

Cross Platform Application Development for Real-Time Transparency System

Summer Internship Report

*Project report submitted in partial fulfillment of
the requirements for the degree of*

Bachelor of Technology

IN

Computer Science and Engineering

BY

VENKATA ABHINAV DEVAGUPTAPU	N100926
NAGA SAMYUKTHA EDARA	N100461
KARTHIK GUMPU	N100208
RAJASEKHAR PANTANGI	N100847
SIVAKOTIREDDY BANDI	N100766

Under the Guidance of

Mr. Rahul Attuluri,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Director, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Vasanth Sai,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Senior Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Ranjith Nori,

B.Tech, Dept. Of CSE, BITS-HYDERABAD

*Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***



CyberEye™

CyberEye™ Research Labs & Security Solutions Pvt. Ltd.

**Flat No.305, Plot No.10,11, Reliance Kamal Gautami Enclave, Masid Banda, Kondapur,
Hyderabad - 500084, Telangana, India.**

May 2015 – July 2015

Cross Platform Application Development for Real-Time Transparency System

Summer Internship Report

*Project report submitted in partial fulfillment of
the requirements for the degree of*

Bachelor of Technology

IN

Computer Science and Engineering

BY

SHAIK IRFAN	N100572
SOWMYA BEZAWADA	N100522
MANGARAJU VELPULA	N100217
BHEEMASANKAR BABU	N092021

Under the Guidance of

Mr. Rahul Attuluri,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Director, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Vasanth Sai,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Senior Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Ranjith Nori,

B.Tech, Dept. Of CSE, BITS-HYDERABAD

*Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***



CyberEye™ Research Labs & Security Solutions Pvt. Ltd.

**Flat No.305, Plot No.10,11, Reliance Kamal Gautami Enclave, Masid Banda, Kondapur,
Hyderabad - 500084, Telangana, India.**

May 2015 – July 2015

Cross Platform Application Development for Real-Time Transparency System

Summer Internship Report

*Project report submitted in partial fulfillment of
the requirements for the degree of*

Bachelor of Technology

IN

Computer Science and Engineering

BY

NAGA MANIKANTA SATYANARAYANA KOTHA	N100622
SOBHA RANI POTNURU	N100381
SHAIK SALMA SULTHANA	N100175
TANUJA PALA	N100049
SUSANTH PANGI	N091130

Under the Guidance of

Mr. Rahul Attuluri,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Director, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Vasanth Sai,

B.Tech, Dept. Of CSE, IIIT-HYDERABAD

*Senior Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***

Mr. Ranjith Nori,

B.Tech, Dept. Of CSE, BITS-HYDERABAD

*Software Developer, **CyberEye™ Research Labs & Security Solutions Pvt. Ltd.***



CyberEye™ Research Labs & Security Solutions Pvt. Ltd.

**Flat No.305, Plot No.10,11, Reliance Kamal Gautami Enclave, Masid Banda, Kondapur,
Hyderabad - 500084, Telangana, India.**

May 2015 – July 2015

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people, who made it possible and whose constant guidance and encouragement crowns all efforts with success.

Firstly, we would humbly thank **Mrs. D V Nagarjana Devi**, Lecturer, CSE, for being the channel and guide for us to have availed such a rewarding opportunity.

We would like to attribute all the achievements and our endeavors to **Mr. S. Vijay Kumar**, the one person who trained us, motivated us and directed us in the right direction. He, is to be thanked for building up the prime qualities of *professionalism, right attitude, excellence as a habit and greatness as a virtue* in us.

We would like to express our deep sense of gratitude and whole-hearted thanks to **Mr. Rahul Attuluri, Director, Cybereye Research Labs and Security Solutions Pvt. Ltd.**; for giving us the privilege of working under his esteemed guidance and for the valuable suggestions, scholarly advice and comprehensive critical remarks during the course of the project and turning the plans into concrete outputs.

We extend the humble gratitude to **Mr. Vasanth Sai, Project In-Charge and Mr. Ranjith Nori, Project Supervisor**, for constantly and consistently guiding us, motivating us and showing complete confidence and trust in our work. Owing to their suggestions, training and encouragement, we consider ourselves to be able to complete the assigned tasks and achieving planned outputs properly and with utmost satisfaction.

We are also extremely thankful for having been trained under the expert guidance of **Mr. Avinash Dara** and **Ms. Sri Kavya**; **Mr. Tarun Krishna** and **Mr. Ram Ganesh**; **Mr. Neerad Kumar** and **Mr. Rahul Attuluri** in our post-project developments in the fields of *Internet of Things, Cyber Security, Android Application Development and Backend Development* respectively. Based on the principles of *learning on-the-go* and learning the core framework to excel in a field, we were trained in the very specifics of development and are extremely grateful.

Abstract

For a system to be completely transparent in terms of its functionality, especially for an NGO, there is a need of achieving the following initially: Financial Transparency, Hierarchy Of Management Transparency, Operations Transparency, Inventory Transparency, Policy Transparency, Decision Validation Transparency, Analytics Transparency, Donor/Sponsor details, Beneficiary Details, NGO Rating, Efficiency and improvement and Optimal quality information.

To approach and address the above specified requirements, the system needs to be an NGO cum user friendly framework; that is, one which is easily accessible and equally secure from both the user's and admin's end. An attempt has been made to develop the system which is totally transparent in terms of both monetary and functional aspects. All the specifiable data regarding the transactions of a social unit (in this case, an NGO) are displayed publicly with each transaction taking a maximum timeframe of one hour to be updated.

We built and designed the module to cater to multiple viewports ranging from desktops, tablets and mobile phones - all alike. Using React JS in collaboration with Flux Architecture, the project module hosts a dynamic, secure framework and is developed to be responsive at all times in all views.

Table Of Contents

Certificate

Acknowledgement

Abstract

Introduction

1.1 Nature of the Project Module

1.2 Background Survey

1.3 Approach Preferred

1.4 Technologies used

User Interface and Module Functionalities

2.1 User Interaction Perspective

2.1.1. Homepage

2.1.2. Campaigns

- I. About
- II. Sub Events
- III. Activities

2.1.3. Transparency

I. Monetary

- i. Discussions and Comments
- ii. Polls and Surveys
- iii. Tasks

2.1.4. User Donations Profile

2.1.5. FAQs

2.1.6. Notifications Bar

2.2 Project Flow

2.3 UML Diagrams

2.3.1 Activity Diagrams

2.3.2 Sequence Diagrams

2.3.3 Use Case Diagrams

2.4 User Interface

Challenges Encountered

Improvisations and Extensions

References

Introduction

For any NGO, or a social unit or an Organization, financial transparency has become the prime concern and requirement. Focusing on NGOs, there is a more concerning probability of miscalculation in financial records. Framing a challenge, the concept of 100% transparency is to be brought forward. And financial transparency has been focused as the first stage.

An attempt has been made to develop a system which is totally transparent in terms of both monetary and functional aspects. All the specifiable data regarding the transactions of a social unit (in this case, an NGO) are displayed publicly with each transaction taking a maximum timeframe of one hour to be updated.

The subject considered here is the NGO named VR1 Foundation, registered under Bombay Public Trust Act-1950 and simultaneously under the Societies Registration Act 1860.

1.1 Nature of the Project:

The module has been built to cater to multiple viewports ranging from desktops, tablets and mobile phones – all alike. Using React JS in collaboration with Flux Architecture, the project module hosts a dynamic, secure framework and is developed to be responsive at all times in all views.

1.2 Background Survey:

For a system to be completely transparent in terms of its functionality, especially for an NGO, there is a need of achieving the following initially: Financial Transparency, Hierarchy Of Management Transparency, Operations Transparency, Inventory Transparency, Policy Transparency, Decision Validation Transparency, Analytics Transparency, Donor/Sponsor details, Beneficiary Details, NGO Rating, Efficiency and improvement and Optimal quality information.

To approach and address the above specified requirements, the system needs to be an NGO cum user friendly framework; that is, one which is easily accessible and equally secure from both the user's and admin's end.

1.3 Approach Preferred:

The preferred Secure Development Life Cycle model is RAD (Rapid Application Development) owing to two prime factors, short development period and a need of rapid prototyping.

The RAD model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product. Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the

customer using iterative concept, reuse of the existing prototypes, continuous integration and rapid delivery.

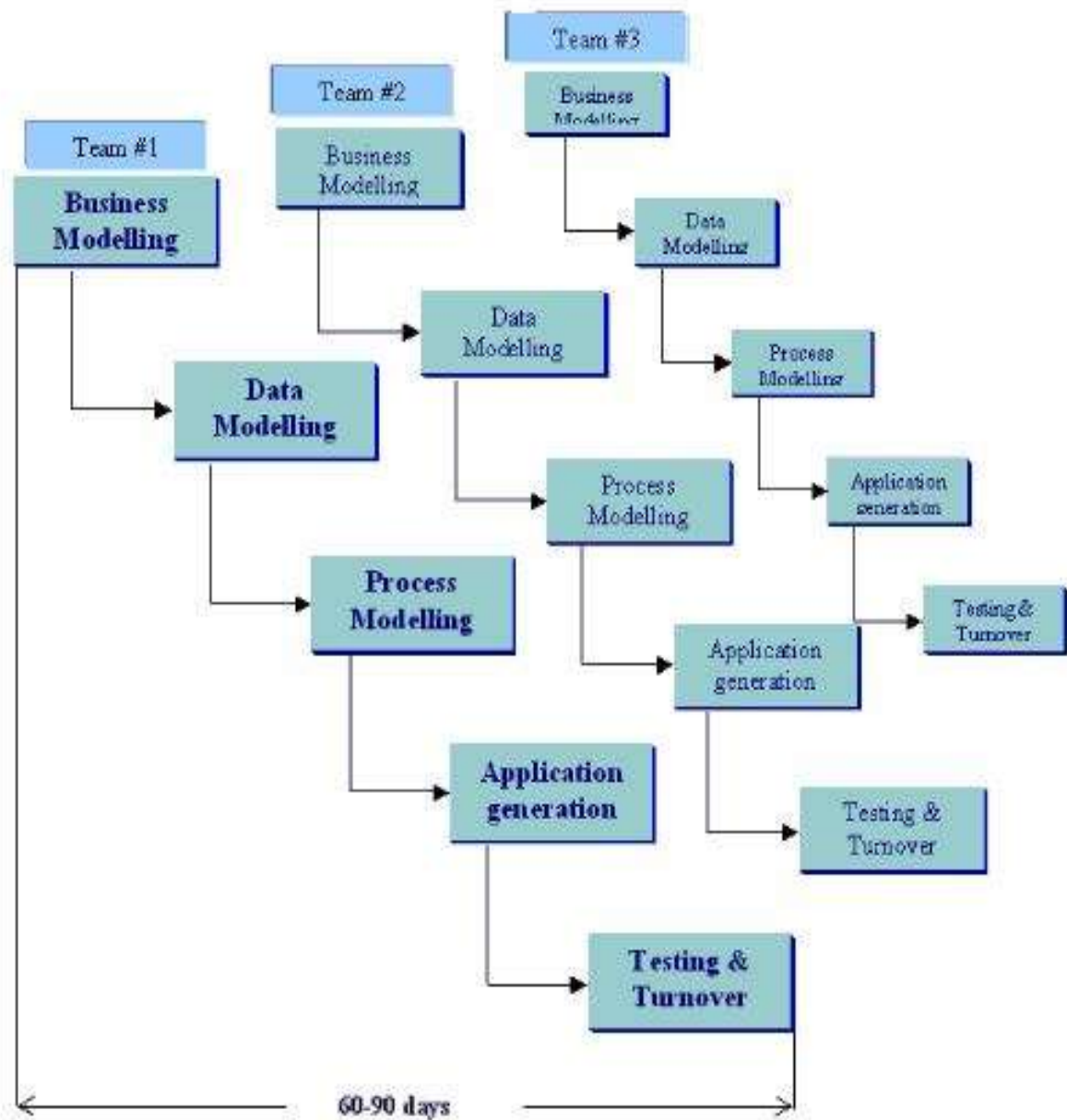


Fig. RAD Model

RAD Model Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

Following table lists out the pros and cons of RAD Model:

Pros	Cons
<ul style="list-style-type: none">• Changing requirements can be accommodated.• Progress can be measured.• Iteration time can be short with use of powerful RAD tools.• Productivity with fewer people in short time.• Reduced development time.• Increases reusability of components• Quick initial reviews occur• Encourages customer feedback• Integration from very beginning solves a lot of integration issues	<ul style="list-style-type: none">• Dependency on technically strong team members for identifying business requirements.• Only system that can be modularized can be built using RAD.• Requires highly skilled developers/designers. High dependency on modeling skills.• Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.• Management complexity is more.• Suitable for systems that are component based and scalable.• Requires user involvement throughout the life cycle.• Suitable for project requiring shorter development times.

1.4 Technologies Used:

The design and developmental approach preferred for the project includes the usage of React JS and Flux architecture. Although, there are multiple other contemporaries, React JS relatively precedes them in terms of processing speed and its resistance to XSS vulnerabilities.

Generally, since DOM manipulations are slow, React replicated the DOM virtually. Any application designed in React, connects to the virtual DOM (which is very fast), and then shifts the virtual DOM with the real DOM and applies all changes efficiently.

- **React JS:**

React is a JavaScript library for creating user interfaces by Facebook and Instagram. React was built to solve one problem: **building large applications with data that changes over time.**

To do this, React uses two main ideas.

Simple

Simply express how one's app should look at any given point in time, and React will automatically manage all UI updates when the underlying data changes.

Declarative

When the data changes, React conceptually hits the "refresh" button, and knows to only update the changed parts.

Build Composable Components

React approaches building user interfaces differently by breaking them into **components**. Since they're so encapsulated, components make code reuse, testing, and separation of concerns easy. This means React uses a real, full featured programming language to render views, which is an advantage over templates for a few reasons:

- **JavaScript is a flexible, powerful programming language** with the ability to build abstractions. This is incredibly important in large applications.
 - By unifying markup with its corresponding view logic, React can actually make views **easier to extend and maintain**.
 - By making an understanding of markup and content into JavaScript, there's **no manual string concatenation** and therefore less possibilities for XSS vulnerabilities.
- **Flux (architecture):**

Flux is the application architecture that Facebook uses for building client-side web applications. It complements React's composable view components by utilizing a unidirectional data flow. It's more of a pattern rather than a formal framework. Flux applications have three major parts: the dispatcher, the stores, and the views (React components).

MVC (model view controller) is good for small applications, but the complexity explodes when many models and their corresponding views are added to a system. Since, the data being processed alters dynamically, Flux is the best option to tackle the "one way" data flow with very specific events and listeners rather than using the pre-existing MVC. It is a new kind of architecture that complements React and the concept of Unidirectional Data Flow.

Or, in more simpler terms,

The *Store* contains all the application's data and the *Dispatcher* replaces the initial *Controller*, deciding how the *Store* is to be updated when an *Action* is triggered. The *View* is also updated when the *Store* changes, optionally generating an *Action* to be processed by the *Dispatcher*. This ensures a unidirectional flow of data between a system's components. A system with multiple *Stores* or *Views* can be seen as having only one *Store* and one *View* since the data is flowing only one way and the different *Stores* and *Views* do not directly affect each other. (ref. Fig.)

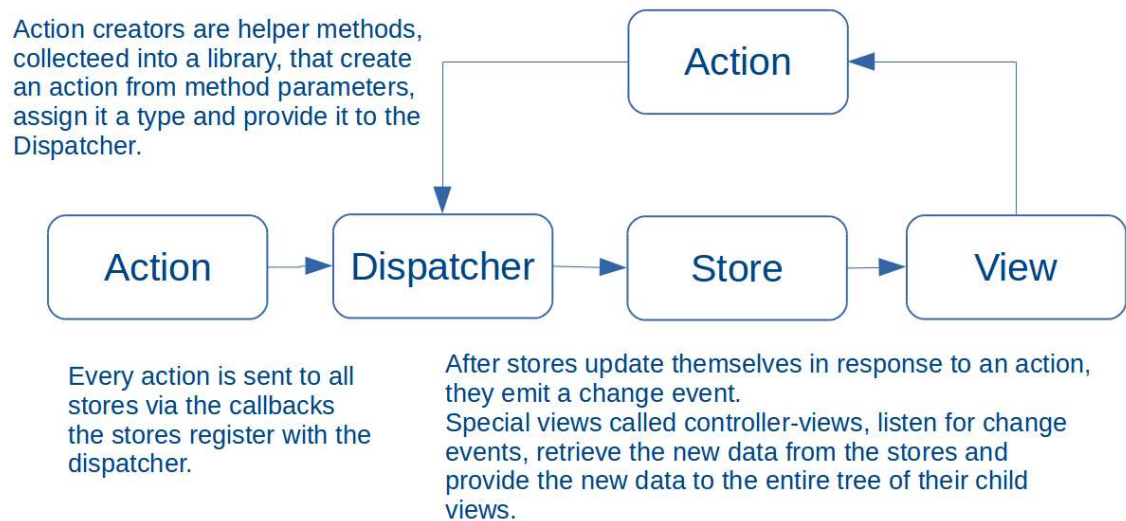


Fig. Flux Architecture.

- **HTML5 & CSS3:**

HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. It is an attempt to define a single markup language that can be written in either HTML or XHTML. It includes detailed processing models to encourage more interoperable implementations; it extends, improves and rationalises the markup available for documents, and introduces markup and application programming interfaces (APIs) for complex web applications.

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. CSS3 is divided into several separate documents called "modules". Each module adds new capabilities or extends features defined in CSS 2, preserving backward compatibility.

- **Twitter Bootstrap:**

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic & responsive websites and web applications.

Bootstrap provides a set of stylesheets that provide basic style definitions for all key HTML components. These provide a uniform, modern appearance for formatting text, tables and form elements.

- **Jquery Ajax :**

Ajax stands for Asynchronous JavaScript and XML. AJAX is a technique for creating fast and dynamic web pages. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This makes it possible to update components of a web page, without reloading the whole page. Classic web pages, (which do not use AJAX) must reload the entire page if the content should change. AJAX applications are browser and platform-independent.

Examples of applications using AJAX: Google Maps, Gmail, YouTube and Facebook.

AJAX is based on internet standards, and uses a combination of:

- XML Http Request object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

2. User Interface and Module Functionalities

2.1 User Interaction Perspective

2.1.1 Homepage and Login

Login Box Component:

Login Box component renders user authentication form. On clicking the submit button, it calls ***Logincheck*** function from the Utils component.

Logincheck () sends user input to the corresponding URL by making an API call. If the data is sent successfully, OTP is generated and access token is stored in a cookie. Further

functionality is totally based on this access token. On fail condition, an error message is displayed.

2.1.2 Campaigns

NavigationBar is component to create a navigation bar. It has two props. Items and Hashes. Items takes list of strings, those strings are visible in navigation bar as links. While Hashes takes list of strings too forming the corresponding hyperlinks for the each string in the items list. **PageNavigation** is a component which renders the Navigation **bar**. Navigation bar contains ['Home', 'Transparency', 'Donations', 'Profile', 'About', 'FAQ'] as prop. Items.

NGOHome (Home) is the root component. Here the server calls will be made. It renders the subcomponent **NGOCardDisplay** component to define the view of the campaign card. The funding percentage is calculated using **campaignFundingPercentage** variable. Properties for this component include: title, pledgedQuantity, fulfilledQuantity, fundedQuantity, goal, timeleft. On clicking the campaign card, the user is redirected to the **Subcampaigns** component.



Fig.1 NGO campaign card

Functions used:

`handleScroll ()`: It handles the scrolling of the campaigns based on window height. When scrolled to the end of the page, it automatically loads the next campaigns.

Sub Navigation Bar

SubNavigation component renders another component **Navigation pill**. Navigation pill is similar to the **NavigationBar**, it takes items and hashes as props.

focusIndex is index number starts from 0 and up to the no. of items displayed in navigation bar.

SubNavigation component takes **focusIndex** as prop. So that corresponding visible link in Navigation Bar will be active. Starting from the Views element- connected to the sub events, display is split into About, Sub campaigns and Activity (User interactions) using the **CampaignNavigation.js** component.

CampaignNavigation component renders navigation pills titled About, Subevents and Activity where the data of respective pill is rendered by **About.js**, **SubCampaigns.js** and **Activity.js** respectively.

I. **About.js** component renders all the details about the NGO.

II. Sub Campaigns.js

A classic example of service and response submission system. An authorized (logged in) user can see all the subcampaigns corresponding to a campaign where the information like goals to be achieved, funded quantity, pledged quantity and time left are provided. And one can see the details about a particular campaign when required. A virtually trackable Sub campaigns component, its functionality is 100% transparent to the user about the subcampaigns.

This component renders sub campaign cards related to a particular campaign where campaigner details of funded and pledged amount is displayed. To the right of this, the goal, time left to reach it, total funded quantity for that campaign is summed up and displayed

In views, the data is requested from the server. An action is called after the server call. From actions through dispatcher stores is called. In stores requested data is stored in the respective component. In views, donation list takes the data from `getsubcampaignslist()` in stores. `Getsubcampaignslist()` retrieves the previously requested list of subcampaigns stored in stores and sends the data back to the Views element where it's displayed in the subcampaigns component.[ref. Section 2.2 – Project Flow]

III. Activities.js

Activities is a field in which the user activities, contributions, comments, initiations, suggestions and any mode of change in the website that reflects the User ID on the website is displayed. Though it seems similar to comments and response submission system, which provides an interface, whose functionality is same as submissions in the Discussions module except in the case of new activity initiation which happens accordingly as per user activity. An authorized (logged in) user is privileged enough to view the activities of himself and others also, provided, these logs are comment-able so that the activities can be encouraged with the suggestions of doing it better and with perfection. In simpler terms, an attempt has been made to design a comment able Activity log; its functionality with reference to the Facebook Activities

For any user, NGO activities are obvious if he/she does any activity in the website, they will be notified and shown transparent by the NGO, if one is interested in any of the discussions, tasks, donations, funds, queries, surveys, suggestions can submit their response within those particular fields provided there itself and submit it, which will be further redirected to the display division where the activities of the user are displayed along with the time stamps of his/her activity.

The user then entitles to a multi-choice decision.

- Activity of the user is added accordingly and automatically by entering any text/likes/comments etc. in any part of the whole website.
- The user can like own/others activity.
- The user can check-out various previous activities and the comments if specified on those activities and also the submission status provided.

2.1.3 Transparency

I. Monetary

Here, **CardFrame** is the root component. It renders list of NGO Cards with respect the current screen width. No. of Cards per single row in large screen, medium screen and small screens are different. If client changes the browser window size, then from previous screen size to current screen size the cards are to be adjusted in each row. CardFrame component has state variable *screenSize*. It represents the current browser window screen width. Whenever this screenSize state variable changes automatically DOM is changed by **handleResize** function.

Functions used:

getCurrentScreenSize(): No arguments required .Return type: **string**

This function returns current window screen size as 'xs' or 'sm' or 'md' or 'lg' according to bootstrap default class names.

getCardsPerRow(): No arguments required. Return type: **int**

This function is used to calculate the number of cards in each row for various screen sizes. For extra small screens '1', for the small screens '2' and for both medium and large screens it returns '3'. According to the need the developer can simply change this number in this function.

findCurrentRow(): Takes two arguments screenWidth and cardIndex of type **String**. Return type: **String**

SetFlag():

This function is used to call corresponding server calls according to tasks, surveys and polls etc. This function shows/hides the arrow that focuses to the clicked tab and loads the corresponding component. This function is used to calculate the position of a particular card after screen resizing by considering the previous screen size and current screen size. cardIndex is the particular NGOCard unique id, and screenWidth is current screen size.

Child Components:

CardWithDiv component renders the cards in each row. **SingleCard** is the child component and it is responsible for individual card display.

i. Discussions and Comments

An attempt has been made to design the discussions and comment box sub module with reference to the Facebook comment box. In comparison, however, the module will not have all the features but will be using the same mode of data retrieval and processing functionality as supported in React JS. Also, when compared to functionality and speed of retrieval in other contemporary sources (in this case, Angular JS), the module acquires an advantage

A classic example of Discussions, comments and response submission system, the module works in connection with the monetary component wherein the functionality is similar to submissions in the latter. An authorized (logged in) user is privileged enough to upvote or downvote a comment or a discussion, delete his/her own entry and report any other entry which the user considers inappropriate. In short, the sub Module, gives an opportunity to start a discussion or comment about an existing one about NGO's Activities, which the NGO has posted in the site.

Functionality:

In the transparency project sub module, for every NGO activity post, there is a discussion form to suggest or discuss about a post, which is provided with an option 'Discussions' associated with every post.

In the discussion form, user can post suggestions/Questions about the post and can select a discussion tag. By clicking on the submit button new discussion is displayed in the discussions list. Every discussion has user's name who has entered the discussion, the profile image, discussion topic, no. of comments, discussion ID and an option to 'initiate a comment' to comment on the discussion, a chance to upvote or downvote too. Votes count is displayed there.

For any pre-existing discussion, if one is interested in adding a comment or a view, one simply has to click on the comment button, and is redirected to a comments division wherein, there are two entities; the comment form and a list of previously rendered comments. The user is then entitled to a multi-choice decision.

- One can add a comment/discussion of one's own by typing in the form and pressing "ENTER"
- Or one can upvote/downvote an existing entry if it favours by clicking on the thumbs-up or thumbs-down sign for upvoting or downvoting respectively.
- It is to be noted that for a single user, only a *single* upvote or downvote is allowed
- Alternatively, one can always click the flag icon to report an entry as inappropriate.
- If one decided to remove one's own entry, this can be done by clicking on the "x" icon.

ii. Polls and Surveys

The Flow of the module goes in a loop [ref. Section 2.2 Project Flow]. Firstly, to get the questions form the server an Ajax call is made from web API Utils to get the poll questions form server. And this information is sent to Actions element. Actions will interact with dispatcher element, which in turn, communicates with the Stores element and the data retrieved get stored. Using the emitChange() function call, the Views element is notified about the information. Views will display the poll questions by retrieving them from the Stores.

After the user interacts with the questions, the response generated is sent to the server for an update. This is done by using another Ajax call written in a function in Web Utils element. After server update, the corresponding response from the server which contains the values of polls for each answer is to be retrieved using another Ajax call. The percentage of each answer is calculated and using the values, vertical bars (poll bars) are created in a

division dynamically and get added to the existing division. New divisions are created dynamically by using JavaScript.

If the user has already polled a question, he/she will not have chance to answer the question again. To achieve this, each user's last response is checked. If the user has already given the survey, only the statistical data is displayed. Else, the user will be displayed with the poll form.

In order to render the surveys, initially, the survey questions forum from the server is retrieved using Ajax calls in web API Utils. Calling that function gets the survey questions from the server. This information is sent to Actions component. Actions component interacts with dispatcher and dispatcher will communicate with stores and discussions list will be stored in stores. Using emitChange() to views, the survey questions list is displayed by getting them from the stores.

Following that, all survey questions are passed to discussions list component with help of props, and survey questions list is divided each survey question with help of survey question component. The survey question component is render each survey question in a row and this component call star rating component. This star rating component is render 5 stars with animation. This animation is done using CSS. Initially all the stars stay light in color. If the user answer the survey question, color of the stars changes from light to dark color. If the user re rates the survey question, the charge is clearly displayed.

- Input is clicking on any one of the answers of a poll question.
- Output is the statistical data display.
- If the user has already given poll survey, then he/she is not allowed to answer for that question. They will be displayed with a message "You have already attempted this polls survey" and current result of that polls survey.
- For every survey question 5 stars are displayed.
- By clicking on the stars, rating is selected, sent to the server and re render the survey questions forum.

iii. Tasks

Tasks is a forum provided to the user similar to comments and response submission system, which provides an interface and whose functionality is similar to submissions in the Discussions module. An authorized (logged in) user is privileged enough to enter one's own response for a given task on a particular NGO activity and/or

edit it. The Task forum's functionality is designed with reference to the Facebook posts responses

For any NGO activity there may be tasks initiated by the NGO, if one is interested in responding to those tasks, one can always add a response within the text box and submit it, which will be further redirected to the display division where the responses of the user is displayed along with the time stamps of his/her responses.

The user then entitles to a multi-choice decision.

- One can add a response of one's own by typing in the Task forum's text space and submitting it.
- User can edit the submitted responses by double clicking on that particular response.
- User can check-out various previous responses and timestamps of those responses and also the submission status provided.

As in the figure [ref. Section 2.2 Project flow], the flow of the module goes in a loop. Starting with the Web API Utils before going to views, these are accessed in order to receive the previously entered or submitted data. Moving on to the Views element-connected to the Task Forum, to get the data as input. The whole tasks is split into 4 components TasksForum, AddResponses, ResponsesList and Responses (User interactions).

Stored data (previously saved) is initially collected through the Web API-Utils calls and the Tasksforum, which goes to ActionCreator elements (like Create, UpdateText functions). This data, via the dispatcher and call back functions is sent to the Stores element to store them locally. The Stores component then calls two other functions which again run parallel in another connected loop –Utils to Views then from Stores to Utils. The Utils (server calls) element repeatedly calls `getAllTasks ()` function and an `emitChange ()` function whenever the `getTasks ()` function is successful. `getAllTasks ()` function retrieves the previously entered list of comments and sends the data back to the Views element where it's displayed in the required context.

Similarly, the ResponseList function sends the following as attributes – create, edit, status, to the ActionCreators element and the cycle continues via Stores and Utils before returning to the Views element. All these responses in the list are editable by double clicking on them which shows up a new division to edit a particular response and after edition the modified time of the response will be updated like *just now, few minutes ago*. If the response is not sent to server then the loader symbol shows up conveying that the server updating of newly updated/modified response is not done.

2.1.4. User Donations Profile

A classic example of service and response submission system and user profile management system. An authorized (logged in) user can see all the donations corresponding to organization, search donations by their attributes and can see his/her own donations to the organization. And one can see the details about a particular donation when required. An attempt has been made to design a virtually traceable donation box, its functionality is 100% transparent to the user about the donations to the organization.

If one is interested in knowing about organization and wants to know the persons who donated the money to the organization, the one simply has to click on the donations button present in the navigation bar, and is directed to a donations division wherein, there are two entities; the donations search and a list of donations with primary details of donor, on what purpose the money was donated to the organization. And one can know the details of a particular donation, how much is donated and how much money/assets spent on what purpose. One can discuss about Donations and can take surveys and polls.

- One can see all the donations with primary details of donor.
- One can search or filter the donations by their category, donation id of donation time.
- User can see the total amount Donated by individual donor and total amount spent by organization.
- User can also see the total amount Donated to the organization and total amount spent.
- One can see the donations of one's own by clicking the Profile present in the navigation bar with his/her profile details.
- One can know about individual donation details and bills and make discussions and comments on it by clicking on that particular donation card.
- One can fill survey and see the polls.

Starting from the Views element- connected to the donation box, data input and display is split into DonationList () and SearchField () (User interactions).

In views, the data is requested from the server. An action is called after server call. From actions through dispatcher stores is called. In stores requested data is stored in the. In views donation list takes the data from getAllDonations() in stores. getAllDontions() retrieves the previously requested list of donations stored in stores and sends the data back to the Views element where it's displayed in the donation list component.

Similarly, the search Field function sends any one of the following as attributes – category, donation Id or time – to the server calls for donation list based on attributes. After server call an action is called. Requested data is stored in stores and sends the data back to the Views element where it's displayed in the donation list component.

If one clicks on any donation card in donations module or on one's own donation card in profile module, another component donationBills and a request is sent to the server and will take details of that particular card as response from server. Detail of card is stored in DonationsStore module. GetDonationDetails () function retrieves the data from the store to views. Data is displayed to the user.

FAQs

FAQs sub module is as common as any other commonly asked questions hosting forum. Designed with reference to common FAQs, the module hosts questions with the most number of occurrences from the Discussions sub module. The design style uses collapse elements arranged in an accordion-type layout. When a user clicks on the heading (in this case: the question) the corresponding division slides up and shows the respective answer.

2.1.6. Notifications Bar with Pagination

A classic example of notifications response system. The sub module requests notifications from the server and projects them to the user. Also with integrated pagination (Lazy Loading) applied to the list of notifications, there is lesser lag in loading the notifications list. The sub

module has been developed using React JS on flux architecture and pagination using the react-infinite plugin. An attempt has been made to design the notification box; its functionality with reference to the Facebook notifications. Also to design the pagination; its functionality was with reference to the Facebook Newsfeeds loading. Pagination development was applied in three different approaches.

A. Simple Method:

In which initially, data is loaded by making Ajax server call with parameters offset and length, when the user scrolls down to the bottom, the DOM element loads some more data.

In the initial states an EventListener function is called which continuously checks whether user scroll reaches to bottom of page or not. If user reaches to the bottom another Ajax call is made and get some more data using same parameters offset and length but offset value changes to offset+length. However, the disadvantage is that the previous DOM nodes (the nodes which are out of user's view) are not removed while scrolling down, and this may not show a good performance when scrollable items' lists can be tens or hundreds of thousands of items long. And the number of additional nodes may well reach the millions.

B. "react-waypoint" plugin:

Here too, the motto is same as above mentioned. However, the idea is based on waypoints; it is a check point defined by the user using keyword 'threshold' that is, the percentage of height of visible part of scrollable height in webpage. Data can be loaded by making Ajax call when user reaches this waypoint. This works in window and in all containers that can be scrolled.

C. "react-infinite" plugin:

Considering, there is long list of data as DOM elements placed in container that can be scrollable. If user scrolls down, all other elements are kept in DOM even if scrolled out of user's view. This might not be much efficient in performance. This can be solved by React-Infinite. In this also a part of the data is initially loaded, after user reaches to bottom of page again loads some more data. An advantage of the method over the previous one is that the previous DOM nodes are removed while scrolling down that it only stores nodes which are visible on user's view. Other DOM nodes are rendered as blank nodes. This can be work in mostly containers that can be scrolled.



Fig. Notifications Demo with Lazy Loading

2.2 Project Flow

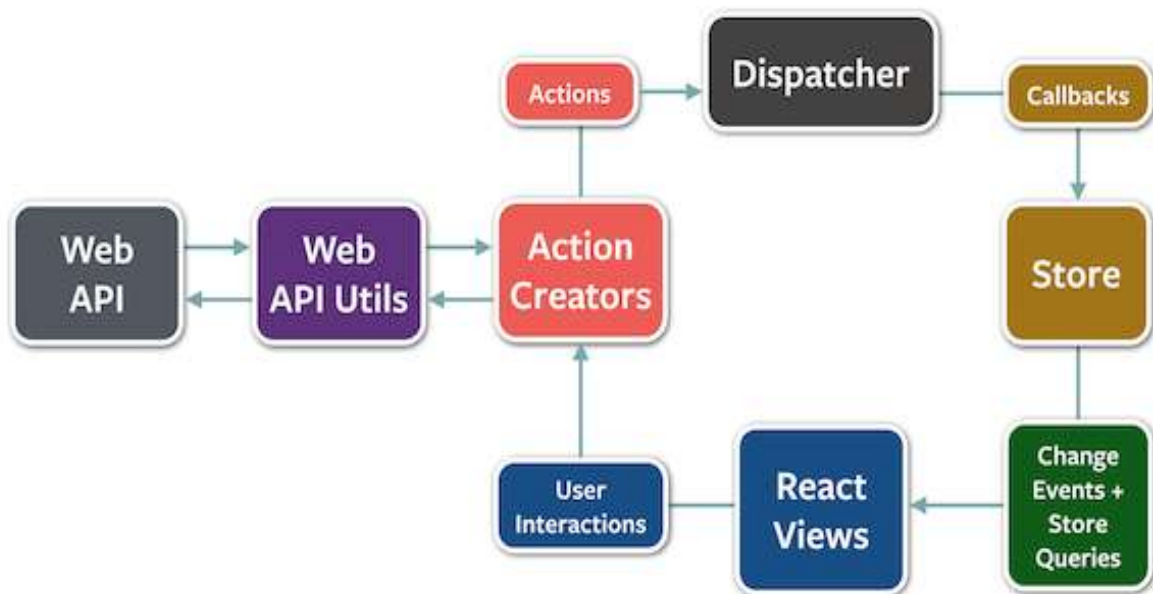


Fig. Project Flow (Flux)

In order to get a unidirectional architecture, the whole module has been built on the Flux framework. As in the figure, the flow of any sub module goes in a loop. The whole flow can be divided into the following components: **Action Creators**: Event objects which get created as actions occur.

Dispatcher: Receives actions and broadcasts payloads to all registered (pre-connected) call backs. Simply, an event dispatcher. However, specifically in Flux, the dispatcher has two distinct qualities:

- Call backs are not subscribed to individual events. Every payload is dispatched to every registered call back.
- Call backs can be deferred in whole or a part until the other call backs have arrived.

Stores: Containers for application state and Logic that have call backs registered to the dispatcher.

Views: React components that grab the state from stored and pass it down via props to child components

The flow runs in a looping unidirectional path. For example, **Action** methods are called and an action constant and action data are sent out. The **dispatcher** receives the event. It can dispatch differently based upon source (here: **Web API**). The dispatcher

dispatches the action constant and data to all registered call backs. If a **Store** has a call back registered with the Dispatcher, it will receive the dispatched event/data. The logic inside store acts on action constant and data, and then emits change event. The **views** get the change event and updates itself and child views. Views then call **action creators** and the **process repeats**.

2.3 UML Diagrams

2.3.1 Activity Diagram

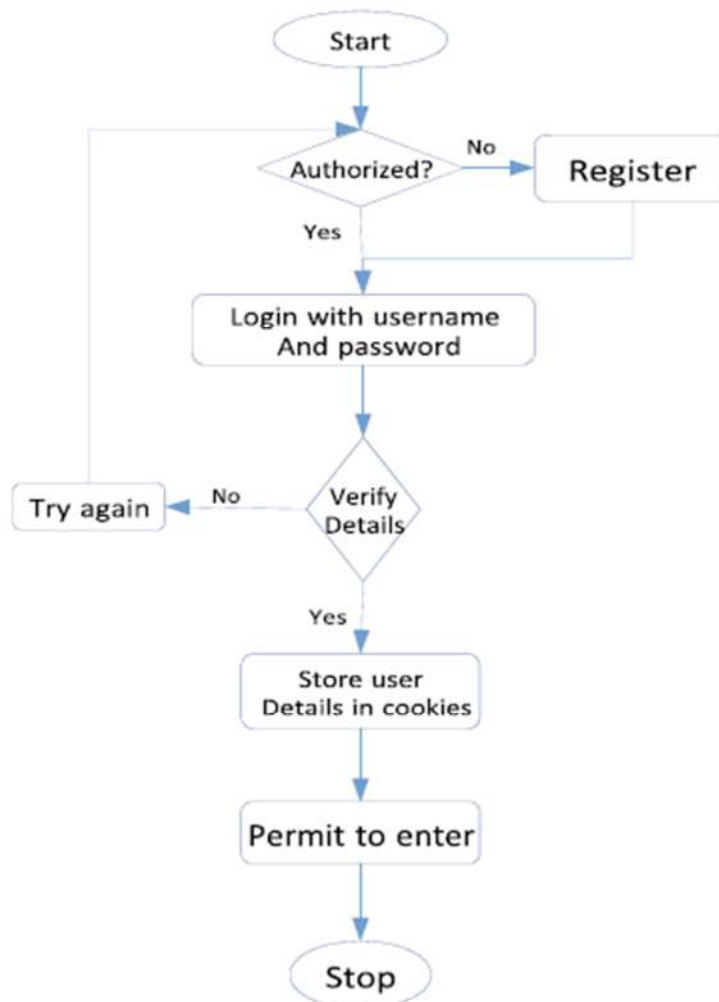
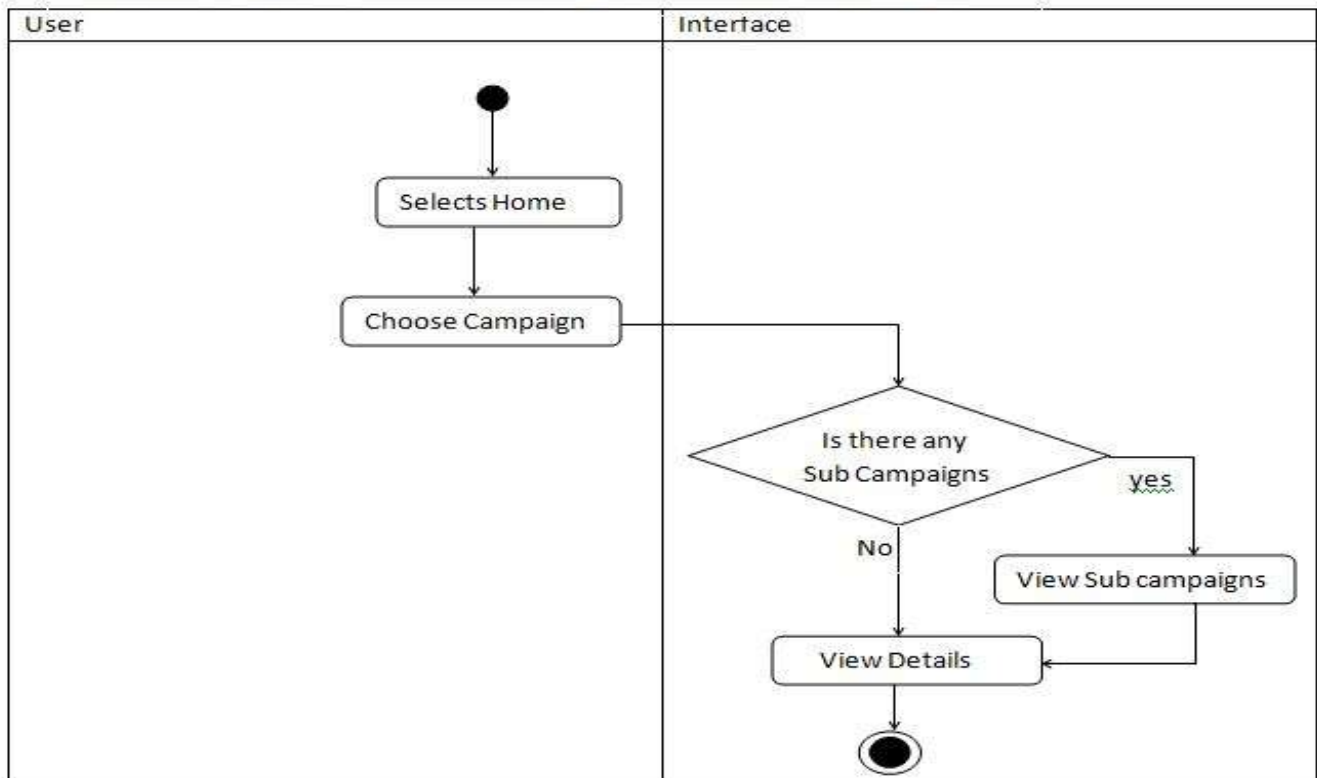
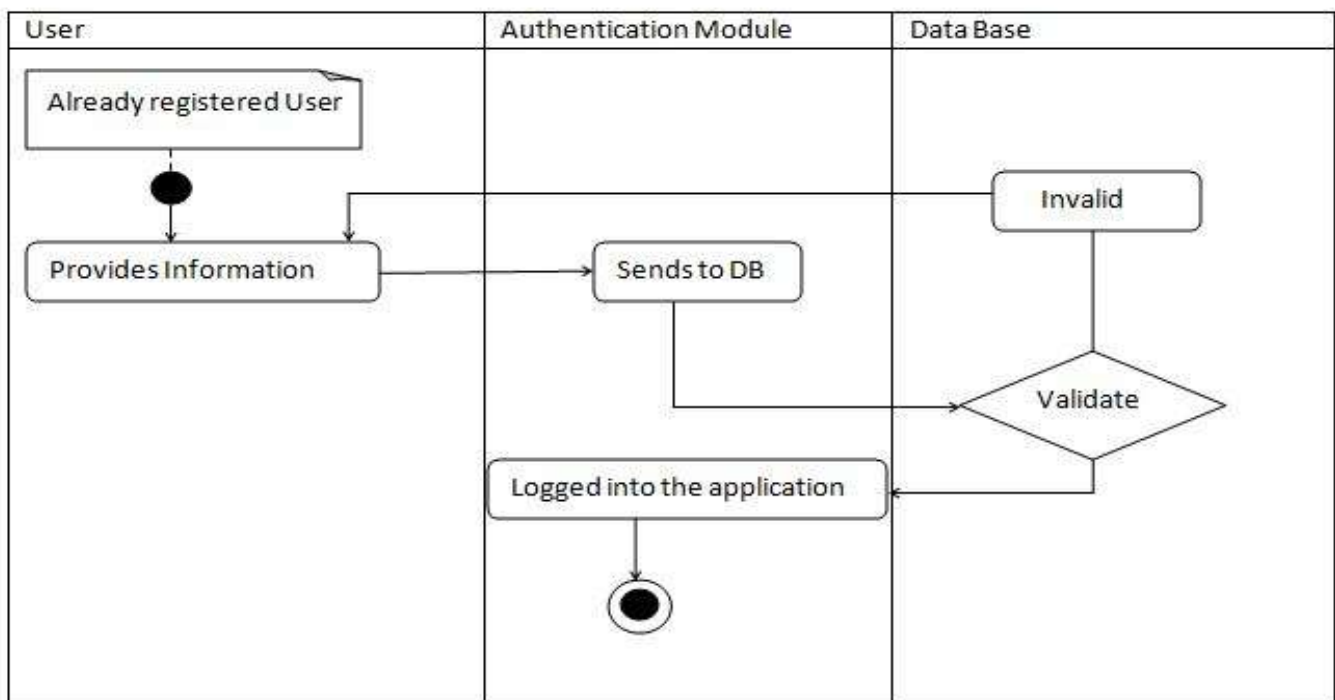


Fig. Login Module

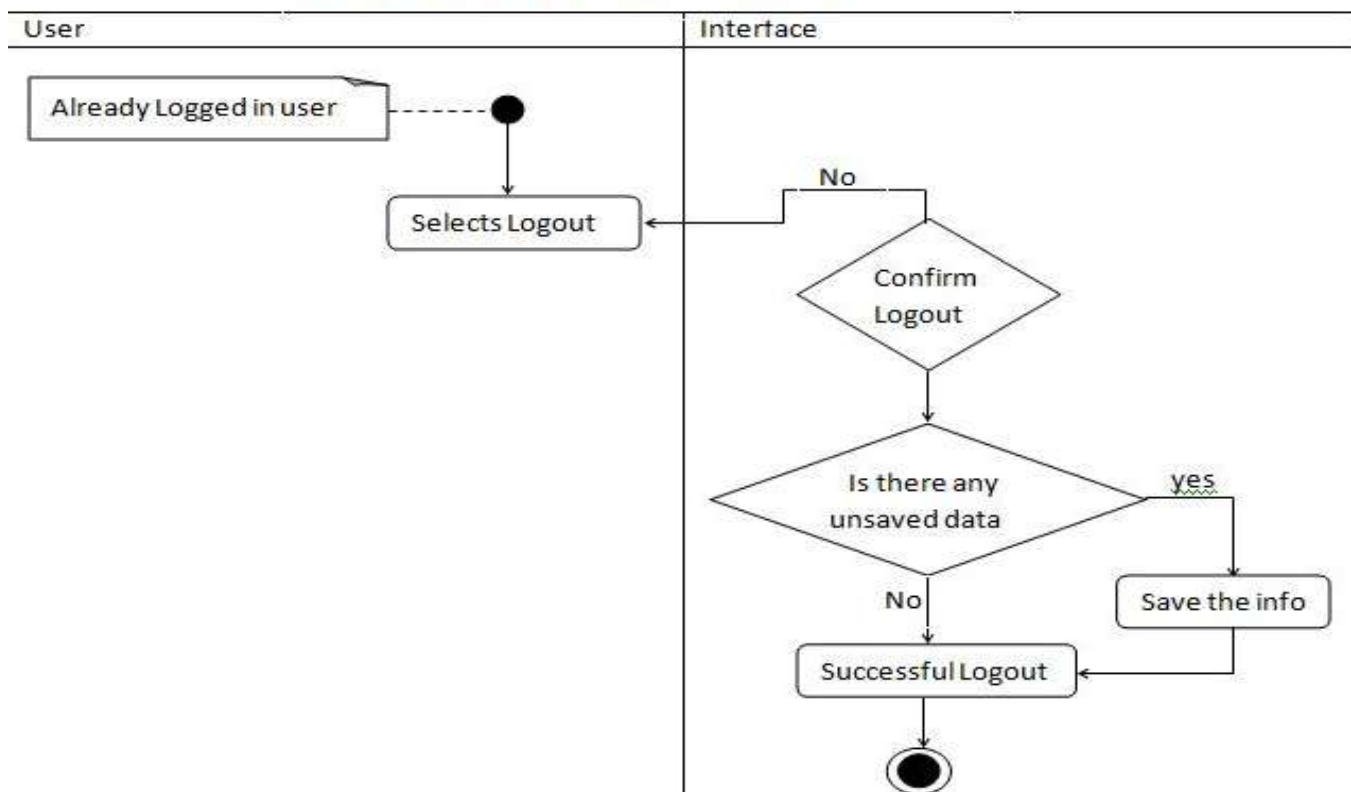
Activity Diagram to View Campaigns and Sub Campaigns



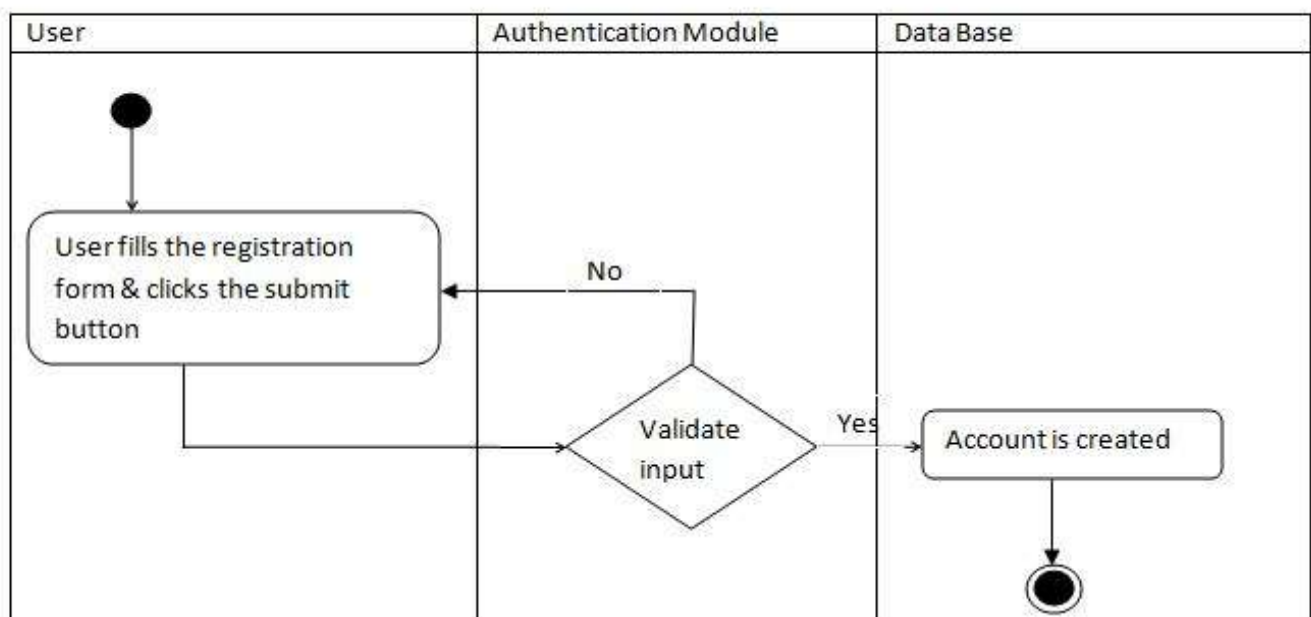
Activity Diagram for User Login



Activity Diagram for User Logout



Activity Diagram for User Registration



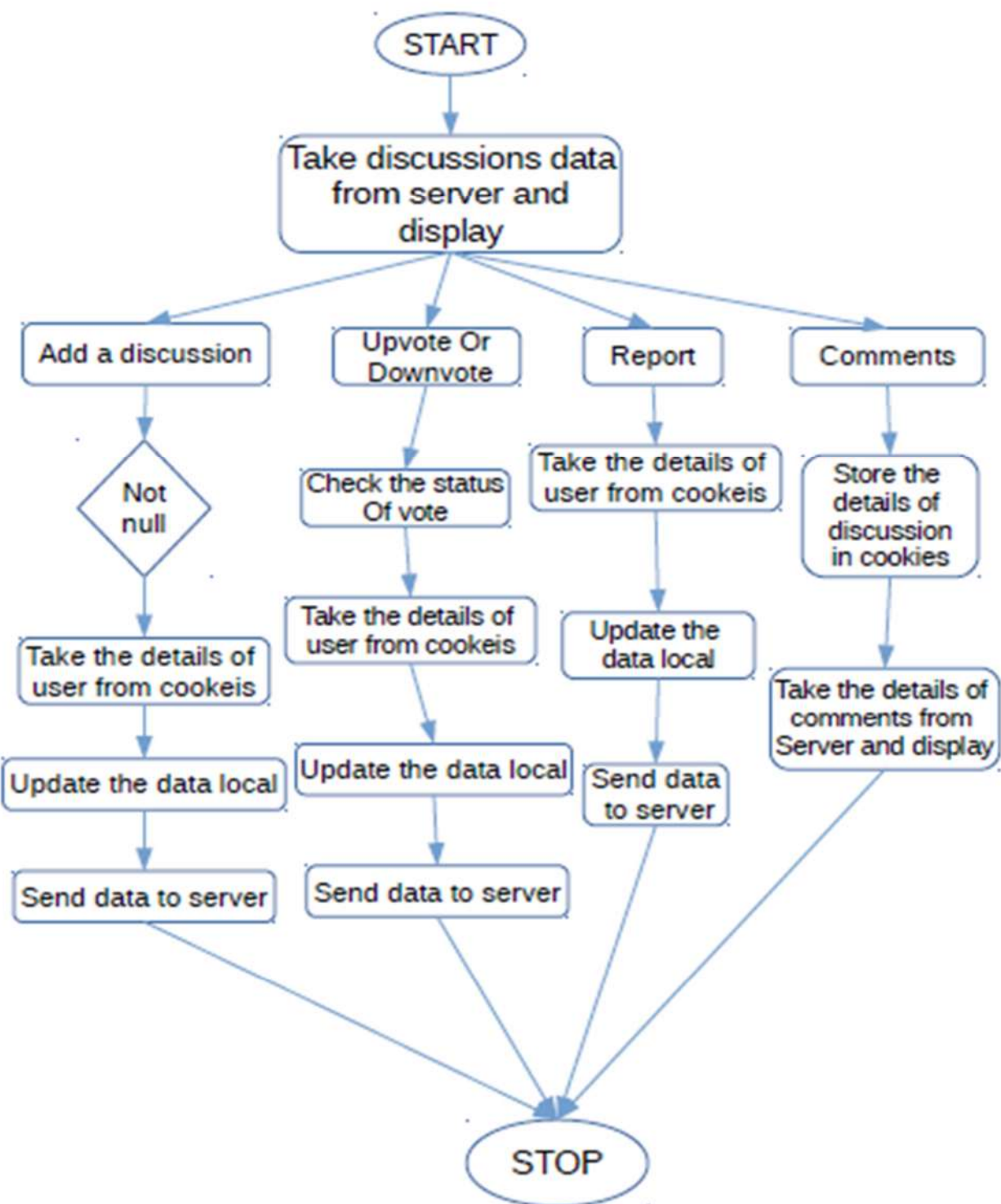


Fig. Discussions Module

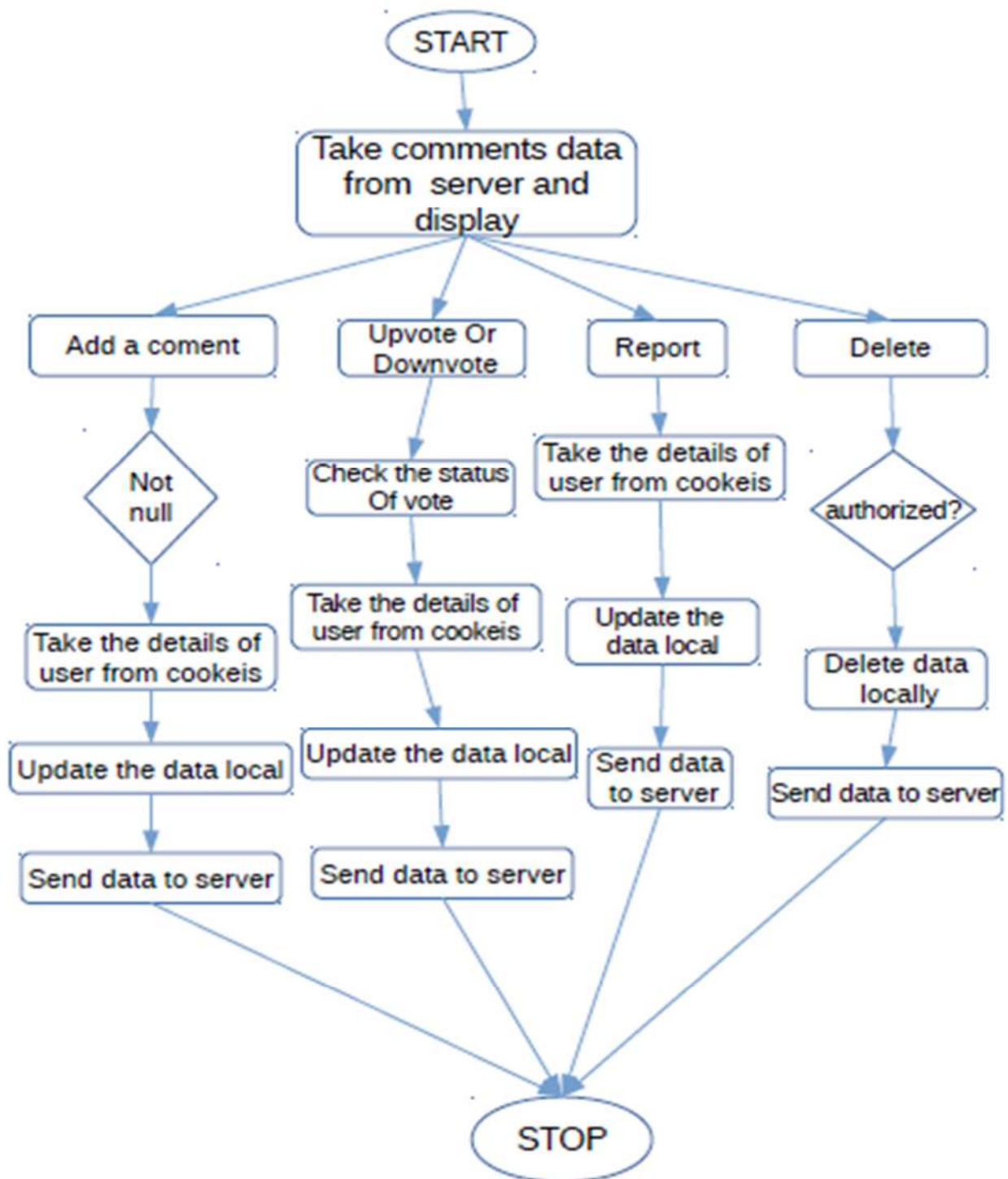


Fig. Comments Module

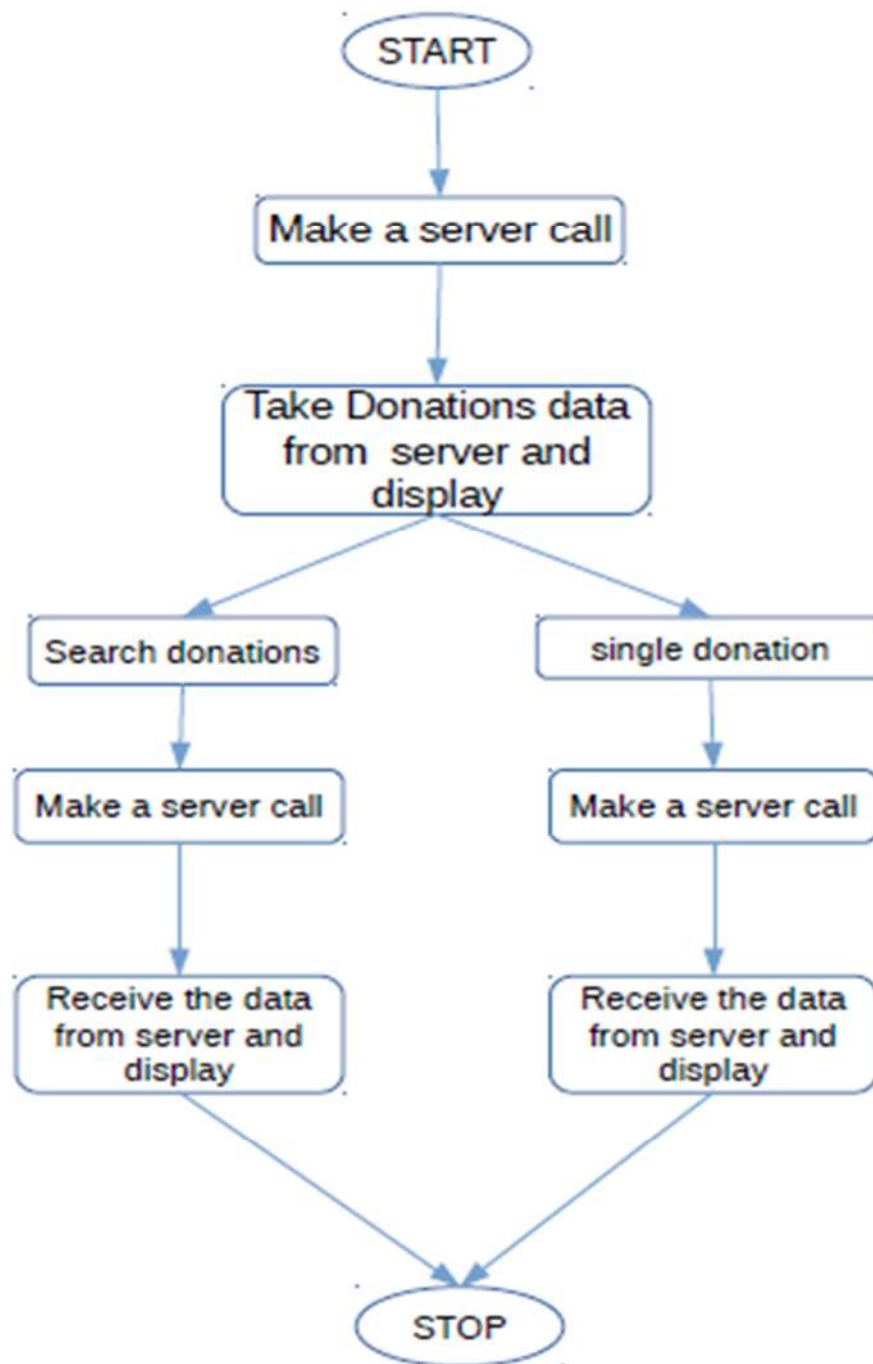


Fig. Donations Details

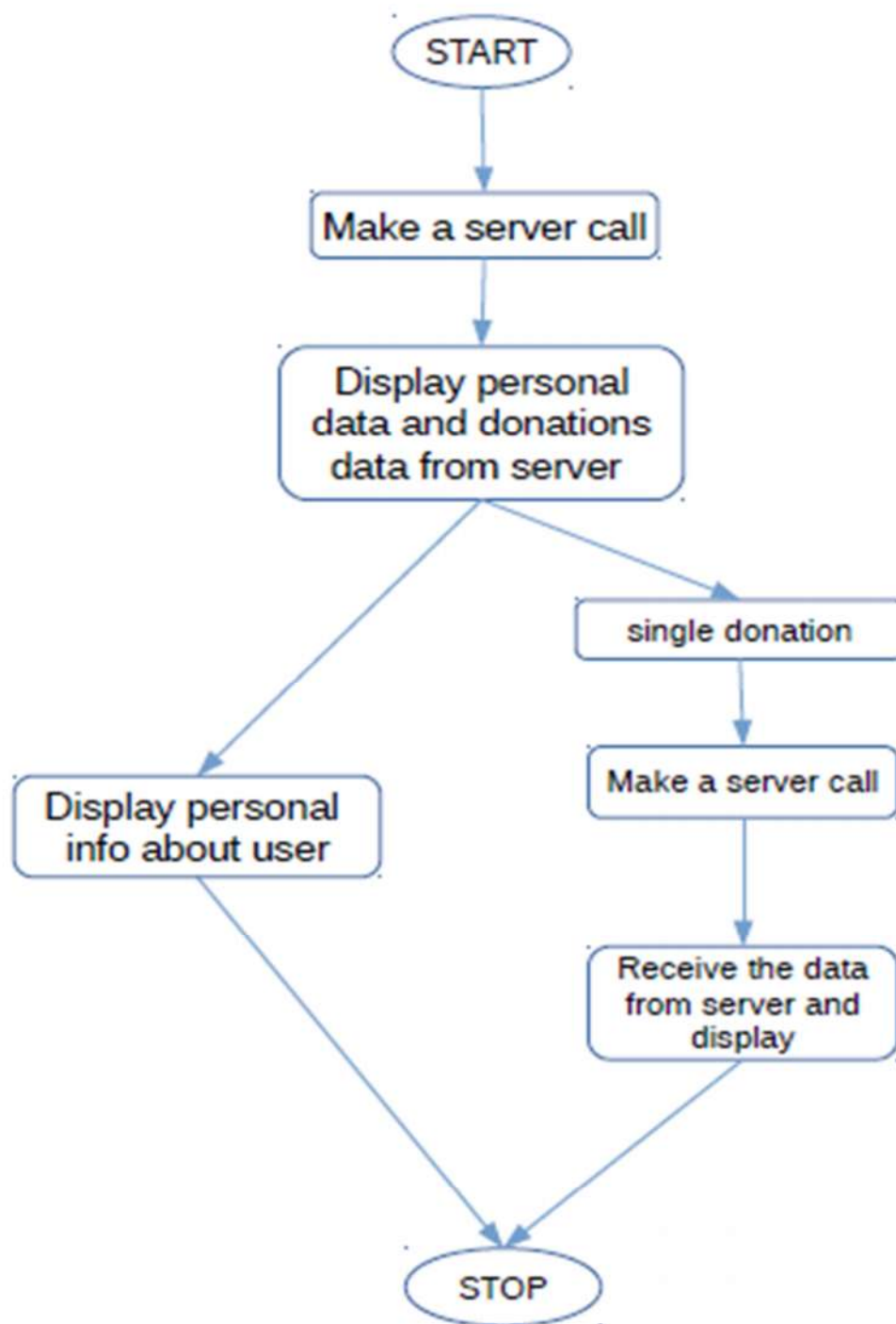


Fig. Profile Module

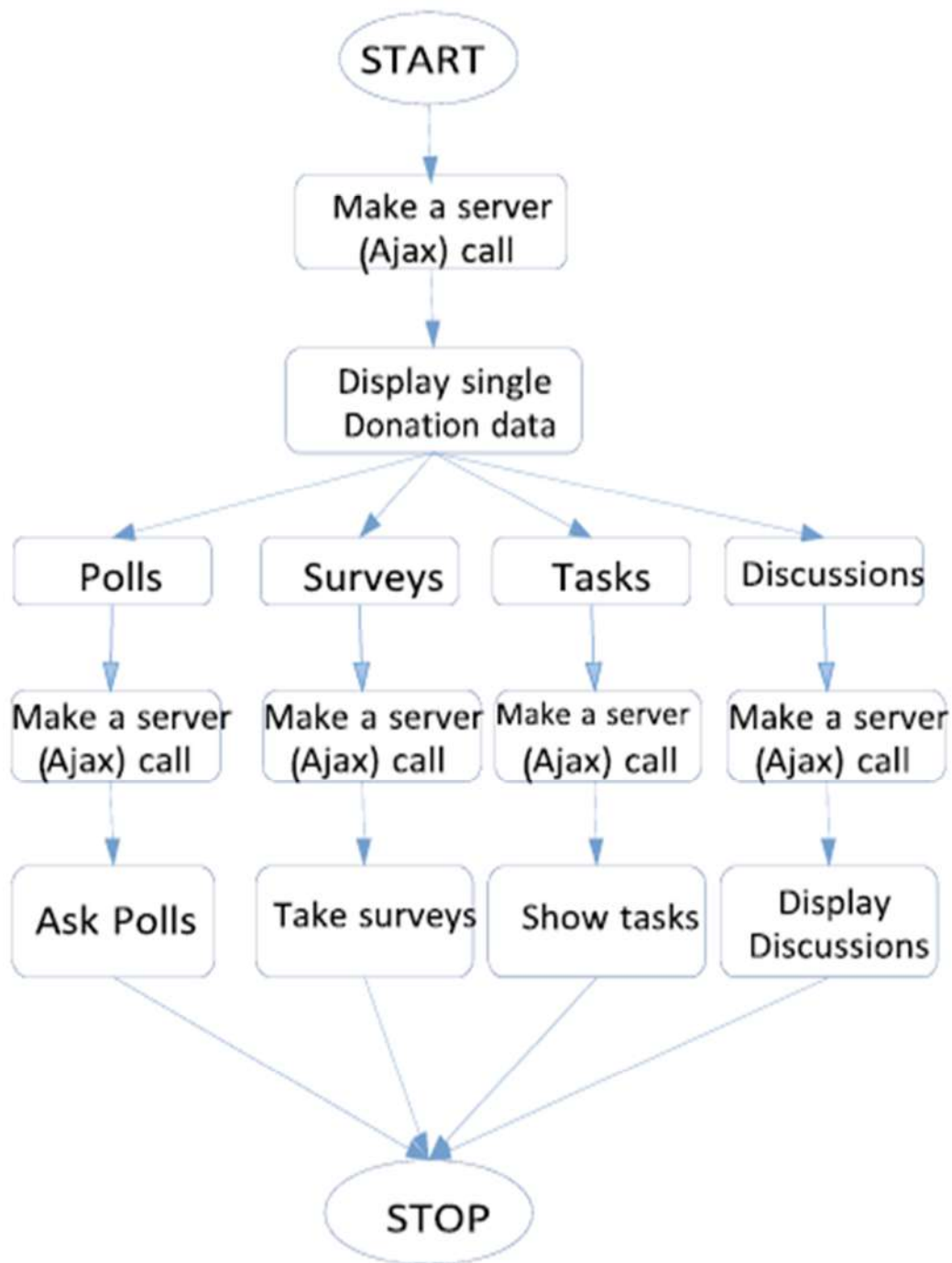


Fig. Single Donations Card

2.3.2 Sequence Diagrams

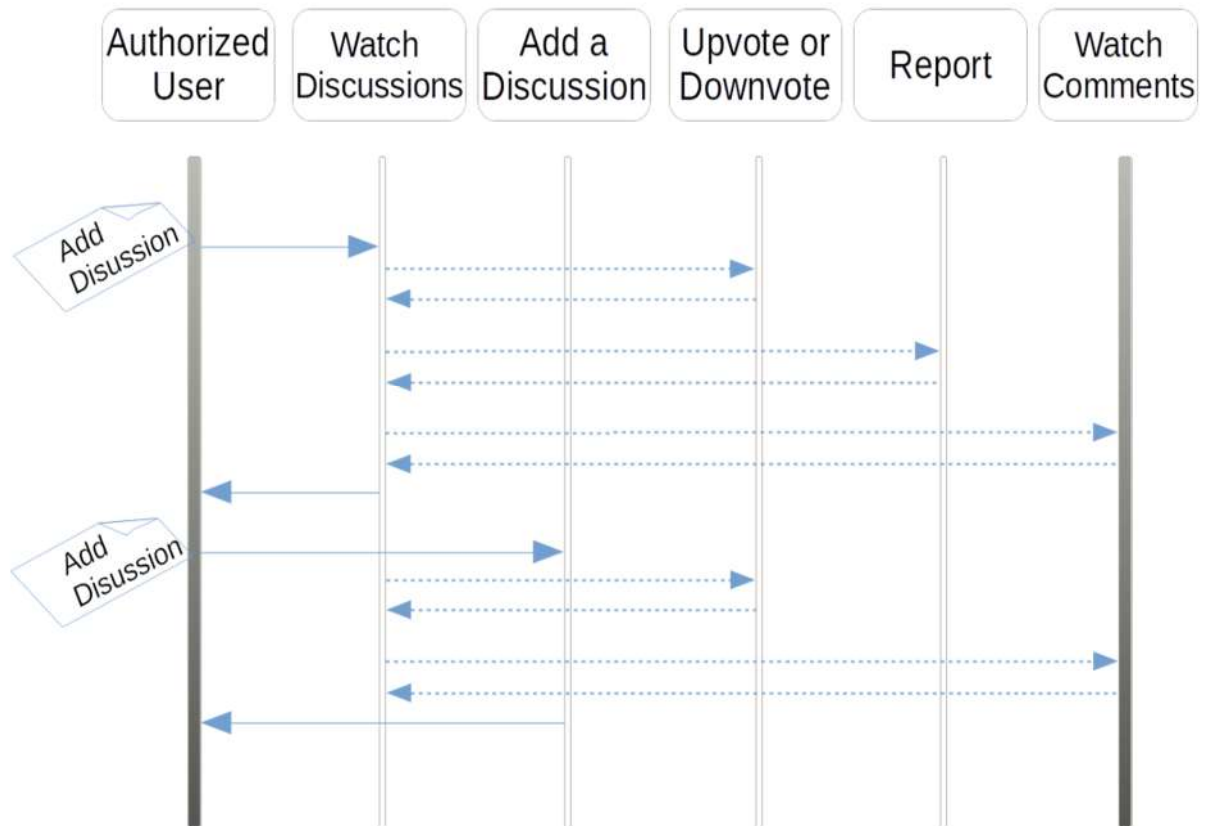


Fig. Discussions

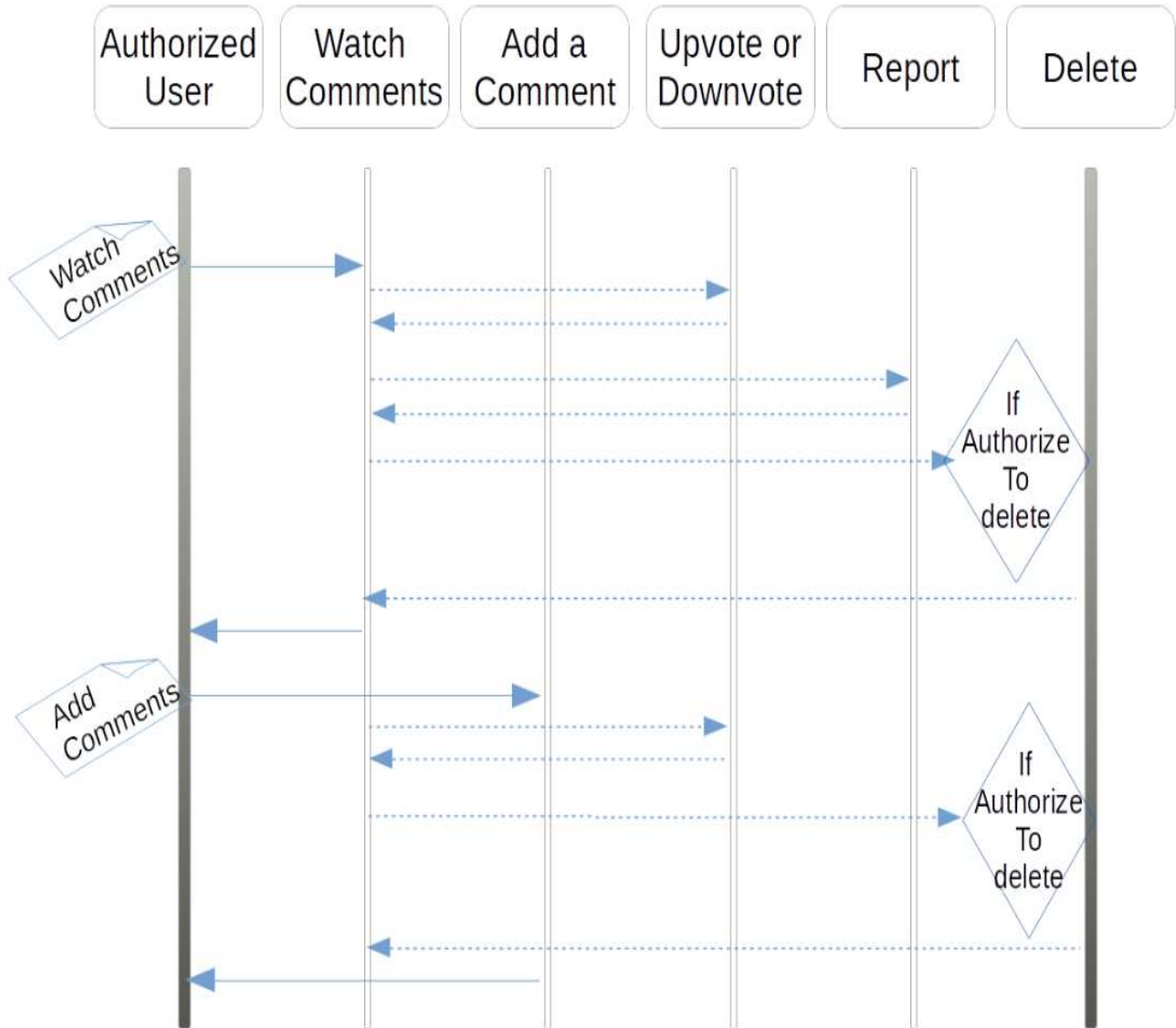


Fig. Comments

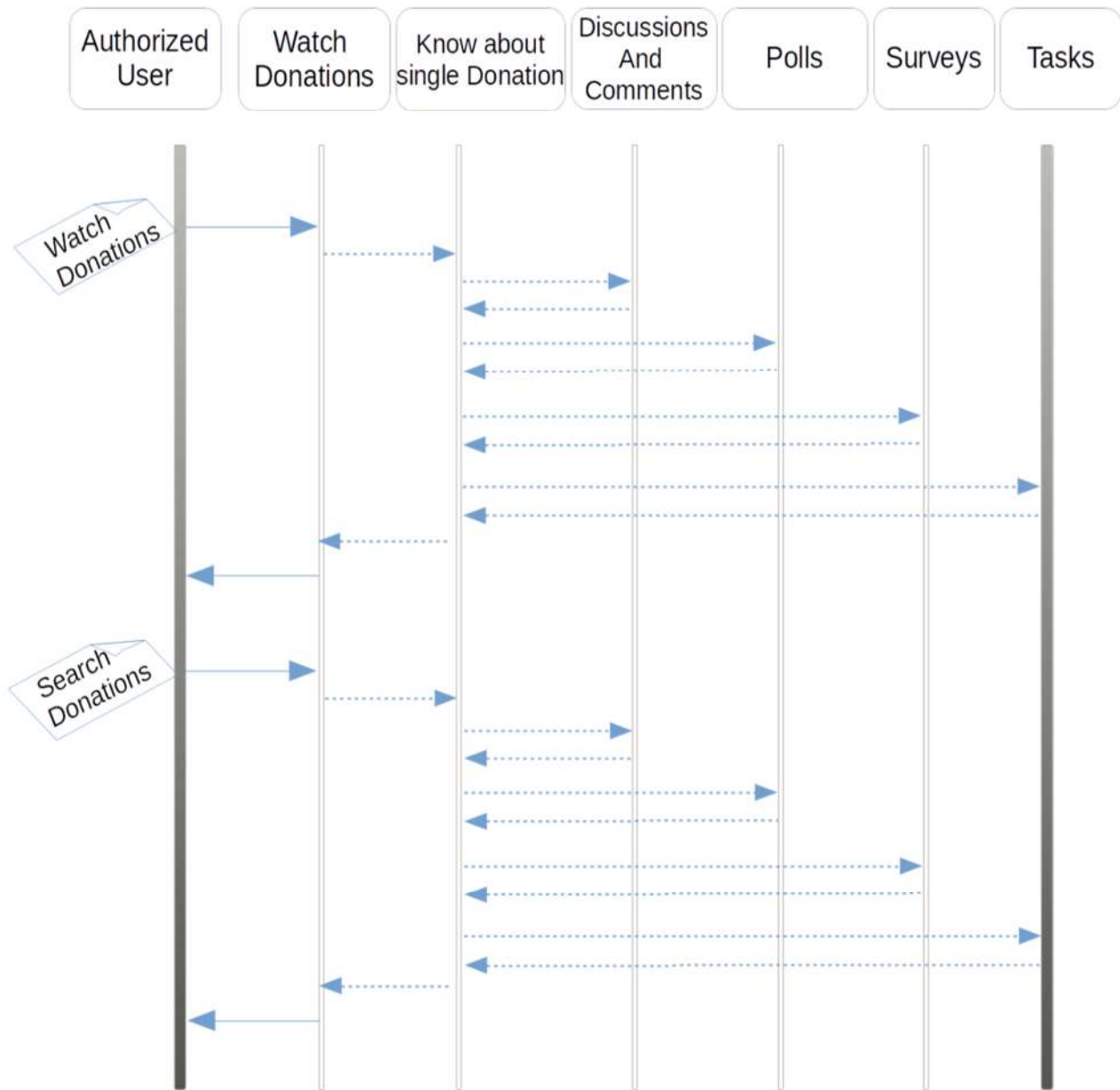


Fig. Donations

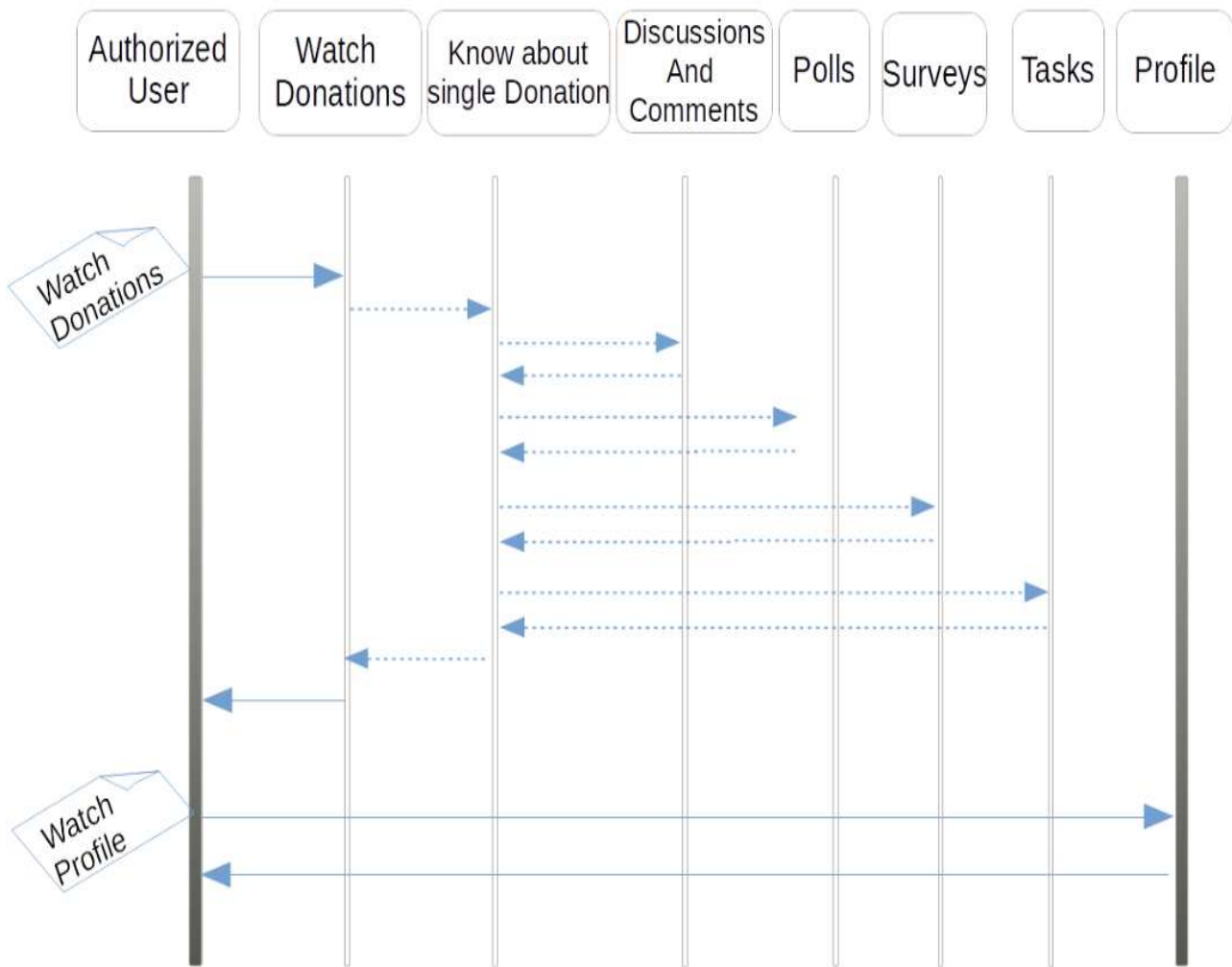


Fig. Profile

2.3.3 Use Case Models

Use Case Diagram for User interaction with Application

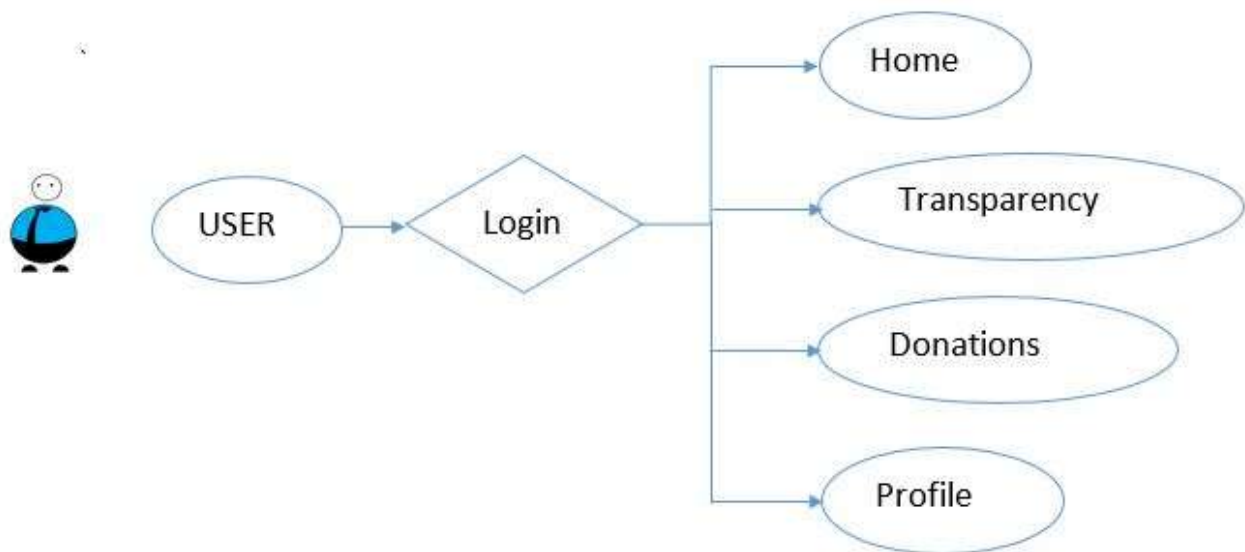
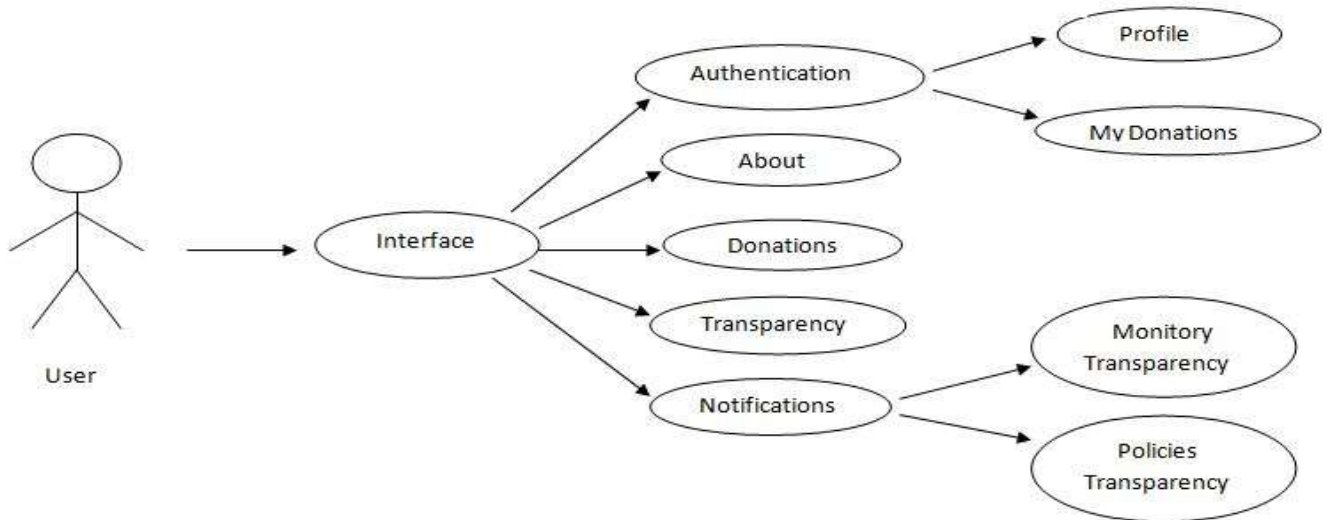


Fig. Login

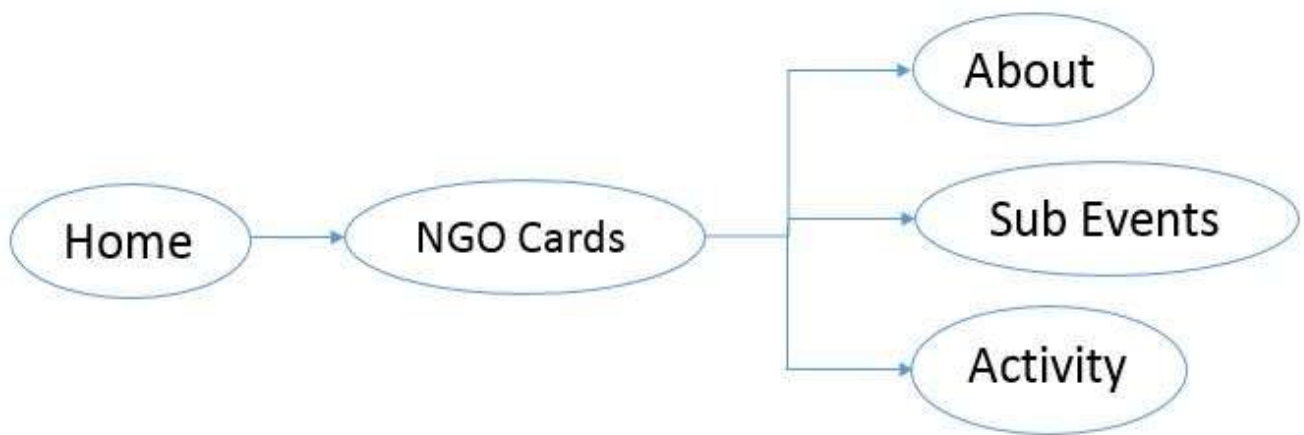


Fig. Home Component

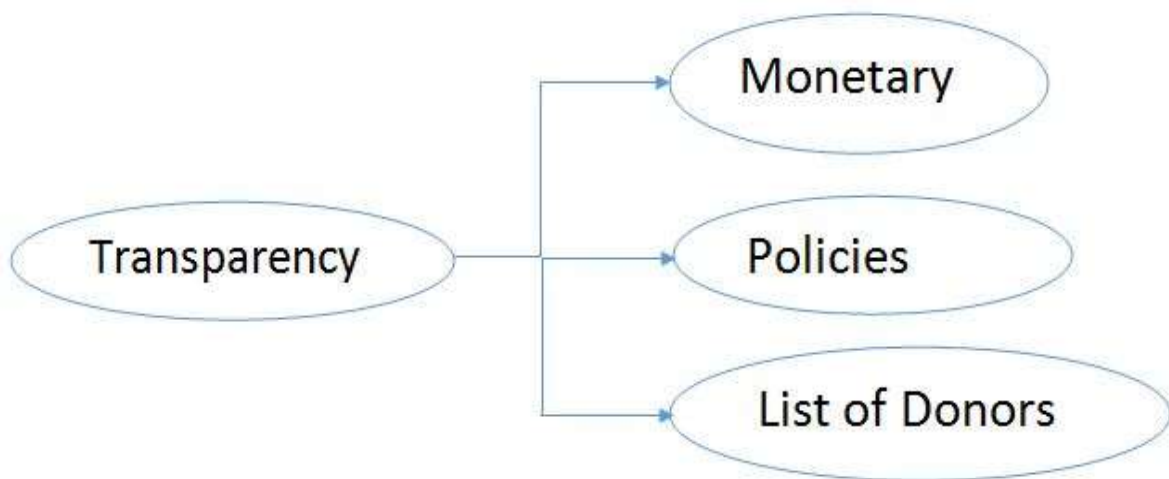


Fig. Transparency Component

2.4 User Interface : <http://maketheworldwonderful.org>

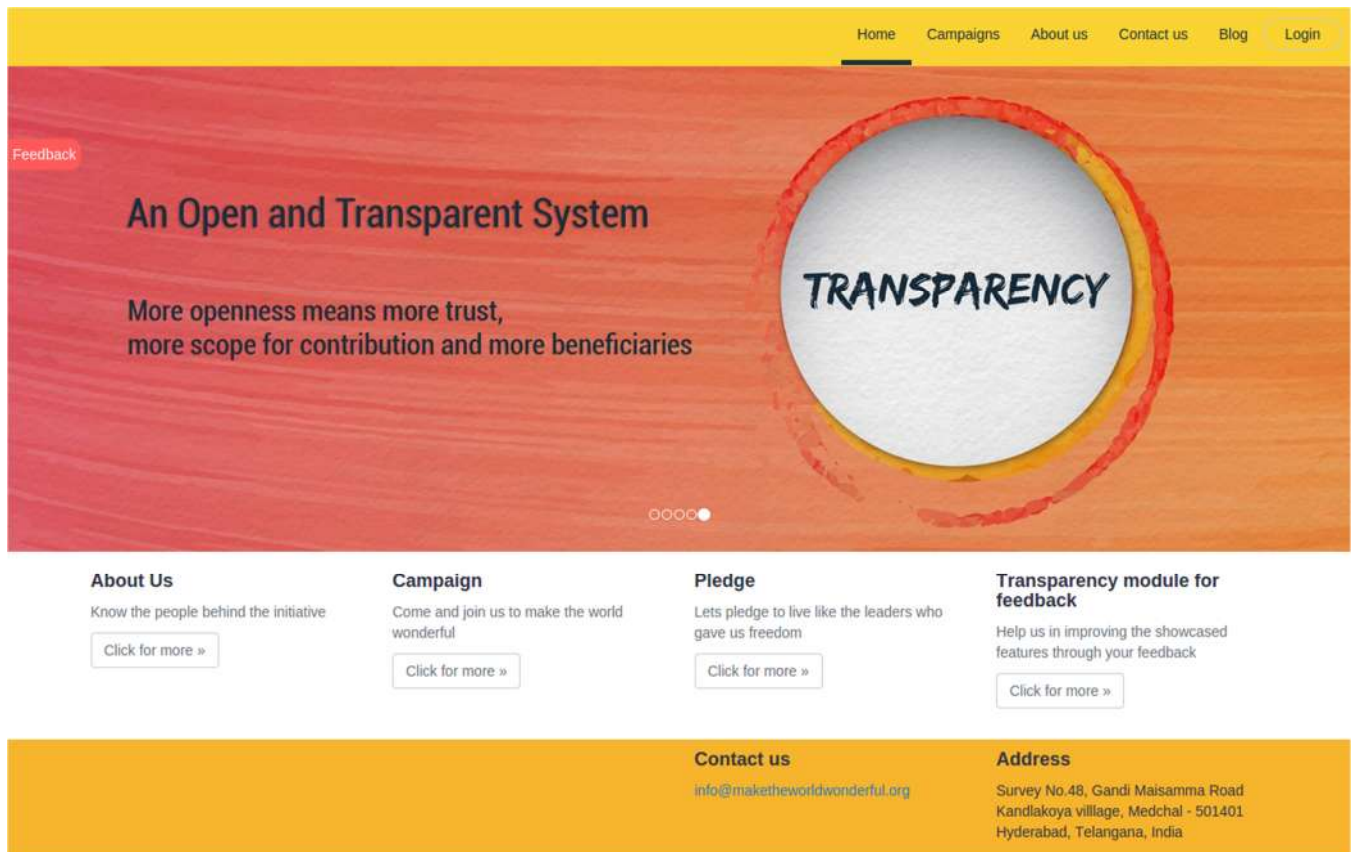


Fig. Home Page



Fig. Login

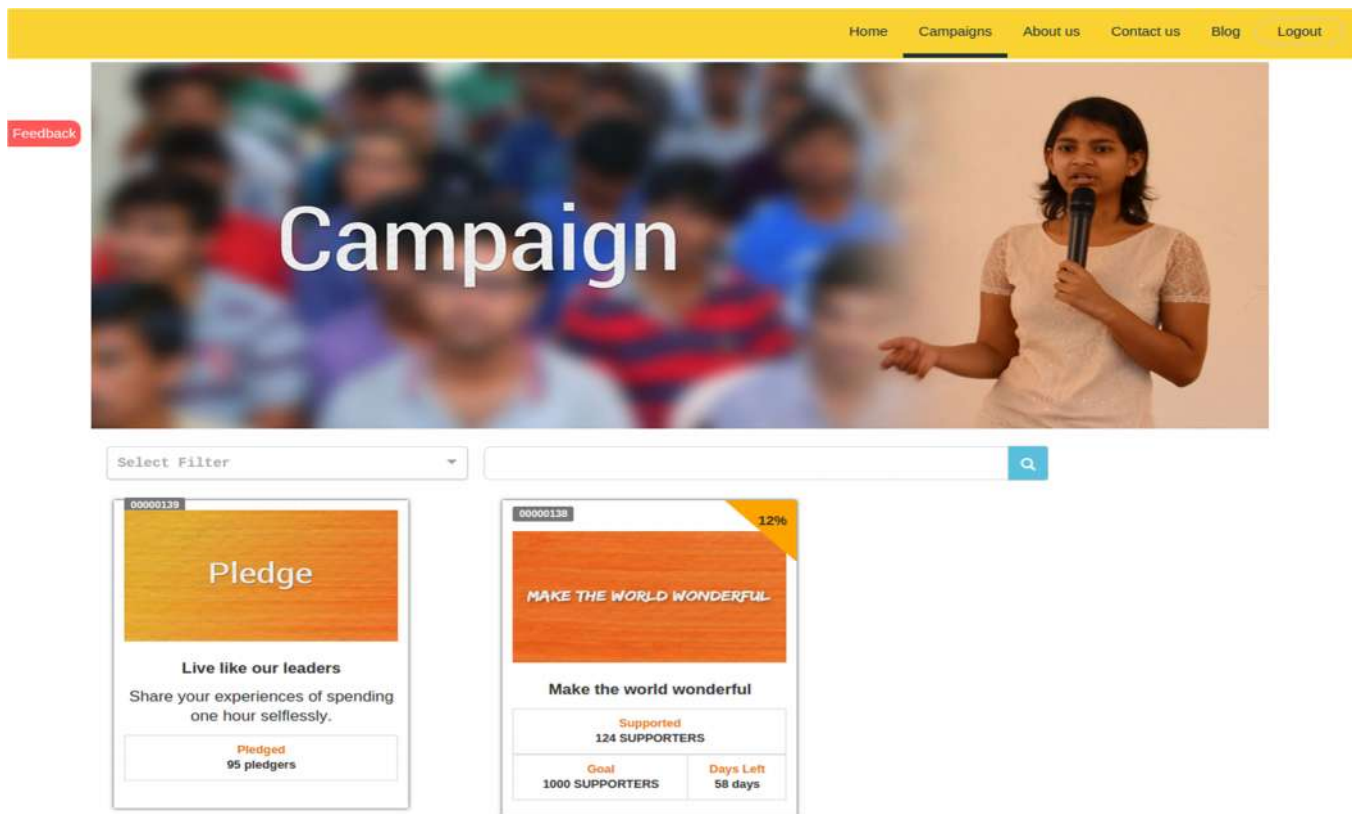


Fig. Campaigns

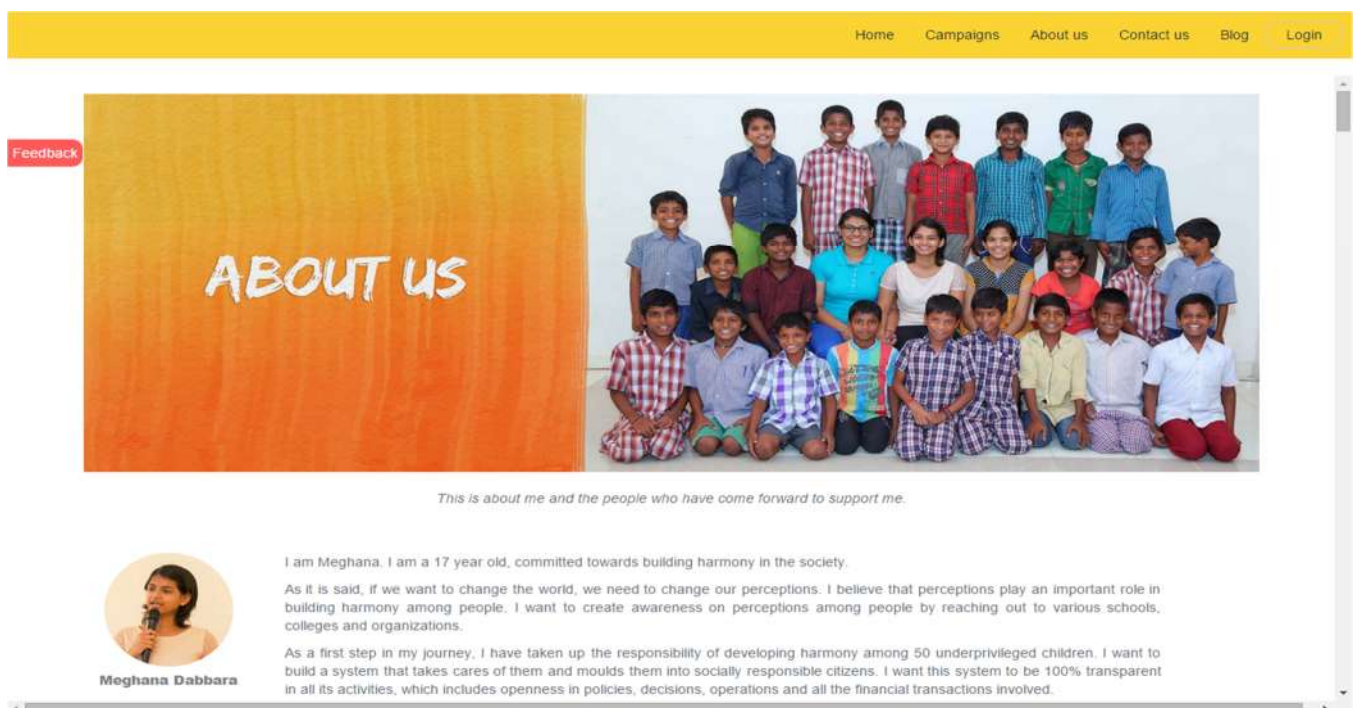


Fig. About Page



People Pledged

136 pledgers

Thanks Naga Manikanta
Satyanaraya Kotha for your Pledge!

[View activity](#) ▼

Pledge :

"I will live like the leaders , the saviours of our nation.
I solemnly pledge to devote one hour every month selflessly for others.
I promise not to expect any praises in return
I promise not to expect any personal benefits through this act.
My happiness lies in each successful attempt of my act."

You can share your experiences of spending one hour selflessly.

Activity

80*****	0
80***** pledged	👍
Tarun siriki	2
Tarun siriki pledged	👍
Deepak	1
Deepak pledged	👍
Eswarnath	1
89***** pledged	👍
Vivek Mudadla	0
Vivek Mudadla pledged	👍

[Load more](#)

Share your experiences



Share your experiences here!

Share

Fig. Campaign- Pledges

Monetary

Policies

Items

Donations

Given here are the sample bills for the financial transactions. Each card represents the details of a single bill. You can verify, raise a question & initiate discussion on each transaction.

Select Filter

General Monetary Donation - Sample

Monetary General Dummy Donation



✓ 41

DetailsTasksSurveysPollsDiscussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation



✓ 125

DetailsTasksSurveysPollsDiscussions

Monetary Food Sample Donation

Monetary Food Sample Donation



✓ 15

DetailsTasksSurveysPollsDiscussions

Monetary Education Sample Donation

Monetary Education Sample Donation



✓ 6

DetailsTasksSurveysPollsDiscussions

Monetary Education Sample Donation

Monetary Education Sample Donation



✓ 2

DetailsTasksSurveysPollsDiscussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation



✓ 2

DetailsTasksSurveysPollsDiscussions

Sample Bill : Education-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga



✓ 16

DetailsTasksSurveysPollsDiscussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga



✓ 1

DetailsTasksSurveysPollsDiscussions

Sample Bill : Medical-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga



✓ 5

DetailsTasksSurveysPollsDiscussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga



✓ 6

DetailsTasksSurveysPollsDiscussions

.Fig. Sub Campaigns

Monetary

Policies

Items

Donations


Given here are the sample bills for the financial transactions. Each card represents the details of a single bill. You can verify, raise a question & initiate discussion on each transaction.

Select Filter

Q

General Monetary Donation - Sample

Monetary General Dummy Donation




✓ 41

DetailsTasksSurveysPollsDiscussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation




✓ 125

DetailsTasksSurveysPollsDiscussions

Monetary Food Sample Donation

Monetary Food Sample Donation



✓ 15

DetailsTasksSurveysPollsDiscussions

TASK:

What do you feel about Transparency module?

Your answer


Answer

Responses:

No responses yet!

Monetary Education Sample Donation

Monetary Education Sample Donation




✓ 6

DetailsTasksSurveysPollsDiscussions

Monetary Education Sample Donation

Monetary Education Sample Donation




✓ 2

DetailsTasksSurveysPollsDiscussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation




✓ 2

DetailsTasksSurveysPollsDiscussions

Sample Bill : Education-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga




✓ 16

DetailsTasksSurveysPollsDiscussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga




✓ 1

DetailsTasksSurveysPollsDiscussions

Sample Bill : Medical-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga




✓ 5

DetailsTasksSurveysPollsDiscussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga



✓ 6

DetailsTasksSurveysPollsDiscussions

Fig. Sub Campaigns – Tasks, Opinions

[Home](#)
[Campaigns](#)
[About us](#)
[Contact us](#)
[Blog](#)
[Logout](#)

Monetary

Policies

Items

Donations

Given here are the sample bills for the financial transactions. Each card represents the details of a single bill. You can verify, raise a question & initiate discussion on each transaction.

Select Filter

General Monetary Donation - Sample

Monetary General Dummy Donation

✓ 41

Details Tasks Surveys Polls Discussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation

✓ 125

Details Tasks Surveys Polls Discussions

Monetary Food Sample Donation

Monetary Food Sample Donation

✓ 15

Details Tasks Surveys Polls Discussions

Topic

Select catagory

Initiate Discussion

Popular

G.Reshika Reddy

0

dtd

Monetary Education Sample Donation

Monetary Education Sample Donation

✓ 6

Details Tasks Surveys Polls Discussions

Monetary Education Sample Donation

Monetary Education Sample Donation

✓ 2

Details Tasks Surveys Polls Discussions

Monetary Medical Sample Donation

Monetary Medical Sample Donation

✓ 2

Details Tasks Surveys Polls Discussions

Sample Bill : Education-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga

✓ 16

Details Tasks Surveys Polls Discussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga

✓ 1

Details Tasks Surveys Polls Discussions

Sample Bill : Medical-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga

✓ 5

Details Tasks Surveys Polls Discussions

Sample Bill : Food-100.000

The bill amounts to 100.000 INR.It is verified by and authorized by baxinga

✓ 6

Details Tasks Surveys Polls Discussions

Fig. Sub Campaigns – Discussions

Given here are the extensive details of every contribution made and how it is being utilized.

Feedback

Select Filter		Q
Sample Donation	Amount Donated : 1000 INR	Amount spent : 0 INR
Sample Donation	Amount Donated : 1000 INR	Amount spent : 0 INR
Sample Donation	Quantity Donated : 1000 INR	Quantity spent : 0 INR
Sample Donation	Amount Donated : 1000 INR	Amount spent : 200 INR
Sample Donation	Quantity Donated : 1000 kg	Quantity spent : 0 kg
Sample Donation	Amount Donated : 1000 INR	Amount spent : 400 INR
Sample Donation	Quantity Donated : 1000 INR	Quantity spent : 0 INR
Sample Donation	Amount Donated : 1000 INR	Amount spent : 200 INR
Sample Donation	Quantity Donated : 1000 INR	Quantity spent : 0 INR
Sample Donation	Amount Donated : 1000 INR	Amount spent : 200 INR

6000 INR

Total amount donated

1000 INR

Total amount spent

List of Items donated

Item	Total donation	Balance left
Rice	1000 kg	1000 kg
Pens	2000 INR	2000 INR
Aspirin	1000 INR	1000 INR


Fig. Activities

Monetary
Policies
Items
Donations

Given here is the sample inventory list which can be updated dynamically. You can verify, raise a question & initiate discussion on each item available.

Pens(Education)

Pens




✓ 7

Details
Tasks
Surveys
Polls
Discussions

Aspirin(Medical)

Aspirin




✓ 4

Details
Tasks
Surveys
Polls
Discussions

Rice(Food)

Rice in Kg




✓ 0

Details
Tasks
Surveys
Polls
Discussions

Pens General Sample Donation

Pens General Sample Donation




✓ 2

Details
Tasks
Surveys
Polls
Discussions

Aspirin Medical Sample Donation

Aspirin Medical Sample Donation




✓ 0

Details
Tasks
Surveys
Polls
Discussions

Rice Food Sample Donation

Rice Food Sample Donation




✓ 0

Details
Tasks
Surveys
Polls
Discussions

Pens Education Sample Donation

Pens Education Sample Donation




✓ 1

Details
Tasks
Surveys
Polls
Discussions

NoteBooks(Education)

Description



✓ 0

Details
Tasks
Surveys
Polls
Discussions

Fig. Sub Campaigns - Items Requirement

Home
Campaigns
About us
Contact us
Blog
Logout

Feedback

CATEGORY

FEEDBACK

ATTACHMENT

Choose Files

No file chosen

Submit

Fig. Feedback

Challenges encountered

- If the server call was to take a time only display initiate discussion form, the module should get the list to display initiate discussion and discussion list.
- Setting fill-color for glyphicons. The glyphicons are similar to fonts and setting up a fill-color for one such element is not possible.
- Setting the DoubleClick-Edit function on the user provided task.
- In react-infinite plugin, container height should be fixed. That makes the container non-responsive. This can be solved by setting height state whenever window height changes.

Improvisations and Extensions

- Storage issues: storing data in cookies is not a reliable method owing to two reasons - Security issues; cookies can be exploited easily to gain data. Also, cookies have very low storage space (<4kb).

Hence, only the server response (a unique access token per user) is stored in the cookies and the rest of the information is sent to the server storage.

- A time delay existed in data retrieval when running server calls from the Stores element due to asynchronous calls made. To cover it, the server calls were shifted from Utils to Views element.
- The problem which can be improvised with the task module is that the order of the responses display which differs in every login session if a new response is entered which can be done by using filter/sort/order by options in the server calls.

References

- [1] TodoMVC, a todo-list application model based and designed on React JS, <https://facebook.github.io/flux/docs/todo-list.html>
- [2] React JS documentation, <https://facebook.github.io/reactn>
- [3] Flux Documentation, <https://facebook.github.io/flux>
- [4] Facebook MVC Vs Flux, <http://www.infoq.com/news/2014/05/facebook-mvc-flux>
- [5] React Infinite, <https://github.com/seatgeek/react-infinite>
- [6] www.stackoverflow.com