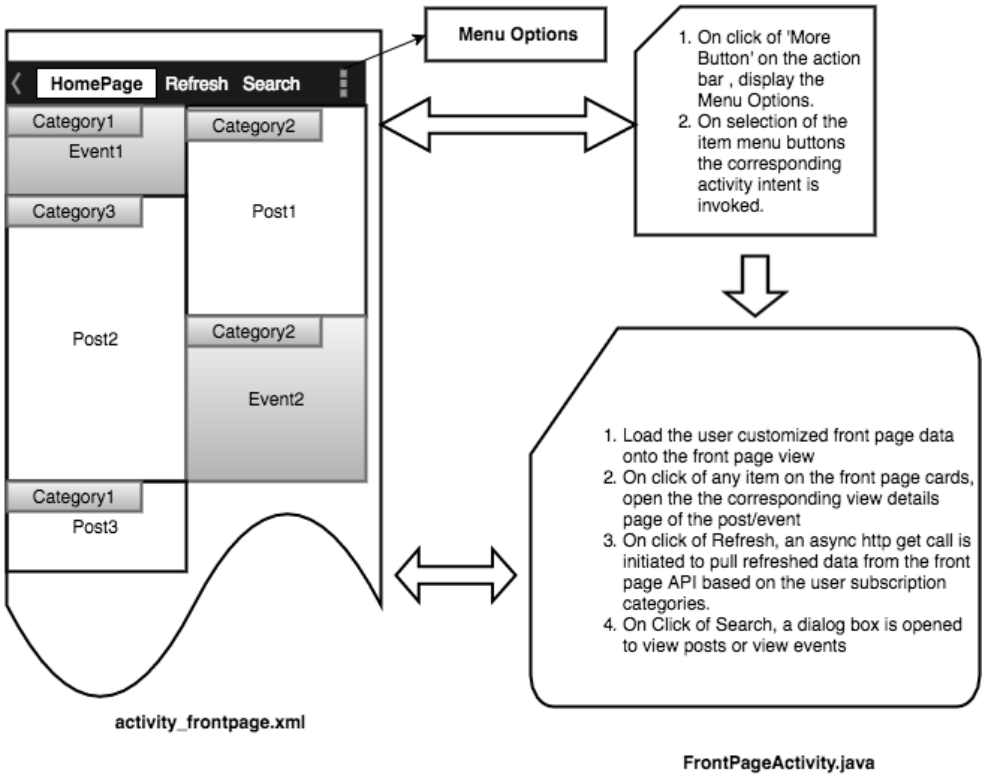


Figure 52: Front page UI Flow

Authorization Token storage in Shared Preferences

The Authorization Token generated from the User Authentication Service on the Server should be stored on the user device memory so that once the user logs into the app the first time, the subsequent visits to the app do not require a re-login. All the Http client requests hereafter should embed the auth token received from the server as an authorization header. This requires the token to be stored using the SharedPreference.Editor class and is made available to different modules of the android application using the PreferenceManager.

```
SharedPreferences sharedPreferences = PreferenceManager
        .getDefaultSharedPreferences(getApplicationContext());
String auth_token_string = sharedPreferences.getString("authToken", "");
String userId = sharedPreferences.getString("loggedInUserId", "");
client.addHeader("authorization","Token "+auth_token_string);
client.get(FEED_URL+"userId", new JsonHttpResponseHandler(){
```

Figure 53: Code Snippet 1

Post Module UI

The post module's client side implementation includes the screens that allow a user to create post, view post and react to a post. The following table shows the important activities and the fragments and their corresponding layout files that have developed to render the UI.

CreatePostActivity.java	activity_create_post.xml
ViewPostActivity.java	activity_view_post.xml
ViewPostFragment.java	fragment_view_post.xml
ViewPostButtonsFragment.java	fragment_view_post_buttons.xml
CommentListFragment.java	fragment_comments_list.xml
ViewCommentFragment.java	fragment_view_comment.xml
CommentListAdapter.java	fragment_comments_list_layout.xml

Table 17: Post Module UI Mapping

The create post activity opens when a user selects the create post option from the side menu. The activity includes input fields to enter details about the post and also allows a user to upload an image from the device gallery. On creation of the post, the user will be taken to the view post activity. Some of the UI components used in the activity are ScrollView, EditText, Spinner, etc. All the validation for the input fields are performed in the activity class.

While creating a post if a user selects an image, the path of the image is obtained and shown on the text field for image URL. On clicking the 'Create' button, the image gets uploaded to Amazon S3 bucket. If the API version is 23 or above, the permission to access the gallery is obtained during runtime. The libraries to support Amazon S3 were added to gradle dependencies.

compile**'com.amazonaws:aws-android-sdk-core:2.2.+'**
compile **'com.amazonaws:aws-android-sdk-s3:2.2.+'**

As part of implementation, we created a bucket in S3 and used the AWS access key id and the secret key to connect to S3. A PutObjectRequest reference is created and with this reference the image is added to the S3 bucket. On successful upload of image to S3, a presigned URL is generated for the image and this is passed to middle tier. This operation runs asynchronously in a thread different from the Main UI thread.

In the view post activity, the user can see the details of the post and also react on the post by voting, reporting, following or commenting. If the user who views the post is

also the creator of the post, then the user can edit or delete the post too. The user is also redirected to the view post activity, when he/she clicks on a post from the front page.

```
File file = new File(filePath);
picId=UUID.randomUUID().toString();
por= new PutObjectRequest( Util.BUCKET,picId , new java.io.File( filePath) );
(Thread) run() → {
        try {
            s3Client.putObject(por);
            ResponseHeaderOverrides override = new ResponseHeaderOverrides();
            override.setContentType("image/jpeg");
            GeneratePresignedUrlRequest urlRequest = new GeneratePresignedUrlRequest( Util.BUCKET, picId );
            Calendar cal = Calendar.getInstance();
            cal.set(2016, Calendar.DECEMBER, 15, 23, 00, 00); //Year, month, day of month, hours, minutes and seconds
            Date date = cal.getTime();
            urlRequest.setExpiration(date);
            urlRequest.setResponseHeaders(override);
            picurl = s3Client.generatePresignedUrl( urlRequest );
            System.out.println(picurl);
            uploadComplete=true;
        } catch (Exception e) {
            e.printStackTrace();
        }
}.start();
```

Figure 54: Code Snippet 2

The view post activity is composed of 4 fragments - fragment to enable viewing and editing a post, fragment to vote/report/follow a post, a fragment to display the comments for the post and a fragment to show a comment. If the post consists of an image, the image is downloaded from S3 with the available URL and rendered. This code is implemented as an Async Task.

```
private class GetImage extends AsyncTask<String, Void, Bitmap> {

    @Override
    protected Bitmap doInBackground(String... urls) {
        Bitmap map = null;
        map = downloadImage(urls[0]);
        return map;
    }

    // Sets the Bitmap returned by doInBackground
    @Override
    protected void onPostExecute(Bitmap result) {
        postImage.setImageBitmap(result);
        System.out.println("finished");
    }

    // Creates Bitmap from InputStream and returns it
    private Bitmap downloadImage(String url) {
        Bitmap bitmap = null;
        InputStream stream = null;
        BitmapFactory.Options bmOptions = new BitmapFactory.Options();
        bmOptions.inSampleSize = 1;

        try {
            stream = getHttpConnection(url);
            bitmap = BitmapFactory.
                    decodeStream(stream, null, bmOptions);
            stream.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        return bitmap;
    }
```

Figure 55: Code Snippet 3

Some of the UI components used for this activity are ImageView, ListView, ImageButton, etc. Clicking on a comment item, opens a dialog to show the comment contents and also allows to edit or delete the comment if the user is also the creator of the comment. The list of comments for this screen are displayed with the help of a custom list adapter.

```
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        convertView = layoutInflater.inflate(R.layout.fragment_comment_list_layout, null);
        holder = new ViewHolder();
        holder.creator = (TextView) convertView.findViewById(R.id.creator);
        holder.commentVal = (TextView) convertView.findViewById(R.id.commentVal);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }

    if(position %2 == 1)
    {
        convertView.setBackgroundColor(Color.parseColor("#006bb3"));
    }
    else
    {
         convertView.setBackgroundColor(Color.parseColor("#005c99"));
    }

    holder.creator.setText("Created By "+commentsList.get(position).getUserId());
    holder.commentVal.setText(commentsList.get(position).getCommentContent());
    return convertView;
}
```

Figure 56: Code Snippet 4

The activity classes and the fragment classes make the server calls using Asynchronous HTTP Client. The AsyncHTTPClient library provides all the methods to make asynchronous HTTP calls and also provides success handlers and failure handlers depending on the response obtained from the backend. If the server returns a JSON response, we use the JSONHttpResponseHandler and otherwise we use AsyncHttpResponseHandler.

The following screenshots show the layout for create post and view post activities respectively.

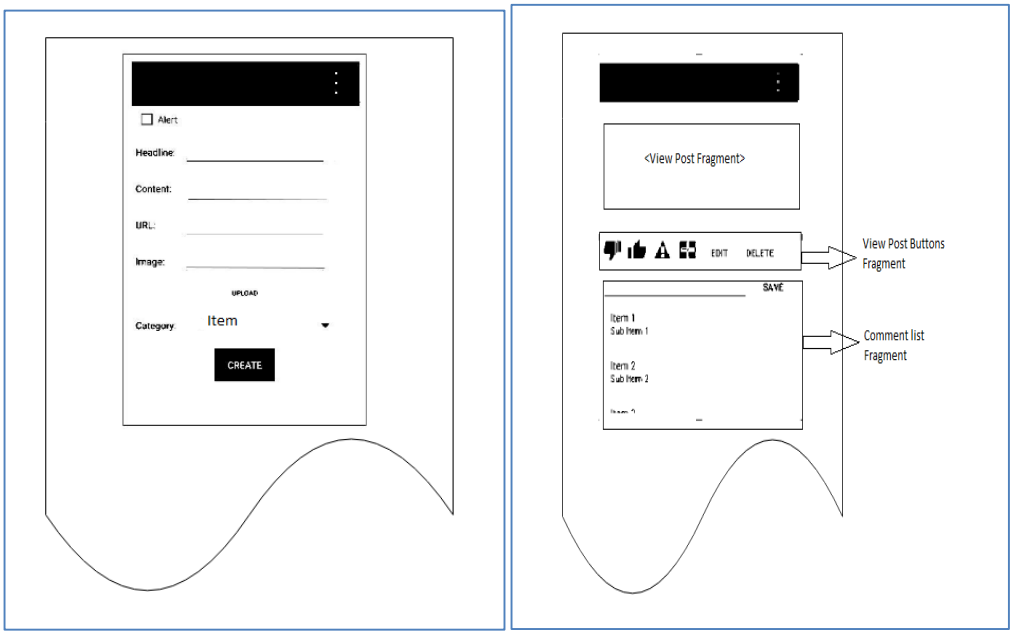


Figure 57: Layout for Create and View Post activities

Event Module UI

Event module broadly includes three functionalities, namely create an event, view an event's details and view the list of all the upcoming events. The implementation of each of these functionalities is described below

Create Event

Create event functionality is implemented using CreateEventActivity.java as the activity class and create_event_main.xml as the activity layout. The activity layout contains an empty fragment container that is used to add, replace and show the fragments related to the create event functionality, using the fragment manager.

The table below shows the list of fragments used by CreateEventActivity with a brief description of the purpose of their use.

Fragment Class Name	Type of Fragment	Layout	Purpose of Use
CreateEventTitleNDesc	Custom Fragment	create_event_title_n_desc.xml	Displays first page of the event creation activity
CreateEventMoreDetails	Custom Fragment	create_event_more_details.xml	Displays second page of the event creation activity

DatePickerFragment	Dialog Fragment	Uses DatePickerDialog	Allows the user to choose a date
TimePickerFragment	Dialog Fragment	Uses TimePickerDialog	Allows the user to choose a time

Figure 58: List of Fragments in Create Event Activity

CreateEventActivity validates the user input data when the user navigates from the first page to second. If the validation fails, the first page fragment is not replaced by the second page fragment. Similarly, CreateEventActivity validates the user input data on the second page when the user clicks create. If the data is valid, CreateEventActivity starts ViewEventActivity and finishes, else the activity stays on the second page. DatePickerFragment and TimePickerFragment are shown when the user chooses to select a date or time on the second page.

The diagram below shows the fragment transactions during the event creation activity.

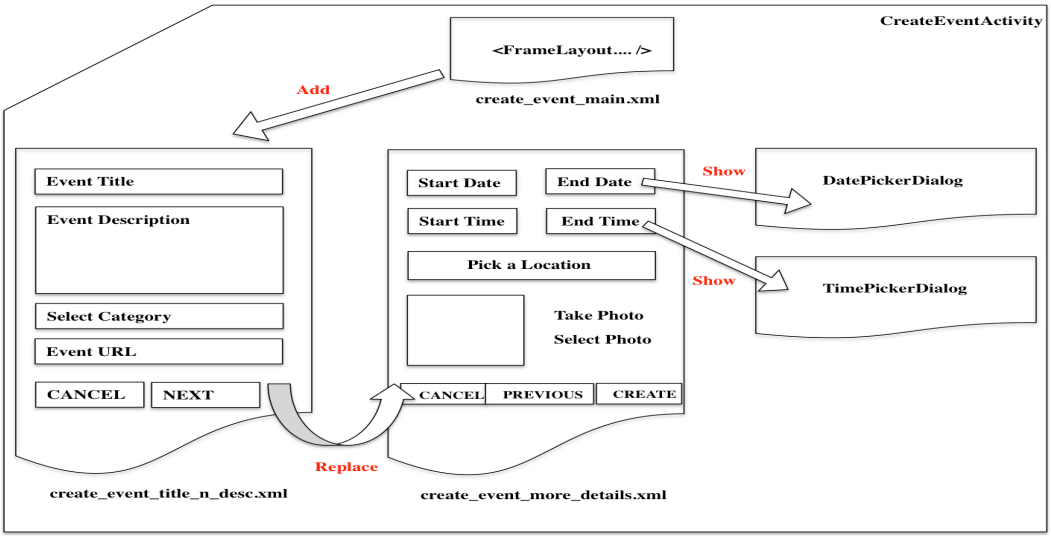


Figure 59: Fragment transactions during event creation activity

The code snippet below demonstrates the method defined in CreateEventActivity to navigate from the first page to the second page using the fragment manager.

```
/**
 * @param view
 * Method to navigate to the next fragment
 */
@Override
public void goToEventMoreDetails(View view) {

    ViewGroup viewGroup = (ViewGroup) view.getRootView();
    EditText editText = (EditText)viewGroup.findViewById(R.id.edit_event_title);
    this.eventTitle = editText.getText().toString();

    if(validateEventFirstPage(viewGroup) && findViewById(R.id.fragment_container)!=null){

        Fragment fragment_more_details = new CreateEventMoreDetails();

        FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
        transaction.replace(R.id.fragment_container, fragment_more_details, "create_event_second_page");
        transaction.addToBackStack(null);

        transaction.commit();

    }
}
```

Figure 60: Code Snippet 5

The code snippet below demonstrates the listener method used in CreateEventActivity to show a date picker. Time picker fragment is shown using the similar instructions.

```
public void datePickerListener(View view){
    if(findViewById(R.id.fragment_container)!=null){

        DatePickerFragment fragment_date_picker = new DatePickerFragment();
        Bundle args = new Bundle();
        args.putInt("clicked_date_button", view.getId());
        fragment_date_picker.setArguments(args);

        FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
        transaction.addToBackStack(null);
        fragment_date_picker.show(transaction, "datePicker");
    }
}
```

Figure 61: Code Snippet 6

CreateEventActivity also starts another activity for performing certain steps. The interaction of CreateEventActivity with other activities is described below.

- **Pick a Location**: CreateEventActivity uses built-in place picker UI widget of Google Places API for selecting the location of an event. This requires to request ACCESS_FINE_LOCATION user permission, which is checked at the application runtime. To use the place picker UI widget, an Intent is created using PlacePicker.IntentBuilder() and passed as the parameter to startActivityForResult() method of CreateEventActivity. A unique request code is also passed to startActivityForResult(), in order to identify the corresponding response received by onActivityResult() method of CreateEventActivity.

```
private void showPlacePicker(){

    PlacePicker.IntentBuilder builder = new PlacePicker.IntentBuilder();
    builder.setLatLngBounds(CreateEventActivity.BOUNDS_MOUNTAIN_VIEW);
    try {
        startActivityForResult(builder.build(this), CreateEventActivity.PLACE_PICKER_REQUEST);
    }catch(GooglePlayServicesNotAvailableException ex){

    }catch (GooglePlayServicesRepairableException ex){

    }
}
```

Figure 62: Code Snippet 7

- **Take Photo**: CreateEventActivity allows the user to add an image to the event by capturing photo from the device's camera. This requires starting an activity with Intent having MediaStore.ACTION_IMAGE_CAPTURE as action. In order to use the full size of the image, a temporary file is created in the external storage of the application and it's URI is passed to the created Intent using Extras. The external storage of the application is accessed using Environment.*getExternalStoragePublicDirectory*(Environment.*DIRECTORY_PICTURES*) and requires WRITE_EXTERNAL_STORAGE user permission to write to the storage.

```
private void startTakePicFromCameraIntent(){
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if(takePictureIntent.resolveActivity(getPackageManager())!=null){
        File imageFile = null;
        try{
            imageFile = createFileForImage();

        }catch (IOException ex){
            System.out.println("Exception : "+ ex.getMessage());
        }

        if(imageFile!=null){
            Uri imageURI = Uri.fromFile(imageFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageURI);
            startActivityForResult(takePictureIntent, CreateEventActivity.CAPTURE_IMAGE_REQUEST);
        }
    }
}
```

Figure 63: Code Snippet 8

- **Select Photo**: CreateEventActivity also allows users to select a photo from the device's photo gallery and add it to the event. This requires starting an activity with Intent having Intent.ACTION_GET_CONTENT. The selected image's URI is received in the data of the intent accessed in onActivityResult() method.

```
@Override
public void pickPhotoFromGalleryListener(View view) {
    Intent pickPhotoIntent = new Intent();
    pickPhotoIntent.setType("image/*");
    pickPhotoIntent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(pickPhotoIntent, "Select picture"),
            CreateEventActivity.PICK_PHOTO_FROM_GALLERY_REQUEST);
}
```

Figure 64: Code Snippet 9

The diagram below illustrates the interaction of CreateEventActivity with other activities.

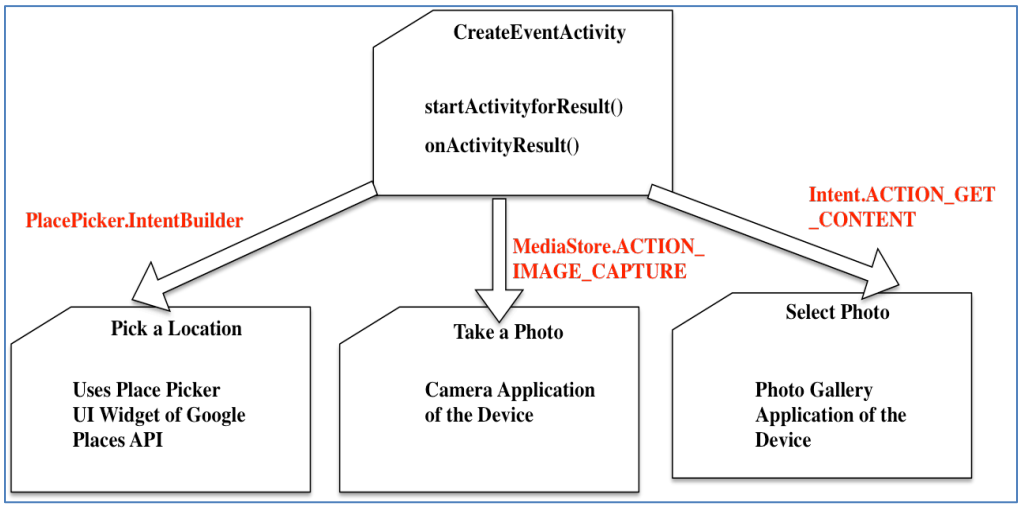


Figure 65: Interactions of Create Event Activity

View Event

View event functionality is implemented using ViewEventActivity.java as the activity class and view_event_details.xml as the activity layout. The view event activity uses map fragment of Google Maps APIs to show the location of the event on a map object. The activity implements OnMapReadyCallback interface to use the callback method onMapReady() and uses getMapAsync() to register the callback with the map fragment.

The view event details layout also allows a user to add comments to the event. The operations related to the comments are managed using a PopupWindow, which uses a different layout using event_comments.xml file. The layout includes a ListView for displaying all the existing comments. The ListView uses a custom adapter named EventCommentsAdapter.java for generating each item view. The adapter uses each_event_comment.xml as the layout of each item view.

The diagram below shows the different layouts used by ViewEventActivity.

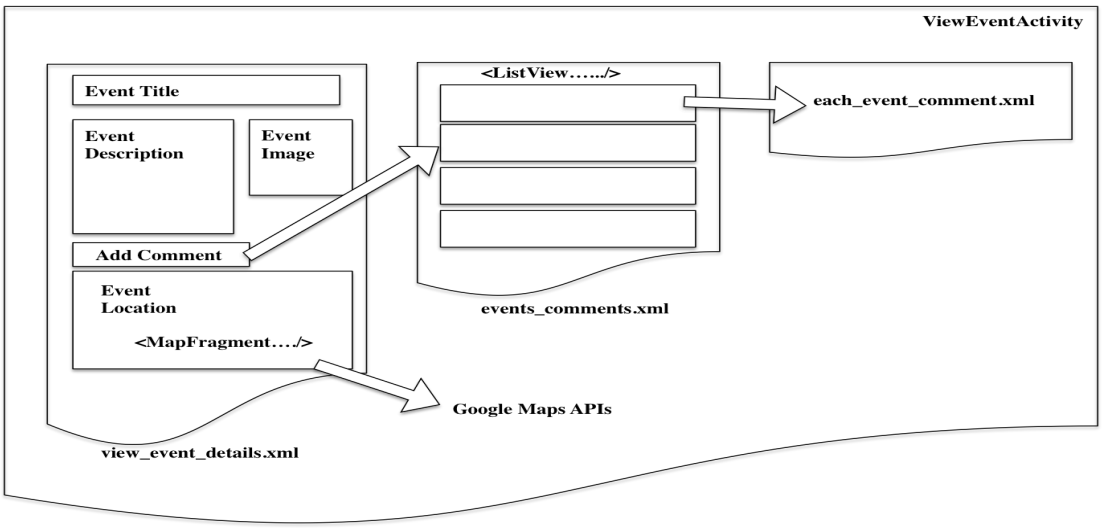


Figure 66: Layouts used by ViewEventActivity

ViewEventActivity interacts with Calendar application of the device for allowing the user to add the event to his calendar. ViewEventActivity uses the intent with action Intent.ACTION_INSERT to add an event to the calendar. The details of the event are passed to the Calendar application using Extras.

View All Events

ViewAllEventsActivity.java is used to list all the upcoming events using view_all_events.xml layout, which contains an empty ListView element. The activity uses a custom adapter named ViewEventAdapter.java to create the item view for each event in the list. The adapter uses view_event.xml for creating the layout of each item.

The diagram below shows the layouts used by ViewAllEventsActivity.

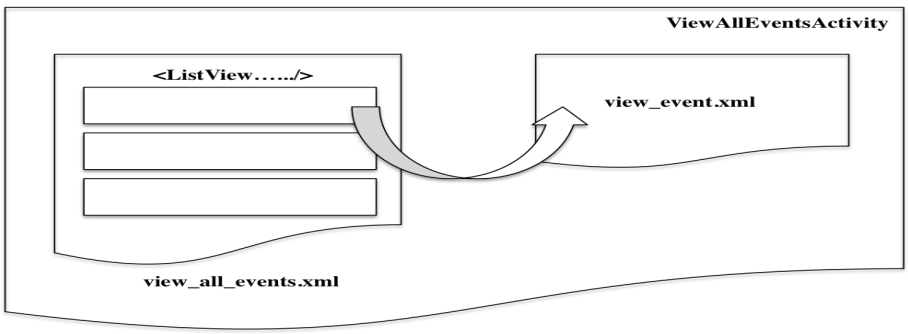


Figure 67: Layouts used by ViewAllEventsActivity

Search Flow

The Search icon is made available on the action bar on every screen. The click on the search Icon opens a dialog box for the user to select the Type of search - Events or Posts. On click of Events radio button, the user is navigated to the ViewAllEvents Screen with a Search View on the top to enter keywords for searching. On click of Posts radio button, the user is navigated to the ViewPostsByCategoryPage, where the user can apply a filter by Category, which then displays a screen with lists of posts with a search view bar to enter keywords for searching.

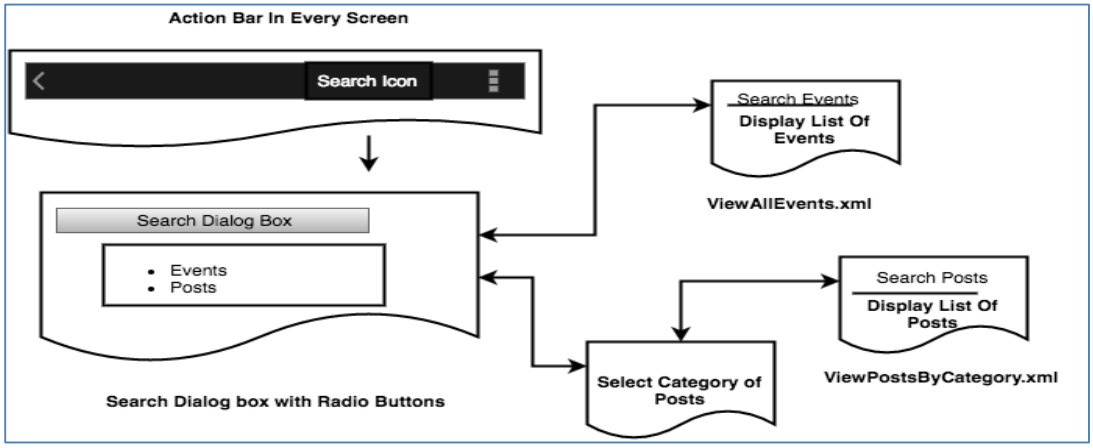


Figure 68: Search UI Flow

Recommendation Email

Deeplinking of application from device email app

Intent Filter Specification in the androidmanifest.xml :

```
166
167  //AndroidManifest.xml
168  <activity android:name=".DeepLinkActivity">
169      <intent-filter>
170          <action android:name="android.intent.action.VIEW" />
171          <category android:name="android.intent.category.DEFAULT" />
172          <category android:name="android.intent.category.BROWSABLE" />
173          <data android:scheme="myRecommendations" android:host="campustribune.com"/>
174      </intent-filter>
175  </activity>
176
```

Figure 69: Code Snippet 10

JSON Configuration for deeplinking:

This json file provides the mapping of incoming deeplink URLs to specific screens in the campus tribune app.It also provides the configurations for custom callback handlers, validation, and logging.

```
{
  "defaultRoute": {
    "class": "com.campustribune.frontpage.FrontpageActivity"
  },
  "routes": {
    "posts/:postid": {
      "class": "com.campustribune.post.viewPostActivity"
    },
    "events/:eventid": {
      "class": "com.campustribune.event.viewEventActivity"
    }
  }
}
```

Figure 70: Code Snippet 11

Deep Linking Filter activity class:

```
public class DeepLinkActivity extends ActionBarActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.deep_link);
    TextView deepLinkUrl = (TextView) findViewById(R.id.deep_link_url);
    Intent intent = getIntent();
    Uri data = intent.getData();
    if (data.getQueryParameter("postid") != null) {
      Support.showPost(this, data.getQueryParameter("postid"));
      finish();
    } else if (data.getQueryParameter("eventid") != null) {
      Support.showEvent(this, data.getQueryParameter("eventid"));
      finish();
    } else {
      deepLinkUrl.setText("Deep link received - " + data);
    }
  }
}
```

Figure 71: Code Snippet 12

GCM Client

The client apps must register with to verify that they can send and receive messages. The client receives a unique registration token and passes it on to the app server. The app server then stores the token and will send an acknowledgement back to the client

app. The client app instance identifier that the app server uses to send messages to the particular client is the same registration token exchanged in this process.

For registering with the GCM, the client app gets a registration token using the Instance ID API. An authorized entity set to the app server's sender ID is needed to call the API and the scope is set to the value for GCM appropriately chosen depending on the platform. The registration token to the app server is passed by the client app. The registration token is saved by the app serve rand the client app is acknowledged that the process is successfully completed.

GoogleCloudMessaging API and Android Studio with Gradle is recommended for the implementation of the client. A configuration file is downloaded by providing the package information and the SHA fingerprint. The file is copied into the configuration of the project. Plug in's required are added to the gradle file. Manifest file is updated with all the service files.

Instance ID API provided by Google can be used to handle the creation and updating of registration tokens. InstanceIDListenerService is defined in order to use the API. To receive a token, we call the instanceID.getToken, providing the app server's sender ID.

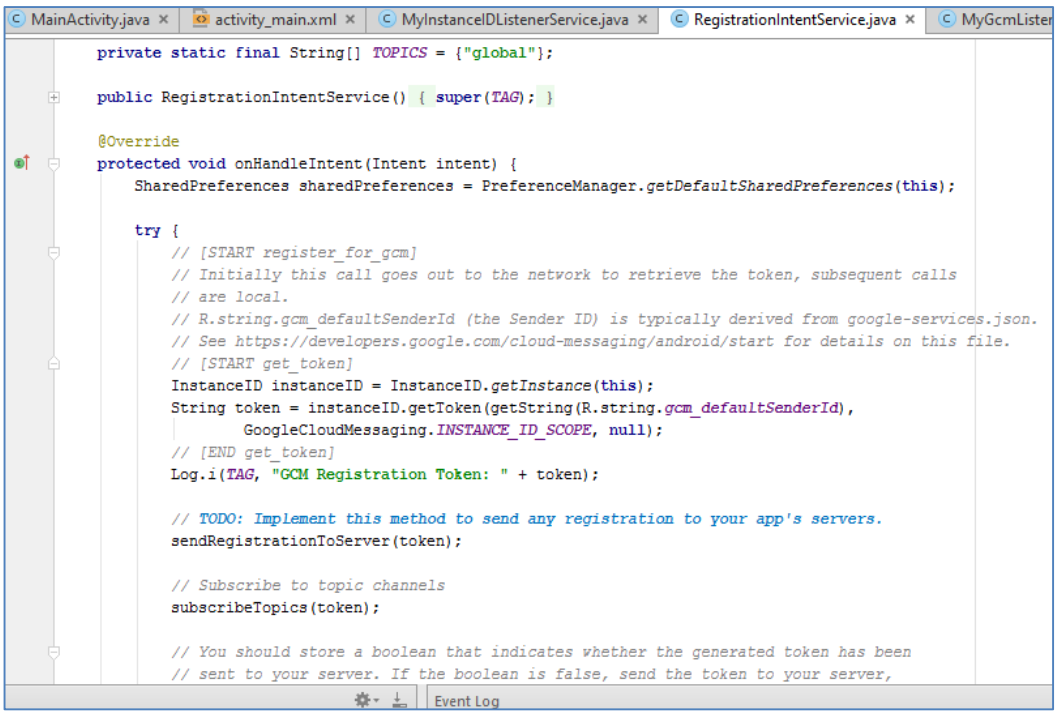


Figure 72: Code Snippet 13

Once the registration token is received we need to send it to the server. The listener service's method. onTokenRefresh() method should be invoked if the GCM registration token has been refreshed as shown above. After the client app is connected, we can start receiving downstream messages and sending upstream messages.

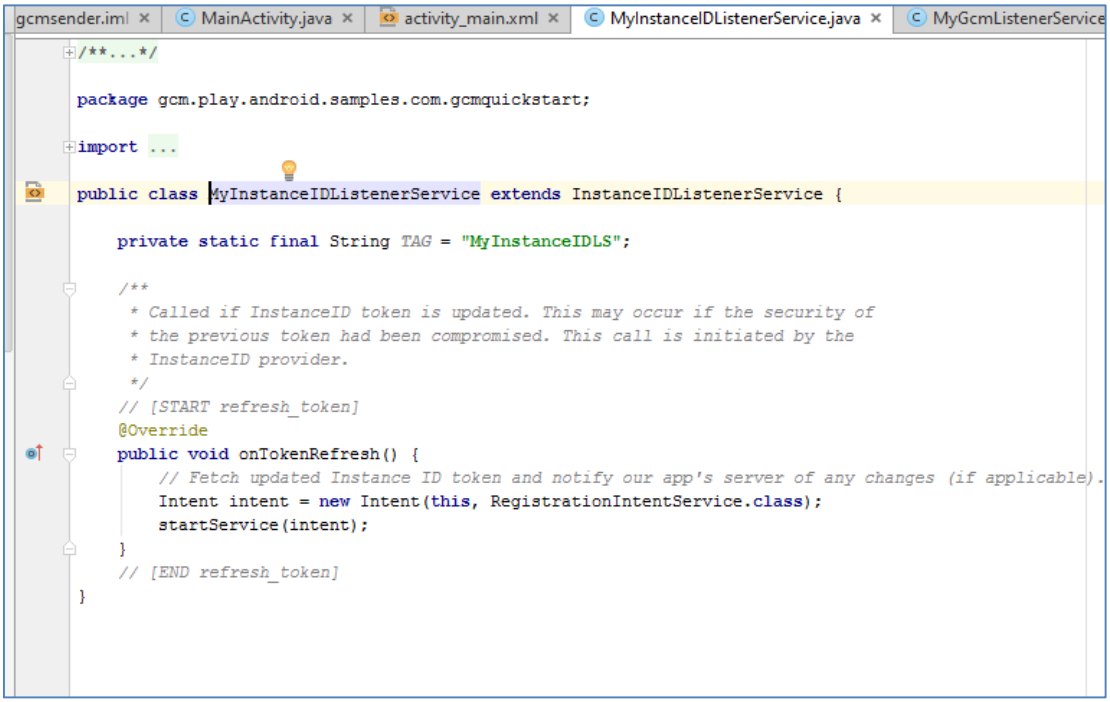


Figure 73: Code Snippet 14

Middle-Tier Implementation

The middle tier of the Campus Tribune application is implemented using the Spring Boot Framework in Java. The main advantage of using the spring boot is that it simplifies spring dependency injections and also eliminates the need of an application container. The application jar can be built directly from the command line and requires only JDK installed on the instance. The services provided by the application are exposed to the android client via REST API calls. The REST endpoints consumes and produces data using JSON and HTTP Status codes. The application uses the Gradle build tool. The build.gradle file includes all the all the dependencies that are downloaded when the app is built. The dependencies used in the backend applications include Spring data mongodb, spring data rest, spring web and spring security configurations. It also includes the Java Messaging Services (JMS) library for email services, Google's Guava library for caching purposes. The Spring data mongodb configuration enables the access of the MongoDB server via the spring data mongo uri

specified in the application.properties file in the source resources folder. The Data Access Objects(DAO) interact with their corresponding document collections in MongoDB via the Repository interfaces that extend the MongoRepository class in the spring framework.

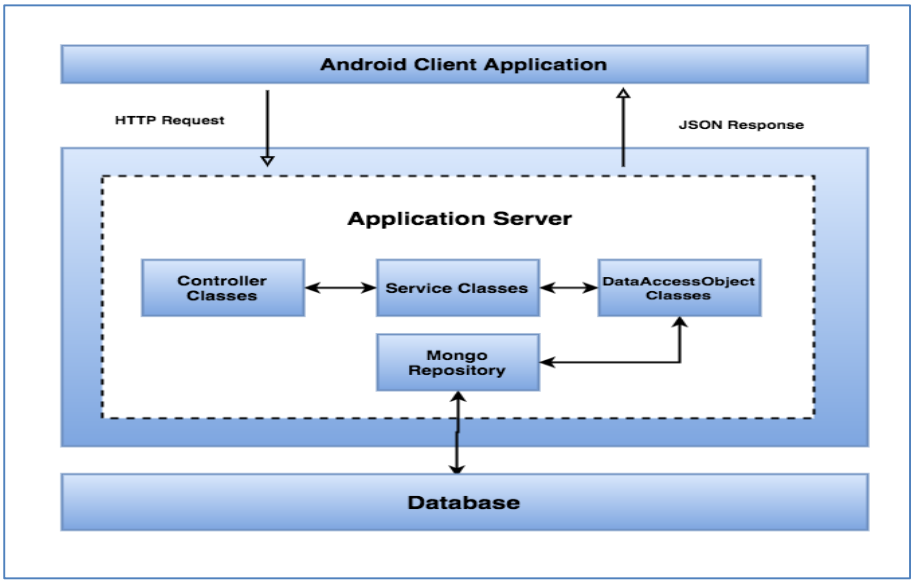


Figure 74: Middle Tier Implementation

User Authentication

We use Spring Security to initially authenticate the user, and subsequently use session tokens to authorize requests from that user. Initially the login request comes in the form of a REST GET request on the endpoint: /user/login. This is intercepted by the class: WebSecurityConfig.java which extends Spring's WebSecurityConfigurerAdapter, and is responsible for authenticating the user's login credentials. We use a class called AuthProvider.java for comparing the user's login credentials with what is stored on the MongoDB database.

Once the user is authenticated, then the method mapped to the /user/login endpoint in the REST controller: UserController.java is executed. The controller then invokes the getAuthenticatedUser() method in the service class: UserService.java. This method maps the UserDAO.java object coming from the MongoDB Repository to the User.java object that will be returned

```java
public class AuthenticationFilter extends ChannelProcessingFilter {

    private final AuthHelper authHelper;

    public AuthenticationFilter(AuthHelper authHelper) {
        this.authHelper = authHelper;
    }
    @Override
    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) res;
        HttpServletRequest request = (HttpServletRequest) req;
        if (WebSecurityConfig.AUTHENTICATION_PATH.equals(request
                .getServletPath())) {
            chain.doFilter(req, res);
            return;
        }
        String authToken = extractToken(request);

        if (checkToken(authToken)) {
            chain.doFilter(req, res);
        } else {
            writeErrorResponse(response, HttpServletResponse.SC_UNAUTHORIZED,
                    "Invalid or missing authorization token");
        }
    }
```

Figure 75: Code Snippet 15

as the API response. It also creates the authorization token and saves it in the Authorization Token Cache. The AuthHelper.java class is a helper class that provides methods helpful to retrieve authentication information & to manage authorization tokens. Finally, the User.java object is returned as reponse to the login API.

Subsequently, the client will send the generated authorization token as a header value. The application must validate this token before executing the API endpoint. This is done in the AuthenticationFilter.java class, which filters all the requests coming into the application, identifies the token and then validates the user. If the token is expired, the client will have to re-authenticate and obtain a new token. If the token is invalid, then a HTTP 401 (unauthorized) status code is returned. If the token is valid, then the AuthenticationFilter.java class allows the request to be executed.

```java
@Service
public class AuthHelper {

    @Autowired
    private UserService userService;

    public static final int TOKEN_EXPIRATION_PERIOD_IN_MINUTES = 12 * 60;

    private final Cache<String, Authentication> tokenStore;

    @Autowired
    public AuthHelper(@Value("${user.cacheSize:1000}") int userCacheSize) {
        tokenStore = CacheBuilder
                .newBuilder()
                .maximumSize(10000)
                .expireAfterWrite(TOKEN_EXPIRATION_PERIOD_IN_MINUTES,
                        TimeUnit.MINUTES).build();
    }

    public Authentication getAuthentication() {
        final SecurityContext securityContext = SecurityContextHolder
                .getContext();
        return securityContext.getAuthentication();
    }

    public String getUsername() {
        final Authentication authentication = getAuthentication();
        return authentication != null ? authentication.getName() : null;
    }

    public void saveToken(String token) {
        tokenStore.put(token, getAuthentication());
    }

    public boolean checkToken(String authToken) {
        Authentication authentication = authToken != null ? tokenStore
                .getIfPresent(authToken) : null;
        final SecurityContext securityContext = SecurityContextHolder
                .getContext();
        securityContext.setAuthentication(authentication);
        return authentication != null;
    }

}
```

Figure 76: Code Snippet 16

Post Module

The backend code consists of controllers (PostController.java, PostCommentsController.java) to redirect the REST API calls to the relevant service classes (PostService.java and PostCommentsService.java). The service classes make the required database calls by using the data access objects to pass data to and from database). The backend implementation also involves the reputation logic developed for voting mechanism, the report logic and the implementation of spam filter algorithm.

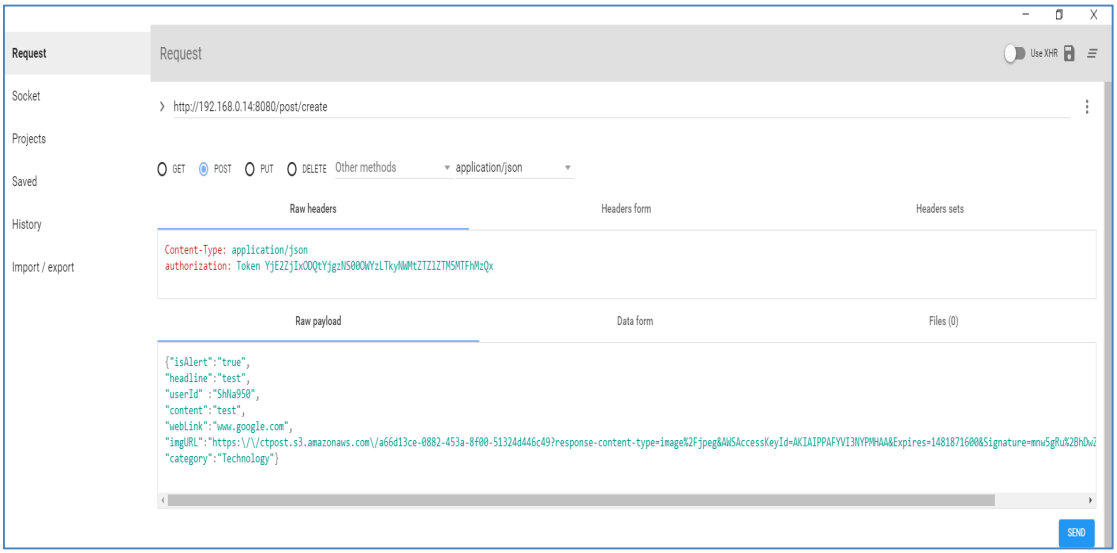


Figure 77: Post Request 1

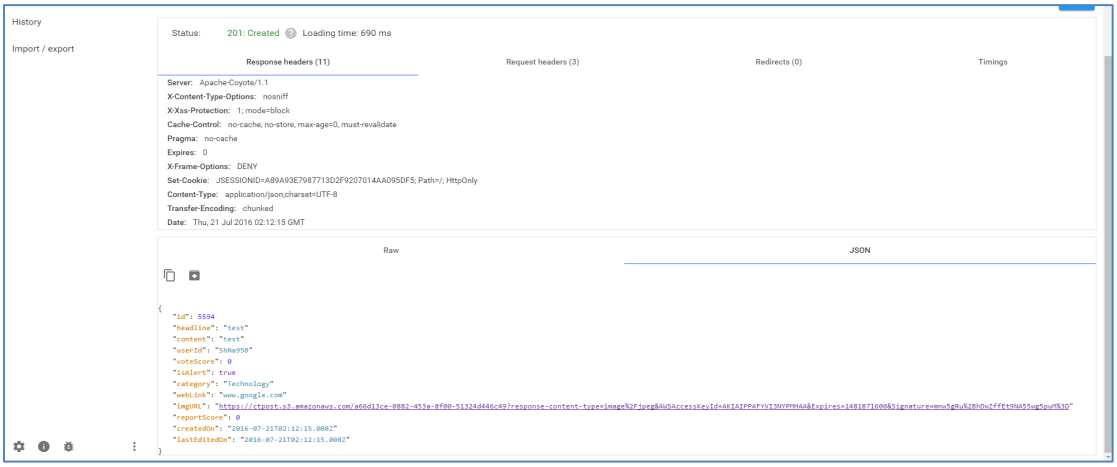


Figure 78: Post Response 1

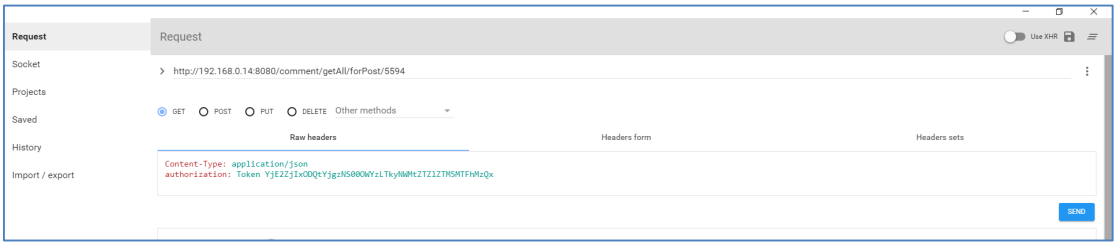


Figure 79: Post Request 2

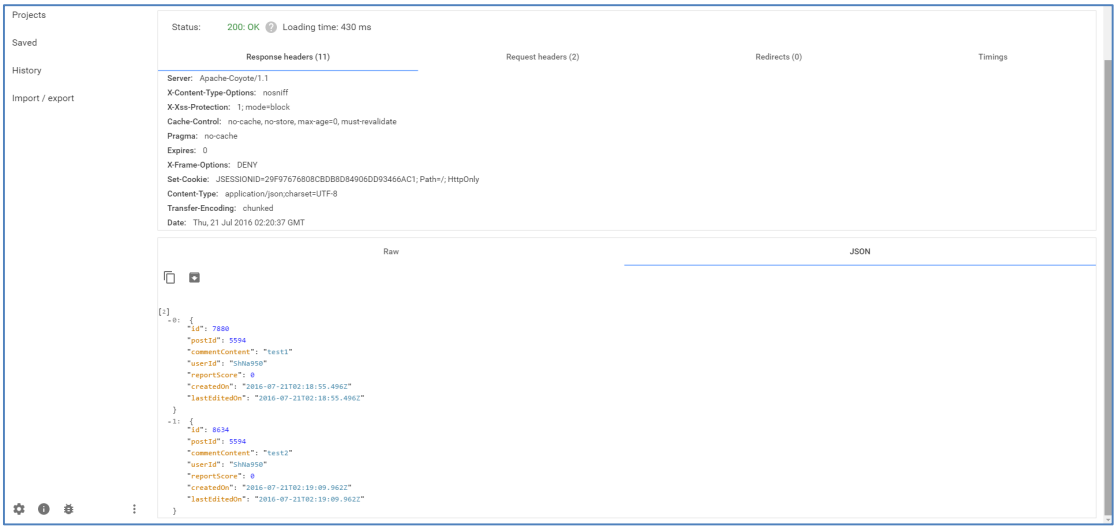


Figure 80: Post Response 2

```
public ResponseEntity<PostComment> createComment(@RequestBody @Valid PostComment postComment) {
    if (postService.postExists(postComment.getPostId())) {
        postComment = postCommentService.createComment(postComment);
        if (postComment != null) {
            return new ResponseEntity<PostComment>(postComment, HttpStatus.CREATED);
        } else {
            return new ResponseEntity<PostComment>(HttpStatus.CONFLICT);
        }
    } else {
        return new ResponseEntity<PostComment>(HttpStatus.NOT_FOUND);
    }
}

@RequestMapping(value = "/get/{comment_id}/forPost/{post_id}", method = RequestMethod.GET, produces = "application/json")
public ResponseEntity<PostComment> viewComment(@PathVariable("comment_id") int comment_id,
        @PathVariable("post_id") int post_id) {
    if (postService.postExists(post_id) && postCommentService.commentExists(comment_id)) {
        return new ResponseEntity<PostComment>(postCommentService.getComment(comment_id, post_id), HttpStatus.OK);
    } else {
        return new ResponseEntity<PostComment>(HttpStatus.NOT_FOUND);
    }
}

@RequestMapping(value = "/getAll/forPost/{post_id}", method = RequestMethod.GET, produces = "application/json")
public ResponseEntity<ArrayList<PostComment>> listAllComments(@PathVariable int post_id) {
    if (postService.postExists(post_id)) {
        return new ResponseEntity<ArrayList<PostComment>>(postCommentService.getAllComments(post_id),
                HttpStatus.OK);
    } else {
        return new ResponseEntity<ArrayList<PostComment>>(HttpStatus.NOT_FOUND);
    }
```

Figure 81: Code Snippet 17

Events Module

The APIs for the CRUD operations related to the event module are implemented using org.springframework.data.mongodb library of Spring framework. The mongodb library provides easy access to mongodb collections through MongoRepository interface. The interface provides many pre-defined data queries and new custom queries can also be added easily. In order to insert (or access) data to (or from) the three collections related to the event module, the following three interfaces are defined.

```
@Service
public interface IEventRepository extends MongoRepository<EventDAO, UUID> {

    public List<EventDAO> findFirst10ByOrderByCreatedOnDesc();
}



public interface IEventCommentsRepository extends MongoRepository<EventCommentsDAO, UUID> {

    public List<EventCommentsDAO> findByEventId(UUID eventId);
    public void deleteByEventId(UUID eventId);
}



public interface IEventUserRepository extends MongoRepository<EventUserDAO, String> {

}
```

Figure 82: Code Snippet 18

The custom queries are added for events and eventComments collections, as shown above. The code snippet below shows an example of using the custom query findFirst10ByOrderByCreatedOnDesc() of events repository in the service class for fetching the top 10 upcoming events.

```
public ArrayList<Event> viewEvents(){
    List<EventDAO> events = new ArrayList<>();
    events = eventRepository.findFirst10ByOrderByCreatedOnDesc();
    if(events.size()>0){
        ArrayList<Event> listOfEvents = new ArrayList<>(events.size());
        for(EventDAO each: events){
            Event eachEvent = new Event(each.getId(), each.getTitle(), each.getCategory(),
                    each.getDescription(), each.getUrl(), each.getStartDate(), each.getEndDate(),
                    each.getLatitude(), each.getLongitude(), each.getAddress(), each.getEventImageS3URL(),
                    each.getUpVoteCount(), each.getDownVoteCount(), each.getGoingCount(), each.getNotGoingCount(),
                    each.getCreatedBy(), each.getCreatedOn());
            eachEvent.setListOfComments(getAllComments(eachEvent));
            listOfEvents.add(eachEvent);
        }
        return listOfEvents;
    }
    else
        return null;
}
```

Figure 83: Code Snippet 19

The viewEvents method of the service class is invoked by the event controller class to request the list of the upcoming events. The screenshots below shows the request and response for fetching the events.

View All Events

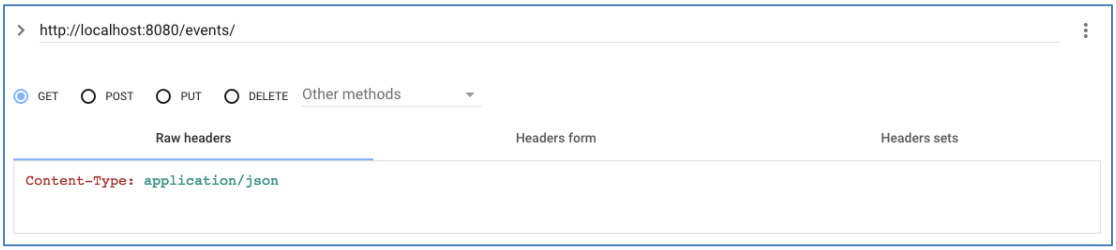


Figure 84: Event Request

```
[7]
 -0: {
     "id": "e389a833-3681-4822-aa21-67d30a79ec17"
     "title": "Turtle Ninja Fun Event"
     "category": "Entertainment"
     "description": "Fun event for new students"
     "url": null
     "startDate": "2016-07-25T09:00:00.000Z"
     "endDate": "2016-07-27T21:00:00.000Z"
     "latitude": -32.567
     "longitude": 456.789
     "address": "101 E San Fernando, San Jose, CA 95113"
     "eventImageS3URL": null
     "upvoted": false
     "downvoted": false
     "follow": false
     "going": false
     "notGoing": false
     "upVoteCount": 1
     "downVoteCount": 0
     "goingCount": 1
     "notGoingCount": 0
     "createdBy": "harry12"
     "createdOn": "2016-07-21T10:30:00.000Z"
    -"listOfComments": [1]
       -0: {
            "id": "9aab03ff-016e-4970-b4a4-89708acfd386"
            "eventId": "e389a833-3681-4822-aa21-67d30a79ec17"
            "createdBy": "harry12"
            "createdOn": "2016-07-21T10:39:00.000Z"
            "comment": "I hope to see you all there!!"
            "reportedBy": null
          }
 }
 -1: {
     "id": "892abdbe-9147-48a3-b914-c86dda02abc7"
     "title": "San Francisco Free Icecream Day"
     "category": "Entertainment"
     "description": "Free icecream for all the students and professionals. Students please bring your student id along!!"
     "url": null
     "startDate": "2016-07-30T09:00:00.000Z"
     "endDate": "2016-07-30T13:00:00.000Z"
     "latitude": -32.567
     "longitude": 456.789
     "address": "Market Street, San Francisco"
     "eventImageS3URL": null
```


Figure 85: Event Response

Front-Page algorithm

The front page algorithm pull up a list of Posts & Events based on the categories that the user has subscribed to and filters the lists based on the front page criterion as given below.

Post Analysis:
Algorithm:

1. Obtain user subscribed category list
2. Accumulate Posts in each of the above categories based on the last edited on date
3. For each post, calculate
    a. votescore
    b. no.of comments made
    c. no.of people following the post
4. Calculate the cumulative score of each post
        1. Find age - no. of hrs since the post was created
        2. Subtract the age from the score accumulated
5. Sort the list of posts accumulated based on the cumulative score
6. Add the top 3 posts (if available) to the Front page post list.
7. Repeat from step 3 for each category the user has subscribed to.
8. Add the most recent posts that are alerts based on the latest created date.

Once the user has updated his subscription list from the user profile screen, the front page algorithm is triggered for a re-accumulation of the list.

Events Analysis:

The front-page will display events that are going to happen in the near future based on the category of events the user is interested in.

Algorithm:

1. Obtain user subscribed category list
2. Accumulate Event in each of the above categories based on the event start date
3. For each event, calculate
    a. votescore
    b. no.of comments made
    c. no.of people following the event
    d. no.of people attending this event
    e. no.of people not attending this event
4. Calculate the cumulative score of each event
        1. Find age - no. of hrs since the event was created
        2. Subtract the age from the score accumulated
5. Sort the list of events accumulated based on the cumulative score
6. Add the top 3 events(if available) to the Front page post list.
7. Repeat from step 3 for each category the user has subscribed to.

FrontPagePostService.java

```
59      public List<PostDAO> findTopPostsForCategory(String category,
60              String university) {
61          List<PostDAO> listOfPosts = postRepo
62                  .findTop10ByCategoryAndUniversityOrderByLastEditedOnDesc(
63                          category, university);
64          int[] scoreArray = new int[listOfPosts.size()];
65          int listIndex = 0;
66          for (PostDAO post : listOfPosts) {
67
68              int voteScore = post.getVoteScore();
69              int commentScore = findCommentScore(post.getId());
70              int followScore = post.getFollowCount();
71              int age = findAgeOfPost(post.getId());
72              int cumulativeScore = voteScore + commentScore + followScore - age;
73              scoreArray[listIndex++] = cumulativeScore;
74
75          }
76
77          return getTop4(listOfPosts, scoreArray);
78      }
79
80      public List<PostDAO> getTop4(List<PostDAO> listOfPosts, int[] scoreArray) {
81          List<PostDAO> topPosts = new ArrayList<PostDAO>();
82          if(listOfPosts.size()<4){
83              return listOfPosts;
84          }
85          for (int i = 0; i < 4; i++) {
86              int highScore = Integer.MIN_VALUE;
87              int highScoreIndex = i;
88              for (int j = i; j < scoreArray.length; j++) {
89                  if (scoreArray[j] > highScore) {
90                      highScore = scoreArray[j];
91                      highScoreIndex = j;
92                  }
93              }
94              int temp = scoreArray[i];
95              scoreArray[i] = highScore;
96              scoreArray[highScoreIndex] = temp;
97              topPosts.add(listOfPosts.get(highScoreIndex));
98              PostDAO tempPost = listOfPosts.get(i);
99              listOfPosts.set(i, listOfPosts.get(highScoreIndex));
100             listOfPosts.set(highScoreIndex, tempPost);
101         }
```

Figure 86: Front page algorithm

Recommendation System

The user will be sent recommendation emails to their registered email address about the new trending posts in the categories that they have subscribed to on a weekly basis. The Recommendations will be listed for each user based on a Scheduler that runs at specific intervals of time.

These recommendation emails sent to the user will have deep-links that open up specific app screen, when the links are clicked on using an android email app.

Notification Module

GCM Server

An application server that is implemented in our environment. This server sends data to a client app via the chosen GCM connection server, using the appropriate XMPP or HTTP protocol.

The app server needs to be:

- Able to communicate with the GCM client.
- Able to send properly formatted requests to the GCM connection server.
- Able to handle requests and resend them using exponential back-off.
- Able to securely store the API key and client registration tokens.

Whenever a post is updated or comments are created/edited the API sends a notification to the GCM client using a GCM Connection Server Protocol. JSON messages are sent as HTTP POST requests to the client and the message can be edited in the onMessageReceived( ) method in the MyGcmListenerService class. The topic needs to be subscribed by the user to be able to receive the notifications.

```
@Override
public void onMessageReceived(String from, Bundle data) {
    String message = data.getString("message");
    Log.d(TAG, "From: " + from);
    Log.d(TAG, "Message: " + message);

    if (from.startsWith("/topics/")) {
        // message received from some topic.
    } else {
        // normal downstream message.
    }
```

Figure 87: Code Snippet 20

Spam Filter Algorithm

The application utilizes a spam filter algorithm to block spam content from being created. The spam algorithm implementation is based on the Naive Bayes Algorithm. The implementation involves the following steps.

1. Create a list of commonly used acceptable words that a posted content can include.
2. Create a list of common spam words.
3. Create a list of common non-spam words.
4. WordStats object will keep track of a word's spam count, non-spam count and the probability that the word is spam.
5. Maintain a HashMap to store the Word and its associated WordStats.
6. Train the application with the above mentioned list of words and record the WordStats for each word that appears in the list of spam or non-spam words.
7. When the filter receives an input that is not present in any of the above mentioned lists, the filter calculates the spam probability and the non-spam probability. If the spam probability exceeds the non-spam probability, the content will be blocked from being created.

```
private static double calculateSpamProbabilityForString(String[] inputWords) {
    double spamVal = totalSpamProbability;
    // System.out.println(totalSpamProbability);
    if (inputWords.length > 0) {
        for (String wrd : inputWords) {
            String w = wrd.trim().toLowerCase();
            if (wordStatsMap.containsKey(w)) {
                spamVal = spamVal * wordStatsMap.get(w).getProbability();
            } else {
                spamVal = spamVal * (1 / totalSpamProbability);
            }
            // System.out.println(spamVal);
        }

        return spamVal;

    } else {
        return 0;
    }
}

private static double calculateNonSpamProbabilityForString(String[] inputWords) {
    double nonSpamVal = totalNonSpamProbability;
    // System.out.println(totalNonSpamProbability);
    if (inputWords.length > 0) {
        for (String wrd : inputWords) {
            String w = wrd.trim().toLowerCase();
            if (wordStatsMap.containsKey(w)) {
                nonSpamVal = nonSpamVal * wordStatsMap.get(w).getProbability();
            } else {
                nonSpamVal = nonSpamVal * (1 / totalNonSpamProbability);
            }
```

Figure 88: Code Snippet 21

Data-Tier Implementation

The Android UI uses REST API's in order to connect to the MongoDB database deployed on the EC2 instance.

MongoDB

Campus Tribune applications uses MongoDB for the backend. Different collections in the database document correspond to the different entities of the application. The data tier constitutes of 3 major divisions. They are: Users, Posts and Events.

When a user sign's up for the application the details are stores in the users collection. When the user logs in, the information entered is validated from the users collection and authorization is provided. User ids are automatically generated and stored.

All the posts created by users are saved in the posts collection. The user comments on the posts are stored in the postcomments collection. When a user creates/edits a post the database is updated accordingly. A new comment or an edited comment is saved into the database. The events module details are stored in the events and event comments collections and the CRUD operations are performed accordingly.

Amazon S3

Amazon S3 is being used for the storage of images. An S3 bucket is created and images are uploaded into the bucket. Posts and Events contain images that need to be uploaded or updated whenever a post/event is created or updated. \

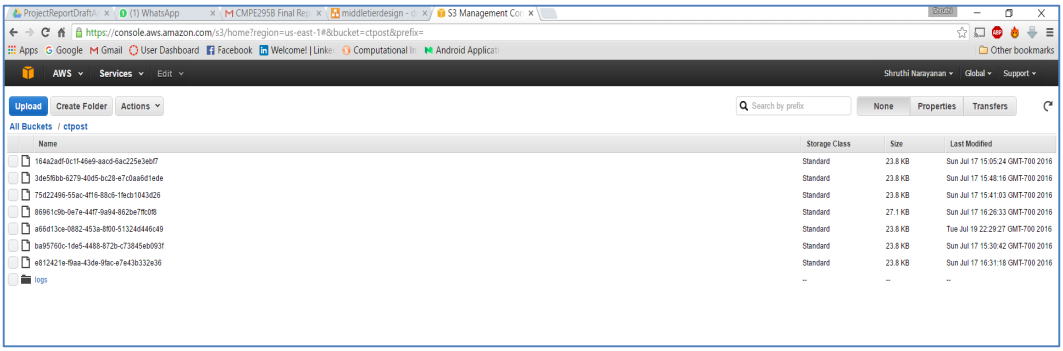


Figure 89: Amazon S3 Bucket Screenshot

An S3 url is returned back which is saved into the database. To load the image when an event or post is opened the S3 url saved in the database is called and the corresponding image stored in the bucket is displayed.

Chapter 6. Testing and Verification

Testing Strategy

This section describes our testing schedule, the different testing approaches that we have used and the scope definition for each of the approaches.

Testing Scope

The following table provides details of the testing scope for the application.

Testing Approach	Priority	Scope Description
Unit Testing	Medium	Validating the UI and backend code developed for individual modules and making sure that developed UI is rendered as expected and the backend implementation functions as expected.
Functional Testing	High	Validating each functionality against the requirements and making sure that all the requirements are captured as part of implementation.
Integration Testing	High	Validating the application for flow of data and logic between different modules by performing end to end testing.
Performance Testing	High	Validating the application performance against performance requirements.
Acceptance Testing	Medium	Validating the Application features against the functional and nonfunctional requirements

Figure 90: Testing Scope

Tools Used

Advanced REST Client

We used the Advanced REST client to test the REST API calls. The client provides fields to enter request body, authentication token and also the different HTTP methods to choose from. Based on the request sent, the appropriate response will be obtained.

JMeter

JMeter is used to run the load tests for REST API calls. For each API call, we can enter the request body, authentication token and the HTTP method type. Multiple requests can be loaded to get fired and the response can be plotted in a graph. This is used to calculate the response time and performance of the application.

Testing Approach

Unit Testing

Unit tests the working of small pieces of individual modules of the application. Each test will test a specific feature of the concerned module and make sure that the result is consistent with the assertion made.

The high-level steps included

- Checking whether UI is rendered correctly.
- Checking if the data from UI are passed to Activity/ Fragment class
- Checking the working of REST API components
- Validating the responses received

Functional Testing

Functional tests will test the complete working of a functional module. The testing makes sure that the client tier, middle tier and the data tier of the module functions properly.

The high-level steps included

- Checking whether UI is rendered correctly.
- Checking if the data from UI are passed to Activity/ Fragment class
- Checking the working of Async HTTP calls from client tier
- Checking whether the data passed from client tier to the middle tier is correct.
- Checking the working of database calls
- Validating the responses received from the data tier.
- Validating the response received in client tier from server tier
- Verifying the UI renders the results correctly

System and Integration Testing

Upon the completion of the module's implementation, the modules will be integrated and the integration testing will be used to test the interaction between them. As part of integration testing, we will test the application flow and make sure that data changes are captured correctly. We will perform system testing to test end to end flow of the system and will also make sure that the implementation meets the application requirements. This will employ both black box testing and white box testing techniques.

The high-level steps included

- Integrating UI components
- Integrating Server side components
- Integrating Database components
- Verifying the application flow and the data changes.

Performance and Stress Testing

Performance testing is done to check the system's adherence to industry-defined benchmarks. The testing will focus on characteristics like server response time, page render response time, touch response time,etc. Load testing will be done to verify the reliability and scalability of the system.

The high-level steps included

- Testing Server Response Time.
- Testing UI rendering speed.
- Testing Image loading speed.

We performed load tests on our application using JMeter. The following screenshots show the response time graphs for some requests with varying number of users.

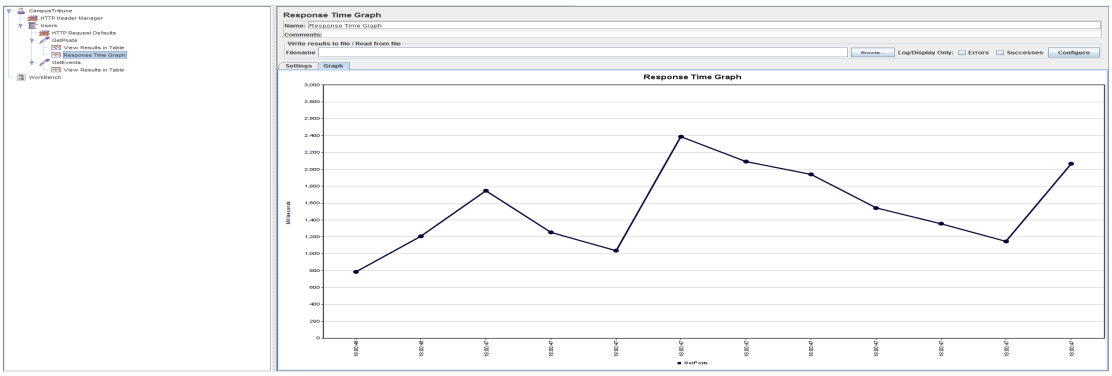


Figure 91 Response Time Graph-View Posts for 500 users

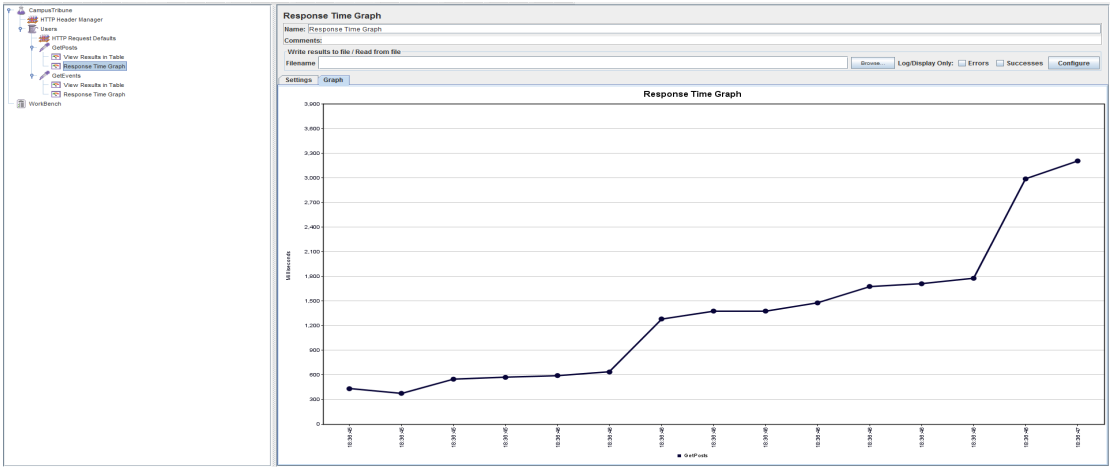


Figure 92 Response Time Graph-View Posts for 1000 users

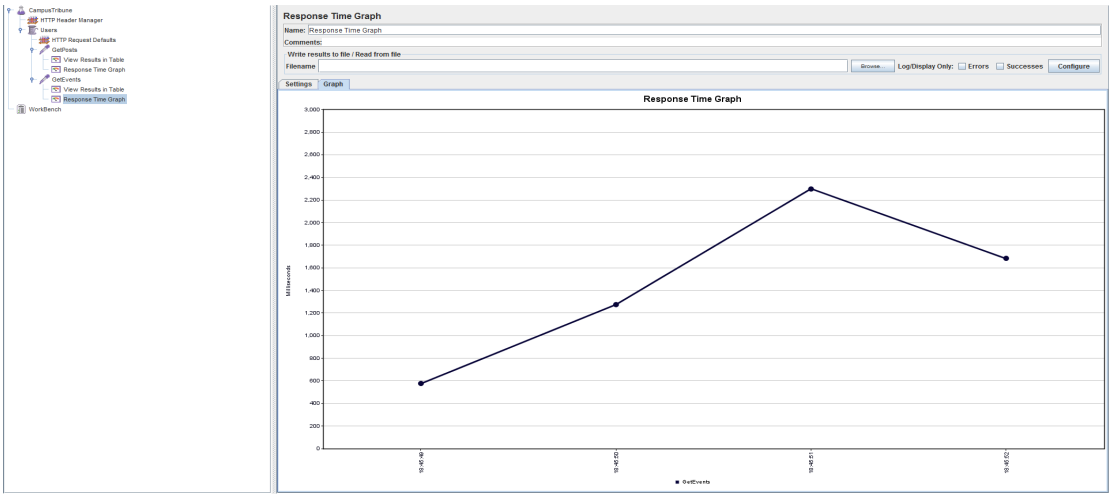


Figure 93 Response Time Graph-View Events for 500 users

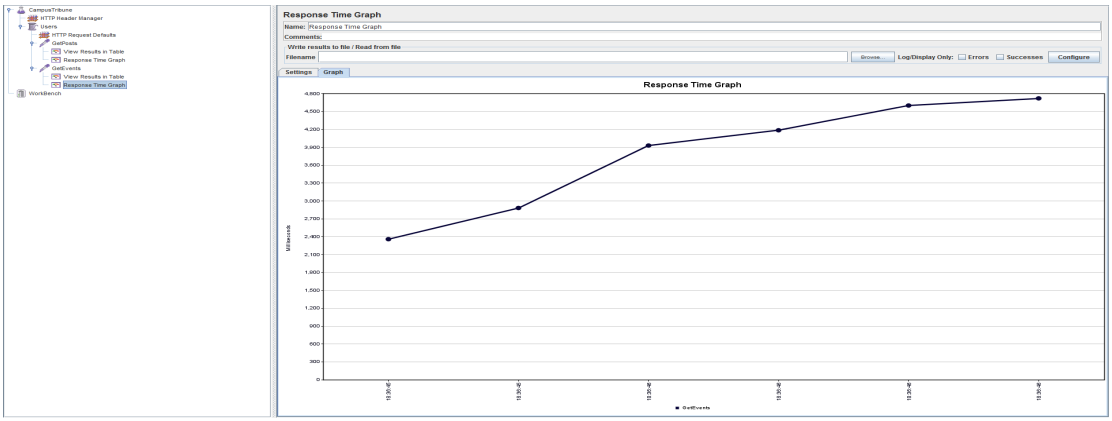


Figure 94 Response Time Graph-View Events for 1000 users

Acceptance Testing

The application will be tested to verify its adherence to the functional and nonfunctional requirement specifications. We intend to roll out a beta version of our application that will be tested by our fellow SJSU students to validate the application's compliance with the requirements.

Test Functionality	**TestCase No.**	**Test Description**	**Expected Result**	**Test Results** Round1	**Test Results** Round2	**Test Results** Round3
User	AT_01	A new user registers with the application using his/her .edu email id.	Upon registering, the user receives an activation email to verify the authenticity of the email id. The user id is sent to the user.	Fail	Pass	Pass
	AT_02	A user who is registered with the CampusTribune application tries to login with his .edu email id and password.	On successful login, the user is redirected to the application front page.	Pass	Pass	Pass
	AT_03	A user logged in to the application creates a post.	The post is created successfully.	Pass	Pass	Pass
	AT_04	A user logged in to the application views a post.	User can successfully view the post.	Pass	Pass	Pass

Post	AT_05	A user logged in to the application edits a previously created post.	User can successfully edit the post	Pass	Pass	Pass
	AT_06	A user logged in to the application tries to delete a previously created post.	User can successfully delete the post.	Pass	Pass	Pass
	AT_07	A user logged in to the application upvotes a post.	User can successfully upvote a post.	Pass	Pass	Pass
	AT_08	A user logged in to the application downvotes a post.	User can successfully downvote a post.	Pass	Pass	Pass
	AT_09	A user logged in to the application follows a post.	User can successfully follow a post.	Pass	Pass	Pass
	AT_10	A user logged in to the application unfollows a previously followed post.	User can successfully unfollow a post.	Pass	Pass	Pass
	AT_11	A user logged in to the application reports a post.	User can successfully report a post.	Fail	Pass	Pass

	AT_12	A user logged in to the application creates an alert.	User can successfully create an alert.	Pass	Pass	Pass
Comment	AT_13	A user logged in to the application comments on a post	The comment is created successfully.	Pass	Pass	Pass
	AT_14	A user logged in to the application views the comments created on a post.	User can successfully view the comments.	Pass	Pass	Pass
	AT_15	A user logged in to the application edits a previously created comment.	User can successfully edit the comment.	Pass	Pass	Pass
	AT_16	A user logged in to the application deletes a previously created comment.	User can successfully delete the comment.	Pass	Pass	Pass
	AT_17	A user logged in to the application reports a comment.	User can successfully report a comment.	Fail	Pass	Pass
	AT_18	A user logged in to the	User can successfully	Pass	Pass	Pass

		application creates an event.	create an event.			
Event	AT_19	A user logged in to the application views an event.	User can successfully view an event.	Pass	Pass	Pass
	AT_20	A user logged in to the application edits an event	User can successfully edit an event.	Pass	Pass	Pass
	AT_21	A user logged in to the application deletes an event	User can successfully delete an event	Pass	Pass	Pass
	AT_22	A user logged in to the application follows an event.	User can successfully follow an event.	Fail	Pass	Pass
	AT_23	A user logged in to the application unfollows an event.	User can successfully unfollow an event.	Fail	Pass	Pass
	AT_24	A user logged in to the application adds an event to his/her calendar.	User can successfully add an event to his/her calendar.	Pass	Pass	Pass
	AT_25	A user logged in to the application can report an event.	User can successfully report an event.	Pass	Pass	Pass

Table 18: Test Cases for Acceptance Testing

Bug Tracking and Verification

We are tracking our bugs in an excel sheet. The details added are the feature name, bug description and the steps to reproduce it if required. When a bug is fixed, the changes are pushed to git. All the members in the team fetch the updated code base and retest the functionality to verify the bug fix.

Functionality	Bug Type	Bug Outline	Asignee	Status
Front Page	Usability	IndicateEvents and Posts in different colored cards	Sandyarathi	Fixed
	Functional	Front Page must show newer posts(if any) when navigated to from other activiti	Sandyarathi	Fixed
	Usability	Show indication of alerts on front page	Sandyarathi	Fixed
	Functional	Front Page must indicate the categories for the posts/events	Sandyarathi	Fixed
	Enhancement	Add the virtual community feature for different universities	Sandyarathi	Fixed
	Enhancement	Add university logo in front page	Sandyarathi	Fixed
	Functional	Allow user to edit category preferences from user prfoile page	Sandyarathi	Fixed
	Functional	Click on event id displays the same event all the time.	Sandyarathi	Fixed
	Functional	Alert symbol displayed , even though the post is not an alert	Shruthi	Fixed
Post	Functional	Only the creator must be able to editdelete a post	Shruthi	Fixed
	Functional	Only the creator must be able to edit/delete a comment	Shruthi	Fixed
	Usability	Follow button is not very intuitive.	Shruthi	Fixed
	Usability	Use indication of alert on the view post page	Shruthi	Fixed
	Usability	Change the headline to appear in a different text color.	Shruthi	Fixed
	Enhancement	Add university field in Post document	Shruthi	Fixed
	Usability	Change the content to indicate that it accepts multiline text	Shruthi	Fixed
	Functional	Report of Posts and Comments does not work	Shruthi	Fixed
Events	Functional	Only the creator must be able to editdelete an event	Aditi	Fixed
	Functional	Only the creator must be able to edit/delete an comment	Aditi	Fixed
	Usability	Follow button is not very intuitive.	Aditi	Fixed
	Documentation	Remove the method type from API definition in the report	Aditi	Fixed
	Enhancement	Add university field in Event document	Aditi	Fixed
	Usability	Buttons in View Events Page are cluttered	Aditi	Fixed
	Functional	Implement Report Functionality for event and comments	Aditi	Fixed
	Functional	A user must not be able to upvote and downvote the same event	Aditi	Fixed
Notifications/Post by category/User Category Pa	Functional	Implement notifications feature	Divya	Fixed
	Fucntional	Implement posts by category UI	Divya	Fixed
	Functional	Implement user category page	Divya	Fixed
	Functional	Error when displaying posts with Alerts	Divya	Fixed
	Enhancement	Validate entered university and email id match	Sandhya	Fixed
User/Login	Enhancement	Add .edu validation	Sandhya	Fixed
	Enhancement	Add validation for unique email id	Sandhya	Fixed
	Usability	Display SignIn instead of SignUp as the launch activity	Sandhya	Fixed

Figure 95 Bug Tracking sheet

Chapter 7. Performance and Benchmarks

Performance Metrics

Mobile Client

Profiling tools used are Memory monitor and TraceView.

Application response time

The most data intensive functionalities in the app are the front page loading (list of items with text & image data) and image uploading for events & posts. The best practice for these activities is to use an asynchronous task to download and cache the images, so that the UI thread is not blocked. The goal of the application is to achieve Responses within 200-300 ms.

Screen rendering times

Use of excessive transitions and animations may increase the screen rendering time. The goal of the application is to achieve 60 frames per second, that gives the app 16.66 ms/frame to reload and update all the information on the screen

Memory and Battery usage

By using the memory monitor we traced the amount of memory space being used by the application. The memory usage by the application can be reduced by reducing the number of garbage collection events and reducing memory leaks. The battery drain can be reduced by lowering the open network connections & GPS access in the application.

API Metrics

Profiling tool used is Advanced Rest Client.

Server Response Time

The Backend Rest API's are being tested using the Advanced Rest Client Chrome extension. The Response returns an HTTP Status code and JSON Response body along with the call response time measure.

HTTP Call Count

The Server capacity can also be expressed in terms of the HTTP calls it can service per minute. The Spring Boot Actuator framework can be used to grab these metrics while running the app.

Benchmark Criteria

Following are the metrics through which the benchmark criteria has been evaluated.

Workload capacity of the app: 1000 users at a given time

Average Response Time: 1second

Data Capacity: Currently the app is using The Free Tier in MongoLab (500MB)

Criteria	Benchmark	Current Reading	Status
Front-page loading time	2s	4s	Needs improvement
Search Results	200 ms	200ms	Good
View Post with Image	300 ms	275ms	Good
Create Event with Image	300 ms	300ms	Good

Table 19: Benchmark Criteria

Chapter 8. Deployment, Operations, Maintenance

Deployment

Our application system consists of an Android front-end application deployed on a mobile device and a back-end server application deployed on a server system. For the back-end server, we are consuming AWS EC2 service to leverage its elastic computing capability. The NoSQL database is installed on the same EC2 node to reduce the network latency. The screenshot below shows the configured node.

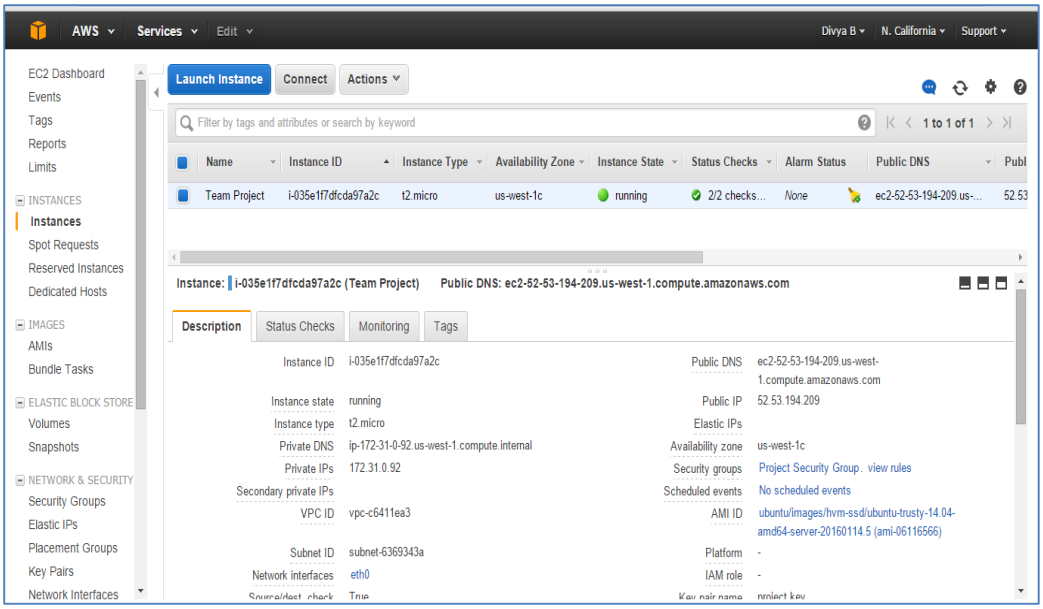


Figure 96: Amazon EC2 Configured Node

Steps for deployment of the back-end server application are as following:

1. Create and configure an AWS EC2 instance
2. Install MongoDb server on the EC2 node and create the application database
3. Set the connection parameters for the EC2 node and MongoDb server in application.properties of the back-end server application
4. Build the application and verify the no. of unit and integration test cases passing
5. If all the test cases pass, package the application into a monolithic jar and deploy it onto the configured EC2 node

Steps for deployment of Android front-end application are as following:

1. Build the application and verify the no. of UI test cases passing using an emulator
2. If all the test cases pass, package the application into a .apk file
3. Deploy the application onto a real Android device

The back-end application is built and packaged using Gradle plugin of Spring Tool Suite IDE and the front-end application is built and packaged using Android Studio.

Operations and Maintenance

The front-end application utilizes Google Maps APIs and Google Play Services for which a Google API key is acquired. The API key uses the application's SHA-1 fingerprint as the short form of its digital certificate. Currently, the application's debug certificate is used to generate the Android API key. When the application will be ready to be released to Google App Store, a new API key will have to be generated using the release certificate of the application. This key is specified in the Android application's manifest file as shown below:

```
<application>
        <meta-data android:name="com.google.android.geo.API_KEY"
                android:value="API KEY " />
</application>
```

The Android API key also has a validity period that needs to be renewed periodically. The Google APIs have a per day usage limit. The load of the application has to be continuously monitored after release to Google App Store and uplift in the usage limit of Google APIs has to be requested, as needed. Failing to do so will cause application failures.

The source code of both the applications is maintained on its respective central GitHub repository. Any bugs identified in the applications will be fixed and the applications will be re-deployed accordingly to the deployment steps mentioned previously.

Chapter 9. Summary, Conclusions, and Recommendations

Summary

The goal of this project was to develop a highly performant, secure and scalable localized mobile news platform for universities and colleges driven by user-generated content. The mobile platform provides liberty to publish posts and events related to campus news and share among all the users instantly. We intended to address three major issues faced by a user-generated content based collaborative online platform: spam-control, front-page algorithm and reputation system.

We have used a content-based spam-filtering algorithm, which analyzes the content of every post and event to calculate the probability of being a spam, which is rejected by the system. The front-page algorithm of our platform is inspired from user-based and item-based collaborative filtering techniques and uses an optimized blend of both in order to recommend valuable content to the end user. To address the third major challenge of the application, we have used a user-driven reputation system to retain and recommend genuine content to its users.

Conclusions

The Campus Tribune app relies solely on user-generated content and thus brings in a varied set of challenges like security, spam control, user engagement, handling of diverse content types. We have employed spam filters, user authentication techniques, recommendation & user specific front-page algorithms to address these challenges. Emphasis has been laid on developing the UI to appease the mobile app consumer/ user by designing an intuitive layout and considering the performance efficiencies.

The extensive literature study along with hands on proof of concepts helped us learn about architecture design choices and their implications. Designing the user interface of an android application which efficiently communicates with a backend application server was a great learning experience. We have also learnt the importance of project planning, team collaboration and time management in making a development project successful. Estimating the scope of the project for a given sprint proved to be challenge which we overcame by taking guidance from peers and our Project Advisor, Prof. Zhang.

Recommendations for Further Research

The application that we have implemented provides a platform for the students and staff of any university to share news and information. As a next step in making sure that good quality content is posted, we would like to use image detection and filtering techniques to implement spam filter for images as well. Our application currently runs only on Android OS platform. We also plan to extend this application to iOS platform.

We also plan to include social community features amongst users like following certain users, upgrading set of users to act as moderators. We plan to dole out recommendations to users based on user-user similarity scores.

Glossary

Front-Page Algorithm

Algorithm used for personalization of the landing page of an application according to the interests of the end-user and the past actions.

Reputation System

Reputation system computes the scores for a set of objects within a domain, on the basis of the explicit ratings that other entities provided.

Search Engine

Search engine is used to search for an article using the relevant keywords. The results are ranked according to their relevance.

Machine Learning Models

Machine learning models are the analytical models that are built by learning from data. It provides insight into data, without being explicitly programmed

Recommendation Algorithm

A type of machine learning algorithm used to recommend items to the users of a system based on item's or user's similarity or content of the items.

Micro-Service

Micro-service is a lightweight service classification framework for REST architectural style. It simplifies a service description and it's use in an application use case.

Monolithic

Monolithic is an architecture paradigm used for development and packing of an application as a single jar or war file.

Sharding

Sharding is the process of storing data records on multiple machines. It supports horizontal scaling as the amount of the data increases to support increased demand of the read and write operations.

Amazon Web Service Elastic Compute Cloud (AWS EC2)

Amazon elastic compute cloud is a web service that provides scalable compute capacity.

Auto-Scaling

Auto-scaling is the ability to scale up or down according to the defined conditions. It is a feature supported by AWS EC2 nodes for supporting increased application availability.

REST

REST is a lightweight alternative to mechanisms like RPC and SOAP and is a stateless, client-server communication protocol that uses HTTP.

MVC

Model-View-Control is a software architectural pattern used for implementing user interfaces of a software application. It divides the application into three interconnected parts so as to separate the way the data is presented to the end-user and the way it is stored internally.

Caching

A computing optimization concept used to store data from earlier requests to respond to similar requests in future without having to access the database.

Natural Language Processing

Natural language processing uses efficient algorithms to process texts and make their information available to computational applications.

Amazon Simple Notification Service (SNS)

Amazon SNS is a web service that enables applications and end-users to instantly send and receive push notifications.

Amazon S3

Amazon Simple Storage Service (Amazon S3) provides developers and IT teams with secure, durable, highly scalable cloud storage.

Spam Filter Algorithm

A spam filter algorithm detects unsolicited and unwanted content present in the user generated data.

Apache Mahout

A project of the Apache Software Foundation that provides in built machine learning libraries focused primarily in the areas of collaborative filtering, clustering and classification.

Redis

An open source in-memory data structure store used for caching purposes.

Memcached

Distributed memory caching system used to reduce the number of database access requests.

References

1. Bitragunta, Divya; Das, Sandyarathi; Narayanan, Shruthi; Rajawat, Aditi; "CampusTribuneWorkbook2Final" Submitted as Workbook 2 as an Assignment in CMPE 295A.
2. Almeida, T. A., & Yamakami, A. (2010). Content-based spam filtering. Paper presented at the *Neural Networks (IJCNN), the 2010 International Joint Conference*, 1-7. doi:10.1109/IJCNN.2010.5596569.
3. Fdez-Glez, J., Ruano-Ordás, D., Laza, R., Méndez, J. R., Pavón, R., & Fdez-Riverola, F. (2016). WSF2: A novel framework for filtering web spam. *Scientific Programming*, 1-18. doi:10.1155/2016/6091385.
4. Zhao, Z., Wang, C., & Lai, J. (2016). AUI&GIV: Recommendation with asymmetric user influence and global importance value. *Plos ONE*, 11(2), 1-21. doi:10.1371/journal.pone.0147944.
5. Chen, B., Zeng, A., & Chen, L. (2015). The effect of heterogeneous dynamics of online users on information filtering. *Physics Letters A*, 379(43), 2839-2844. doi:10.1016/j.physleta.2015.09.019.
6. Alfaro, L. D., Kulshreshtha, A., Pye, I., & Adler, B. (2011). Reputation systems for open collaboration. *Communications of the ACM*, 54(8), 81-87.
7. Fernández-Villamor, J.I. Microservices: Lightweight Service Descriptions for REST Architectural Style [PDF document]. Retrieved from Online Website: https://pdfs.semanticscholar.org/8630/40cc3ed8cb7dc002847d5b36d8d659bb7439.pdf.

Appendices

Appendix A. Description of Implementation Repository

We are maintaining two separate source code repositories – one for the native Android application and one for the back-end application. We are using Git repositories for version controlling in order to track every change made to the code and to maintain each version. We are using GitHub as the web-based graphical interface of Git.

GitHub provides several other useful features, which help us collaborate easily and merge individual changes without any conflicts. The commit history will track every change committed to the source code. We are leveraging the notification service provided by GitHub for being notified about every update made on the repository.

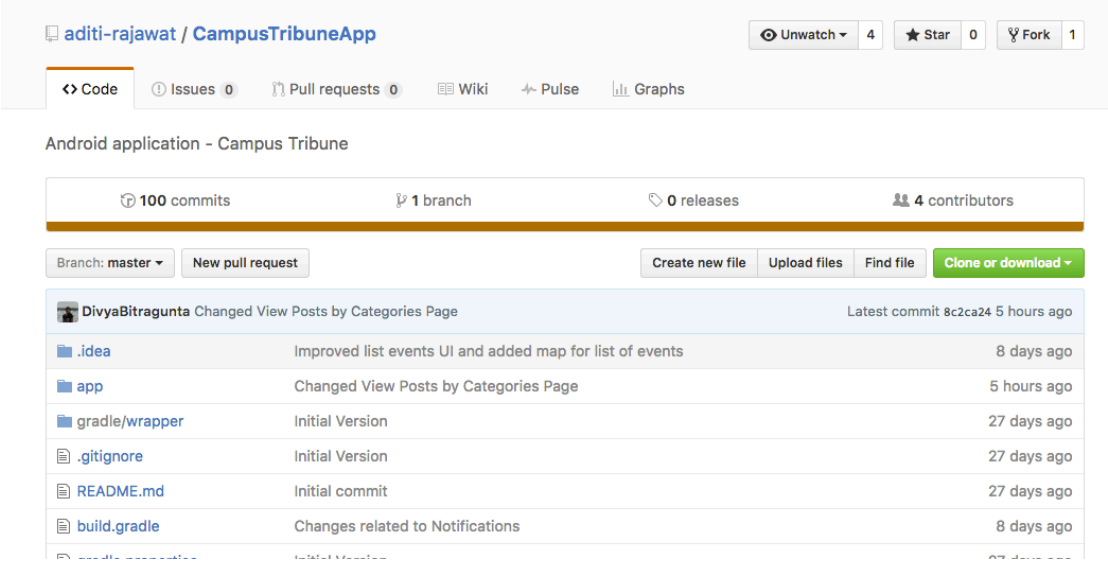


Figure 97: UI GitHub Repo

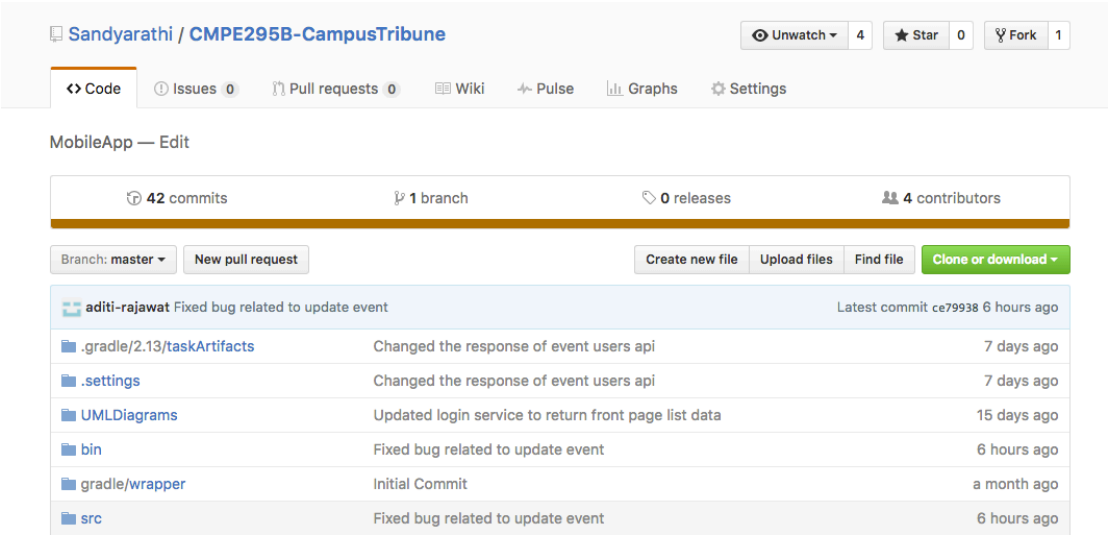


Figure 98: Backend GitHub Repo

Appendix B. Other References

Tutorial link for Google APIs: https://developers.google.com

Tutorial link for S3: https://aws.amazon.com/articles/3002109349624271

Background images used in the screens:
https://psbehrend.psu.edu/student-life/student-services/campus-safety

UI Code Github Repo:

https://github.com/aditi-rajawat/CampusTribuneApp

Backend Code GitHub Repo:

https://github.com/Sandyarathi/CMPE295B-CampusTribune