COP290: Complaint Management App

Prabhu Prasad Panda (2013ME10859)
Sauhard Gupta (2013ME10117)
Yash Kumar Bansal (2013ME10742)

9, March 2016

The **CMS** App is a mobile application of Complaint Management of IIT-Delhi. For User comfort the background is kept bright and the text fields are made large to ensure an easy viewing experience.Also the app guides the User through the various steps interactively and thus providing a user friendly interface.Lastly the application is easy to use.

1 SCOPE

- The Assignment aims to connect IIT Delhi community with Android Devices which includes:-
  1. faculty
  2. students
  3. staff/institute employees
- Using the app these end Users can submit their complaint/grievances to the concerned authorities.
- Thus, the application allows for *three* type of complaints:
  1. **Individual complaint**: A student wants to complain about the electricity problem in his hostel room to an electrician
  2. **Hostel level complaint**: Hostel residents want to complain about the bad quality of the mess food to their warden.
  3. **Institute level complaint** : Students across the institute want to protest against the LAN ban after 1 am.
- Lastly, we will extend this assignment to web clients in addition to the Android clients, if time permits, since API's support exists.

2 DATA STORAGE

1. **Users Table:-**Every member of the complaint management system is stored in this table. It comprises of the following fields.
   (a) ***User ID** (which is a string field that stores the Kerberos ID of the user)
   (b) ***Password** (String Password Field)
   (c) ***Name** (String)
   (d) **Email ID** (other than the Kerberos mail id - String)
   (e) **Mobile**(Numeric-10 digits)
   (f) ***Address** (String)
   (g) ***Locality** (Enumerated Data Type includes hostels, apartments, outside IIT etc.) – The outside IIT people can't raise complaints.
   (h) **Profile Pic** (Address)

   * - Compulsory Fields Keys: User ID will act as primary key.
2. **Administrators Table:-** It stores the various administrative posts in the complaint management system and their corresponding hierarchies. The fields are:
   (a) ***Admin ID** (which stores the ID associated with a particular post.)
   (b) ***Admin Password** (Password for the admin account).
   (c) ***User ID** (which stores the User ID of the user from Users Table who is currently managing the post. It is set as null if there is a vacancy. Validation rule – It must exist in the Users Table.
   (d) ***Parent ID** (stores the Admin ID of the parent of this admin.)
   (e) ***Name** (name of the post)
   (f) ***Description** (details of the post)
   (g) ***isLeaf** (Boolean states whether it is a leaf-level admin or not)

   Any Group Level Complaint can be addressed only to a leaf-level admin who can approve the complaint and sent it to the solvers for solving or can forward it to a higher level admin. Each admin can conduct polls to get the general opinion.

   The tree hierarchy must have the following root structure:

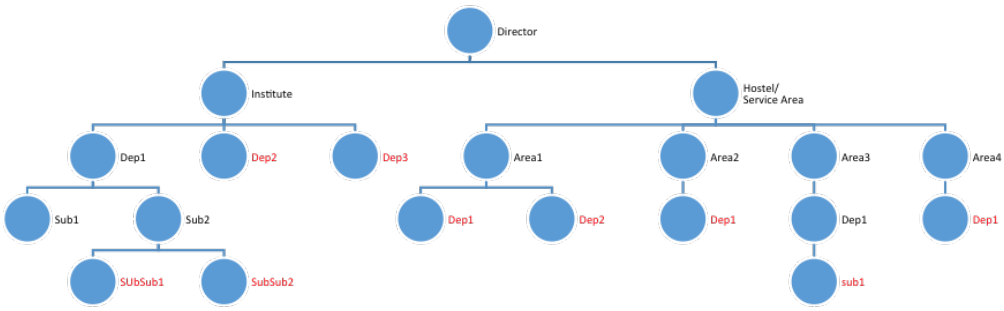

Figure 1: Hierarchy Tree

- **Solvers Table:-**This table stores the various departments who solve the problem. Each department is managed by a single user. (Currently we are not considering any hierarchies in the solving department but there is scope for extension) The fields are:

  1. ***Solver ID**: Stores the ID associated with a solving department
  2. ***Solver Password:** Password for the solver account.
  3. ***User ID:** Stores the ID of the user managing the post
  4. ***Department:** Name of the department
  5. ***Description**: Details of the department

- **Individual Complaints Table:**It stores all the individual complaints (or the approved group complaints of which an individual complaint will be lodged from the approver). The fields are:

  1. ***Complaint ID (key):** The ID of the complaint.
  2. ***User ID:** The User ID of the person who lodged the complaint.
  3. ***Solver ID:** The ID of the department to which the complaint is sent.
  4. ***Date:** The date on which the complaint is lodged or approved
  5. ***Status:** Status of the complaint
  6. ***Title:** title of the complaint

7. ***Description:** details of the complaint
8. **Links:** Stores the links attached
9. **Images:** Stores the addresses of the images attached
10. **Approved/Discarded Date:** If it is approved or discarded, the date is stored here

- **Group Complaints Table:**It stores the group complaints. Once approved an individual complaint will be sent from the approver to the solvers' department. The fields are:

1. ***Complaint ID (key):** The ID of the complaint.
2. ***User ID :** The ID of the user who lodged the complaint
3. ***Init Admin ID:** The ID of the leaf-level admin to which the complaint was initially addressed.
4. ***Cur Admin ID:** The ID of the Admin who is currently in charge of the complaint.
5. ***Date:** The date on which the complaint was lodged.
6. ***Transaction History:** It stores as text the history of transactions e.g., "forwarded by ¡admin1¿ to ¡admin 2¿ on ¡date¿¡time¿".
7. **Title:** Title of the complaint
8. **Description:** Details of the complaint
9. ***Type:** Type of complaint (for ex: institute level or hostel level for area 1 etc.)
10. ***Status:** Status of the complaint
11. **Links:** Stores the links attached
12. **Images:** Stores the addresses of the images attached
13. **Poll Results:** Poll results stored as string in a specified format
14. **Approved/Discarded Date:** If it is approved or discarded, the date is stored here

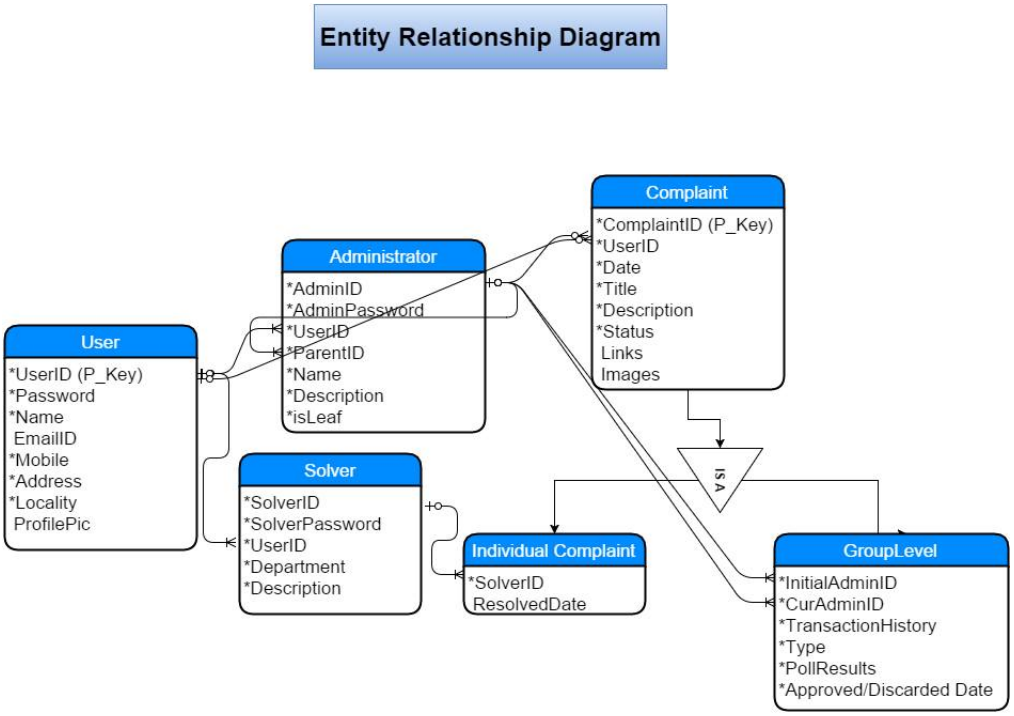

Figure 2: Entity Relationship Diagram

3 APIs and EVENT FLOW

1. **Login and Logout APIs:** We need to have APIs to enable login and logout. On logging in setCookie will be used to store the necessary cookie data in the device which includes encrypted information of the logged in user. This will be a session cookie and will be deleted as soon as the user logs out. APIs:
   -Logging in: domain/login/userid= userid pswd= password
   -Logging out: domain/logout.json
2. A normal user must be able to see individual level complaints raised by him, hostel –level complaints for his/her hostel or area and institute level complaints.
   APIs for the same:

   Individual level complaints: From the cookie the User ID can be accessed and a query sent to he individual complaints table that all complaints with User ID same as the user should be sent as JSON.
   API – domain/viewComplaints/Indilevel.json
   Method- GET

Hostel Level Complaints: Similarly the group level complaints with type as the hostel level complaint from the hostel/area of the current user should be sent as JSON.
API – domain/viewComplaints/Arealevel.json
Method- GET

Institute Level Complaints: The group level complaints with type as Insititute level complaints should be displayed.
API – domain/viewComplaints/Instilevel.json
Method- GET

3. A normal user should be able to lodge and edit (sometimes) individual, hostel-level and institute level complaints. So accordingly 3 APIs will be provided for each of these and POST method will -be used to send the JSON information.

- Individual Level:

API – **url/postComplaint/Indilevel/**
Method- **POST**
JSON format-

```
{
    "UserID":  <UserID>;
    "SolveID": <SolverID>;
    "Title": <Title>
    "Description":<Description>
    "Links":<Link1><Link2>...
    "Images":<Image1><Image2>...
}
```

Images can be sent using byteArray.

- Hostel Level:

API – **url/postComplaint/Arealevel/**
Method- **POST**
JSON format-

```
{
    "UserID":  <UserID>;
    "InitAdminID": <InitAdminID>;
    "Title": <Title>
    "Description":<Description>
    "Links":<Link1><Link2>...
    "Images":<Image1><Image2>...
}
```

Images can be sent using byteArray.

- Institute Level:

API – **url/postComplaint/Instilevel/**
Method- **POST**
JSON format-

```
{
        "UserID":  <UserID>;
        "InitAdminID": <InitAdminID>;
        "Title": <Title>
        "Description":<Description>
        "Links":<Link1><Link2>...
        "Images":<Image1><Image2>...
}
```

Images can be sent using byteArray.

4. Administrators need to view the complaints addressed to them, the history of their works (complaints which they forwarded to a higher admin, discarded complaints, approved complaints etc.) . The following describes the permissions for complaint viewing of administrators:
   - The director or the CMS head admin can view all complaints.
   - Everybody can view all Institute-level complaints.
   - Hostel/Service Area head admin can view all hostel/area-level complaints.
   - Admins associated only with a particular service area can view all the hostel-level complaints for that area only.
   - An admin can exercise his administrative privilege over only those complaints which are addressed or forwarded to them.
   - After forwarding a complaint to a higher admin, the admin losses his administrative privileges over that complaint.

   So according to the logged administrator (can be found from the cookie and will be sent along with the API), the APIs will be accessible. For example a hostel-level admin for a particular area will not have access to the hostel-level complaints of another area.

   The APIs will be of the form:

   - for institute level complaints retrieval: **/viewComplaints/Instilevel.json** (Method – **GET**)
   - for an area level complaints retrieval: **/viewComplaints/Arealevel/areacode.json** (Method – **GET**) (will return an error if the admin does not have access to that area).

5. **What can an Admin do?** He/she can approve, discard, start a poll and forward a complaint. So there has to be APIs for the same. All the fields except the polling will be discussed here. Polling will be discussed on the next point.

- **Approve a complaint:** Meaning the admin understands that the complaint is a valid one and he/she has the authority to act on it. So he approves it, i.e., the group complaint is sent as an individual complaint from the admin's account to the solver as chosen by the admin.
  API: **domain/Complaints/<ComplaintID>/Approve**
  Method: **POST**
  JSON format:
  {
  "SolverID":<SolverID>;
  }
- **Discard a complaint:** The admin does not feel the complaint needs attention and hence discards it.

  API: **domain/Complaints/<ComplaintID>/discard**
  Method: **GET**
- **Forward a complaint**
  The admin thinks the complaint needs to be viewed by some higher authority and hence he forwards the complaint to his parent admin and transfers the administrative privilege to him.
  API: **domain/Complaints/<ComplaintID>/forward**
  Method: **GET**

6. **Polling:** This includes starting a poll by an administrator, choosing a choice by a user, or extending deadline by an administrator.
   - **Starting a poll** :
     API: **domain/Complaints/ <ComplaintID>/Poll/Start**
     Method: **POST**
     JSON format: {
     "Deadline": <Deadline>
     "Choices":[{"ChoiceName":<ChoiceName>}]
     }
   - **Extending deadline:**
     API: **domain/Complaints/<ComplaintID>/Poll/ExtDeadline=<Deadline>**
     Method: **GET**
   - **Voting by User:**
     API: **domain/Complaints/<ComplaintID>/Poll/Choice=<Choice>**
     Method: **GET**

7. **Viewing a particular complaint:** Previously, we have defined APIs to view all complaints of a particular category. Now to see the details, the user can click on that particular complaint and modify the status if he/she is an administrator.
   API: **domain/Complaints/<ComplaintID>**
   Method: **GET**

8. **Changing the status of individual level complaints:** We have already considered these for group level complaints. Here we will deal with individual complaint version of the group level complaints or simple individual complaints. The status is resolved or unresolved.
   API: **domain/Complaints/<ComplaintID>/Status/Resolved=<Resolved>**
   Method: **GET**

9. **Viewing complaints by the solvers:** The solvers can view the complaints sent to them. We require APIs for these:
   - For viewing all the complaints (Unresolved Complaints will show first arranged chronologically followed by chronologically arranged resolved complaints):
     API: domain/viewComplaints
     Method: **GET**
   - For viewing a particular complaint
     API: domain/Complaints/<ComplaintID>
     Method: **GET**

4 WORK FLOW

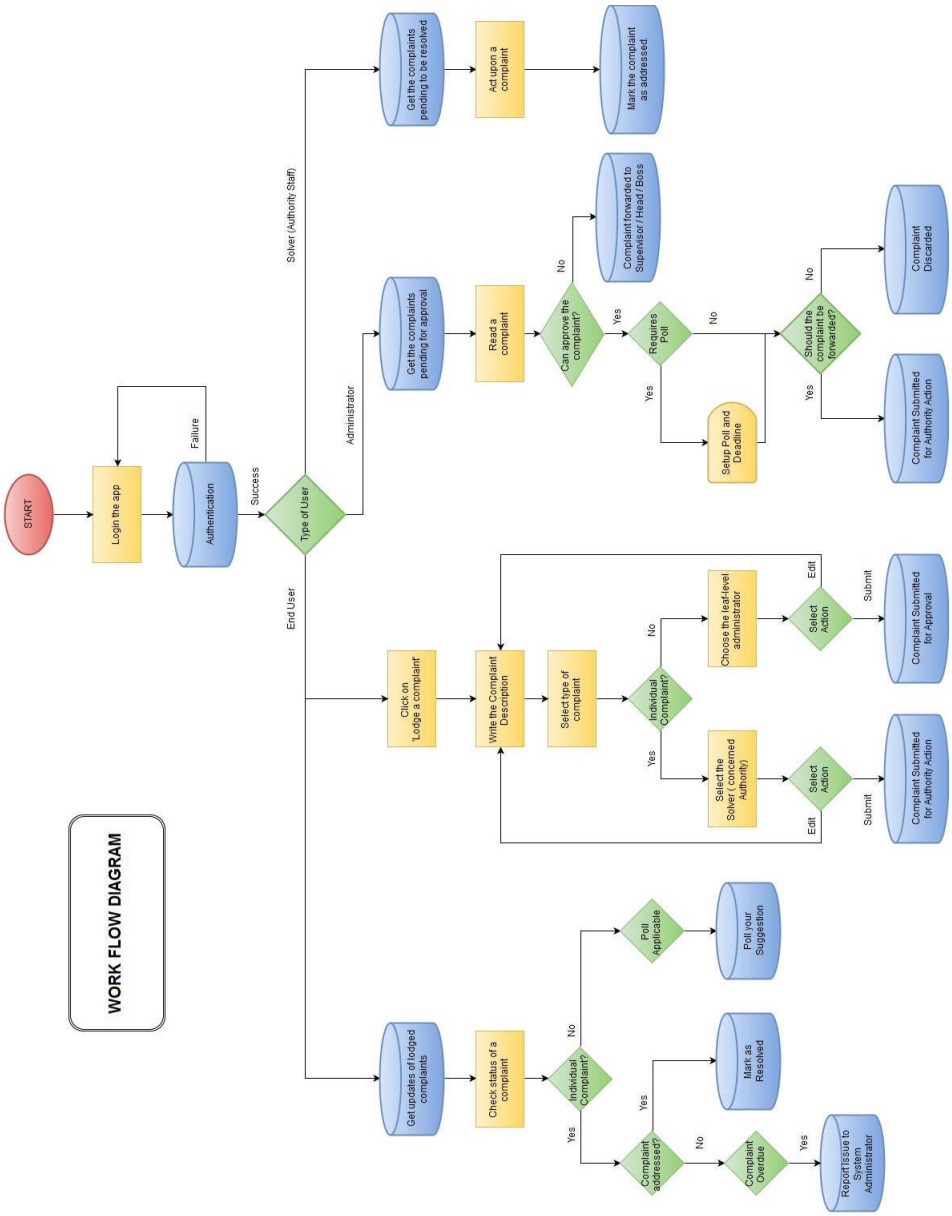

Figure 3: WORK FLOW

5 MODULARITY

(a) In this application, to store the information regarding the User and Complaint following classes need to be created :

i. USER

ii. ADMIN

iii. SOLVER

iv. COMPLAINT
    INDIVIDUAL COMPLAINT
    GROUP COMPLAINT

(b) Complaint Viewing would require:

i. Linked Lists of type Complaint

ii. Complaint Layout Class which gives the display for one complaint. For each complaint in the linked list, an object of type complaint layout must be defined dynamically so that the app is able to display all the complaints.

6 INTERFACE

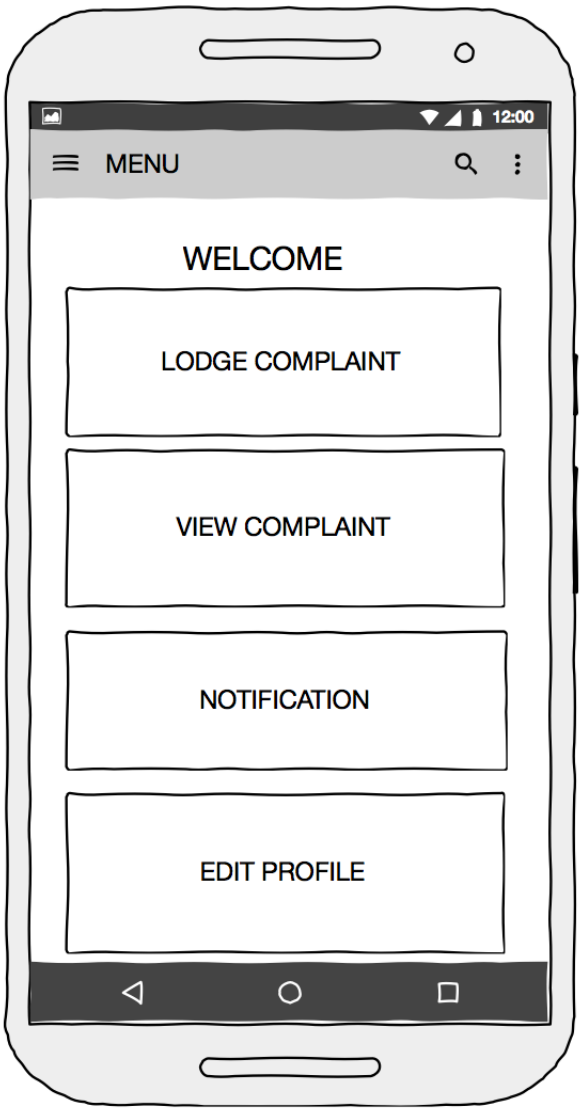

Figure 4: WELCOME PAGE

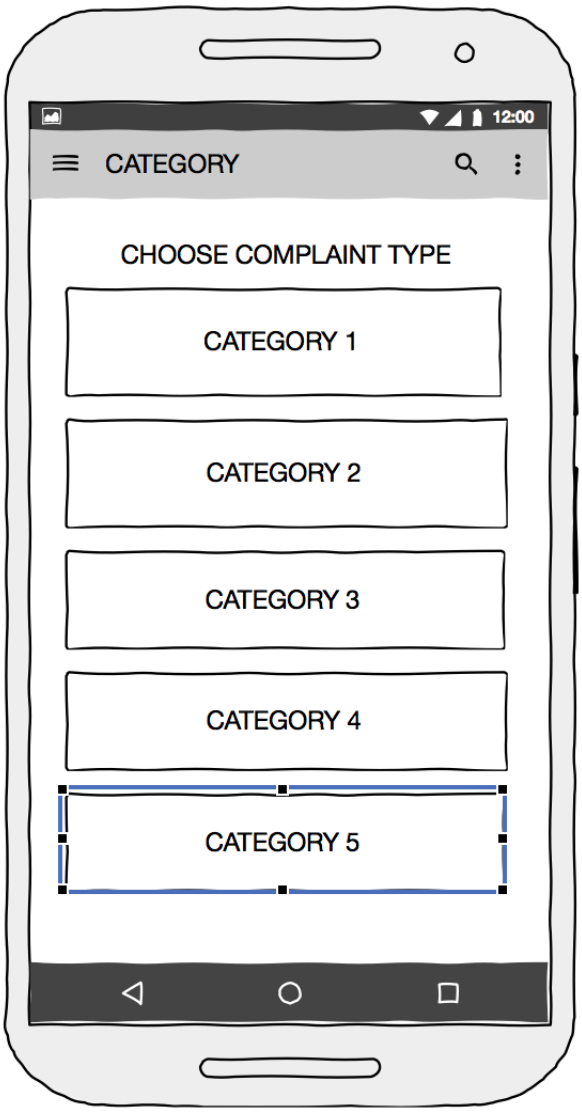

Figure 5: CATEGORY

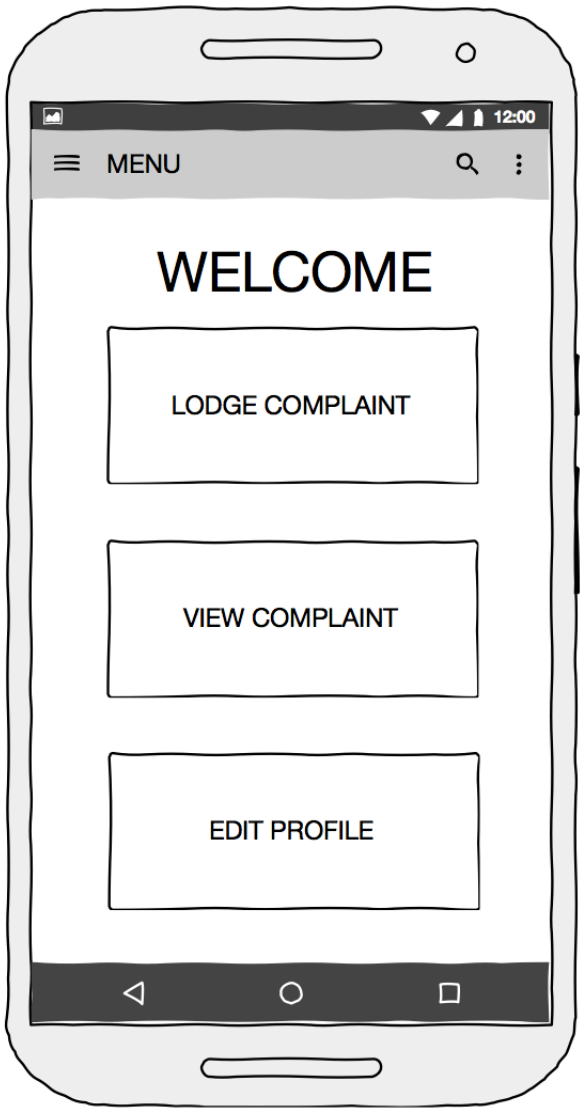

Figure 6: WELCOME PAGE

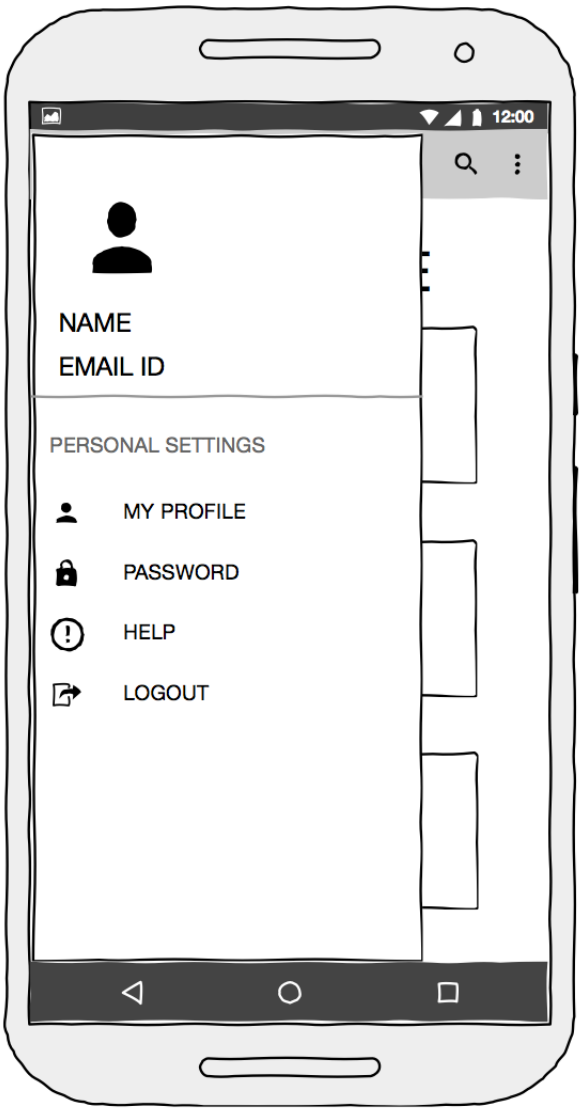

Figure 7: WELCOME NAVIGATION BAR

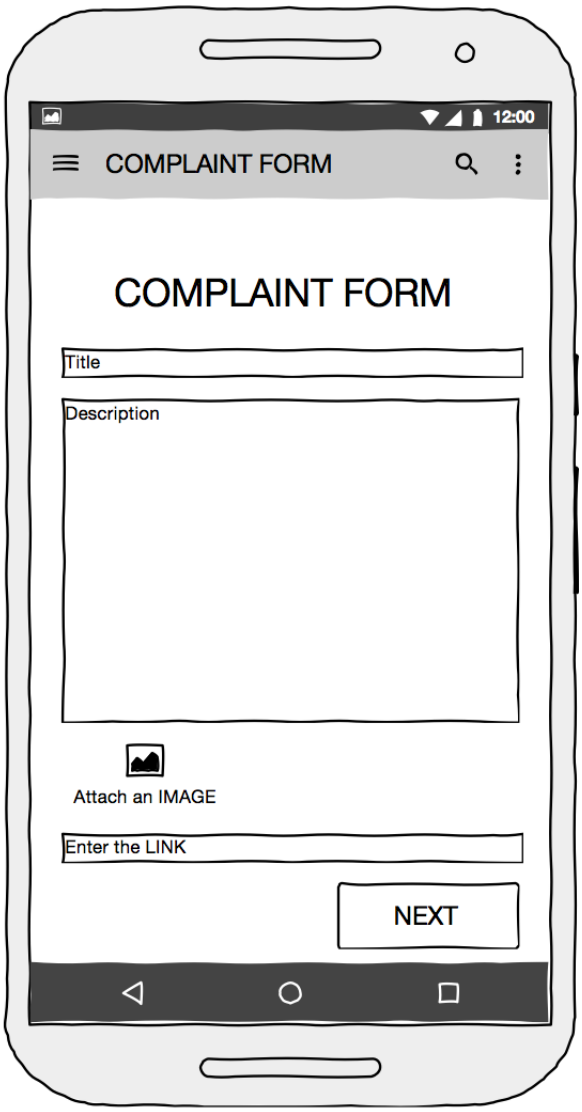

Figure 8: COMPLAINT FORM

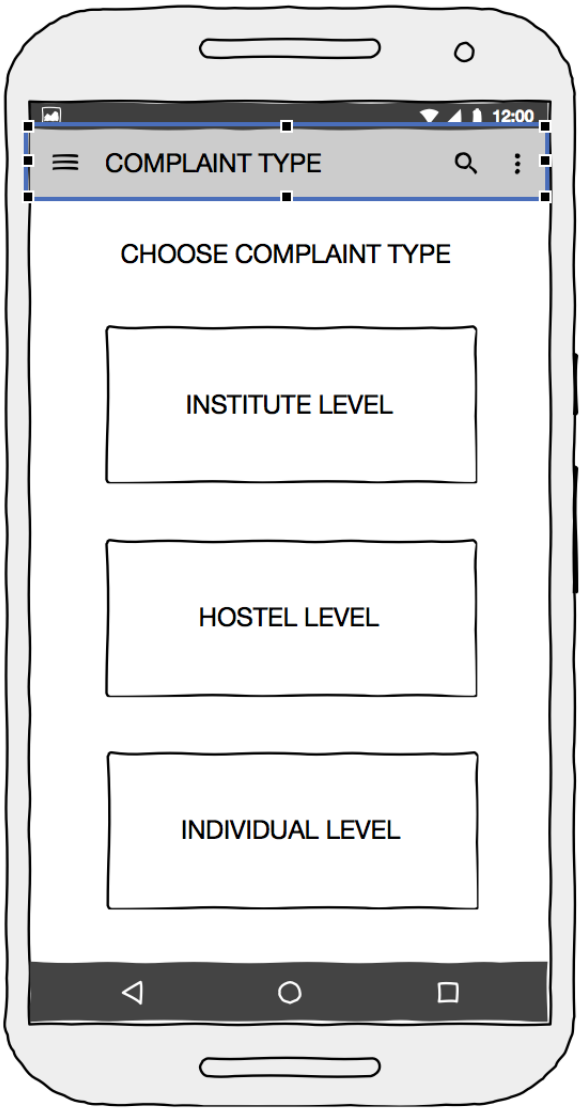

Figure 9: COMPLAINT TYPE

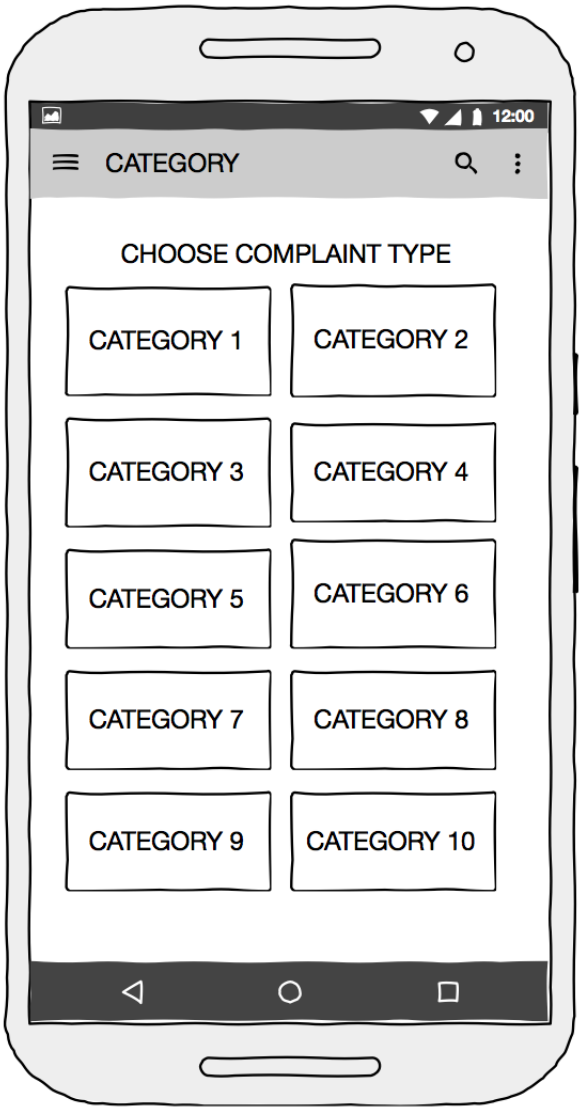

Figure 10: CATEGORY

All the above screens created in WIREFRAME to show a representation of the final diagram.

Report compiled by *Sauhard Gupta 2013ME10117.*

REFERENCES:-

http://developer.android.com
http://stackoverflow.com
https://github.com