SAN JOSÉ STATE UNIVERSITY

Smart-City

CMPE 281 - Final Project Report

By
TEAM 10
Poojitha Amin (011811306)
Sneha Vadakkemadathil (011810721)
Anupama Upadhyayula(011325041)
Abinethri Santhanam (008634782)

Advisor: Dr. Jerry Gao

Table of Contents

Chapter 1: Introduction

Social Networking has become a part of our lives in today's world. Communication through social platforms helps us stay connected. We need such platforms for Recognition, Relationships and Resources. These are required not only for us human beings, but the government too. Government requires a medium of communication for recognition, relationship and resources, as an always-on service medium for a nation or city's huge population that convert to users. This can be achieved through a Cloud based Smart City Community Platform.

Smart City Social Network Platform, in a nutshell is a service-oriented community platform for city residents to collaborate with the government, and vice versa. It not only helps the city residents to communicate with each other, but acts as a one-stop solution for the citizens to easily file and track service requests. The platform also allows cross-communication with other social networking platforms like Facebook or Twitter.

Social networks, such as Facebook and LinkedIn are predominantly relationship-driven, which means people go there to connect with their friends and peers. Smart communities in this sense, take social networks a step further in that they are purpose-driven. Social networks are

bound together by pre-established interpersonal connections, whereas connected communities are bound together by a common interest or topic. This makes it a more complex, overlapping and nested structure.

Chapter 2: A Review, Classification and Comparison of Social Community Networks and Technologies

2.1 Review

Cloud based Smart-City social network platform

Features :

- ✓ Service-oriented cloud community platform for citizens and government
- ✓ Intuitive UI to file and track citizen requests
- ✓ Smart interface for easy inter-communication
- ✓ Integrated cross-collaboration with other social platforms
- ✓ High availability with easy cloud deployment
- ✓ Easily scales to millions of users
- ✓ Distributed design helps achieve high cluster and node availability
- ✓ Extensible and scalable component design
- ✓ Supports multi-language platform

2.2 Classification

Open Source Social Network Platforms	Description

Oxwall	It has features of blogging, photo sharing, advertisement file and video sharing amongst friends. It uses PHP and MySQL environment for social network development.
Elgg	It has features such as activity streams, user management, powerful data model, web services API, plugin API and access controls. It operates on Apache, PHP, MySQL and Linux environments.
XOOPS	YouTube, friend list, pictures, communities, mp3 tracks, wall to post messages are some of the features of XOOPS.Environments used are MySQL, Linux, PHP, and Apache.
OSSN	Open Source Social Network (Ossn) is a rapid development social networking software written in PHP. OSSN can be used to build different types of social apps like Private Intranets, Public/Open Networks and Community.
Mixxt	It enables collaboration, communication, and knowledge management through a single tool. It provides community functions such as Forum, images, events & wikis.

2.3 Feature Set Comparison of Social Community Networks:

	Exo	HumHub	Diaspora	Elgg	Open source Social Network
Scalability	Scalability is an issue. Currently supports small groups. Eventually would end up supporting bigger networks	Designed primarily for intranets. Scalability for large groups will be an issue.	Scales very well especially with the decentralized architecture.	Scalability is a concern in Elgg as well.	Scales very well with its superior architecture and design
Wiki	Exo lacks the wiki feature in its feature list.	Humhub has support for document/file sharing and discussion around topics.	Diaspora lacks the wiki feature as the primary focus seems be on hashtags.	Elgg has support for Wiki	Open source network has very good support for wiki
Source code license	LGPL	GNU AGPL	GNU AGPL	GPLv2/MIT	OSSN License v3
Event calendar	Has good support for event calendar	Supports event calendar	Supports event calendar	Does not support calendar	Supports calendar

Chapter 3: System Infrastructure Design

The cloud based system infrastructure supports both software and hardware components such as servers, storage and virtualization features, all of which will be used to support the computing requirements of the cloud computing model.

Cloud computing infrastructure provides an abstraction layer by virtualizing the resources and providing them to the user via a graphical interface and api enabled command-line. This ensures that the users have simplified access to the resources without needing to know the underlying data stores.

These are hosted by a third party and provided to the users over a network or internet. These include virtual machines and components such as servers, memory, firewall, load balancers and storage.

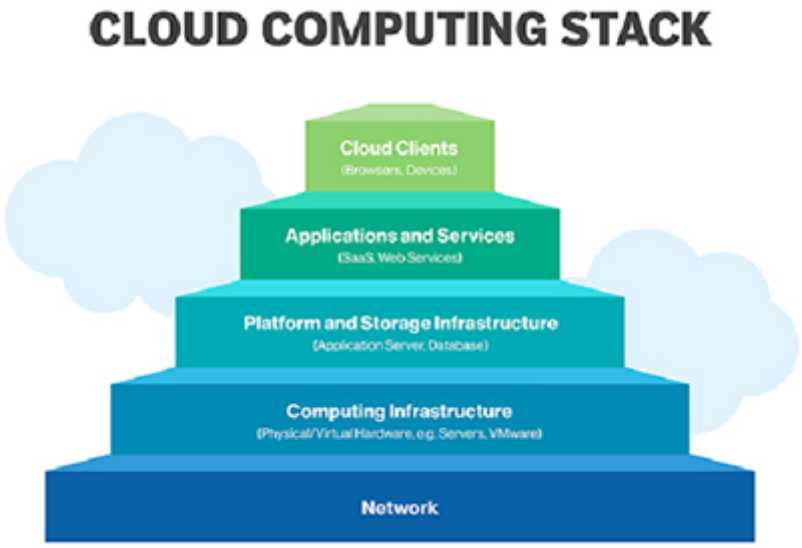


Fig 1. Cloud Computing Stack

3.1 Cloud Infrastructure Components

Cloud infrastructure components refers to the back-end components such as hardware elements within most enterprises. They include multi sockets, multi-core servers, persistent storage, local area network such as switches and routers.
Examples such as AWS and google platform offer services based on multi-tenant and shared servers.

3.2 Smart community services

- Activity notifications: Whenever user receives a message he is notified and keeps users engaged in whatever is happening. This feature is useful during an emergency.
- Chat: Users can chat with other users.
- Mass-mail: Department staff send mass-mail to the users in case of emergency.
- Request submission: Every user can submit their requests to the department with the help of this feature.
- Traditional mail-box: Every user will have a traditional mailbox. For example, if the status has been resolved the updated status goes to their mailbox.

3.3 Connected community services

- Messages: Users are allowed to send messages to one or many users or to the department.
- Privacy: User's profile is safe with this feature. This includes restricting access to the user's profile and other settings to safeguard user data.
- Tracking the submitted requests: In order for the users to know about the status of their requests, they can track the requests.
- Standard Facebook-style friend system and many other features can be built on top.

3.4 System Infrastructure Diagram

The diagrams below describe the detailed infrastructure diagram for a single cluster instance and the high-level infrastructure depicting all the connected clusters. There are four departments in the system: police department, transportation department, animal protection department and client request center with each cluster component communicating with the other components.

Each component has a system admin, department staff, DB storage etc. Each individual object talks to other objects through smart-net. Users can use any of the components described and their service requests are processed by department staff. Each department has it's own features.

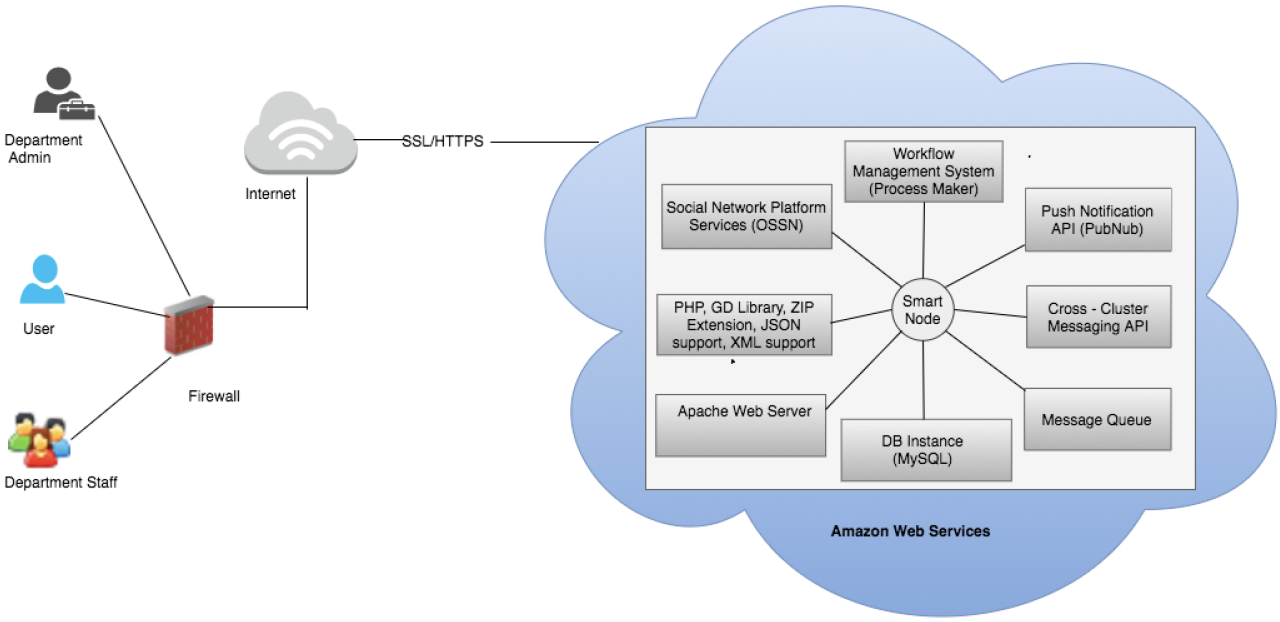


Fig 2. Infrastructure diagram for a single cluster instance

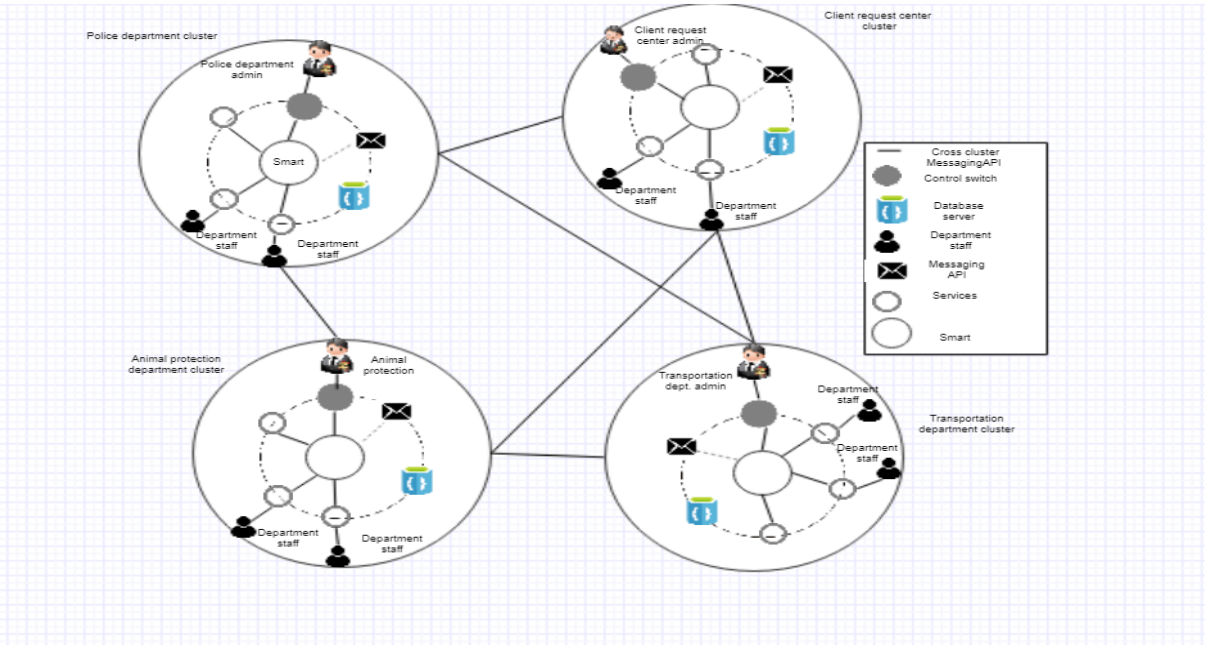


Fig 3. System Infrastructure diagram depicting all the clusters

Chapter 4: Technology Selection

4.1 List of Technologies

- **Open Source Social Network (OSSN)**
  OSSN and its predefined tools are used as the social network development platform for customizing and building the solution.

  Prerequisites:

  - PHP
  - MYSQL
  - APACHE
  - MOD_REWRITE
  - PHP GD Library
  - PHP ZIP Extension
  - JSON Support
  - XML Support

- **ProcessMaker**
  ProcessMaker open-source web based application is used for workflow management and to streamline request processes.

- **ProcessMaker API**
  ProcessMaker API is used for allowing external scripts to remotely access and control Processmaker using REST.

- **MySQL**
  MySQL instance will be used in each cluster for storing the data.

- **PHP**
  PHP programming language is used for customizing the OSSN modules and for creating new modules.

- **AWS**
  Amazon web services for building the cloud infrastructure and application deployment.

- **Apache Web Server**
  To host the website and to serve the user requests.

- **GitHub**

GitHub repository is used for version control.

Chapter 5: System Design

Implementation of the integrated platform, consisting of different departments, transforms the city from just being responsive to providing a smart and connected service environment. Some key smart communication and connectivity features in the system are addressed as follows:

- Automatic routing of incidents to relevant departments using Messaging API and ProcessMaker API.
- Decision making on a service request using ProcessMaker workflow.
- Real-time status tracking of service requests using OSSN component.
- Send notifications within and outside the group using PubNub Push Notifications.
- Define predefined message templates for emergency alerts and policy change notifications.

5.1 System Workflow for Processing User Requests

The figure below demonstrates the flow of a user request from the request acceptance stage to request resolution.

- The user places the request which is received by the citizen request center and the user is notified with an acknowledgement.
- After the request category check, the request is routed to the respective department.
- The request is processed by the department.
- The user is notified about the request status.
- User can also check the status of the request when the request is in process.

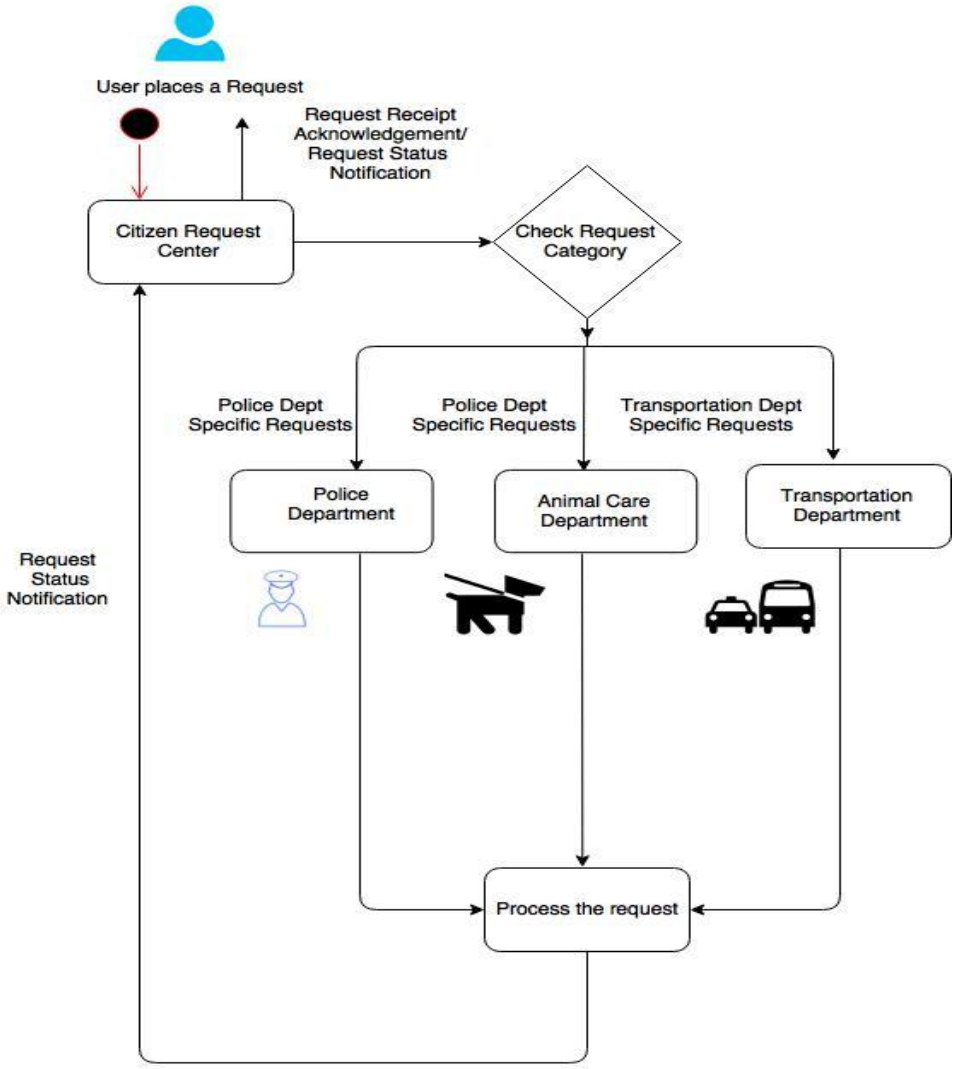


Fig 4. High level request processing workflow diagram

5.3 System Component Diagram

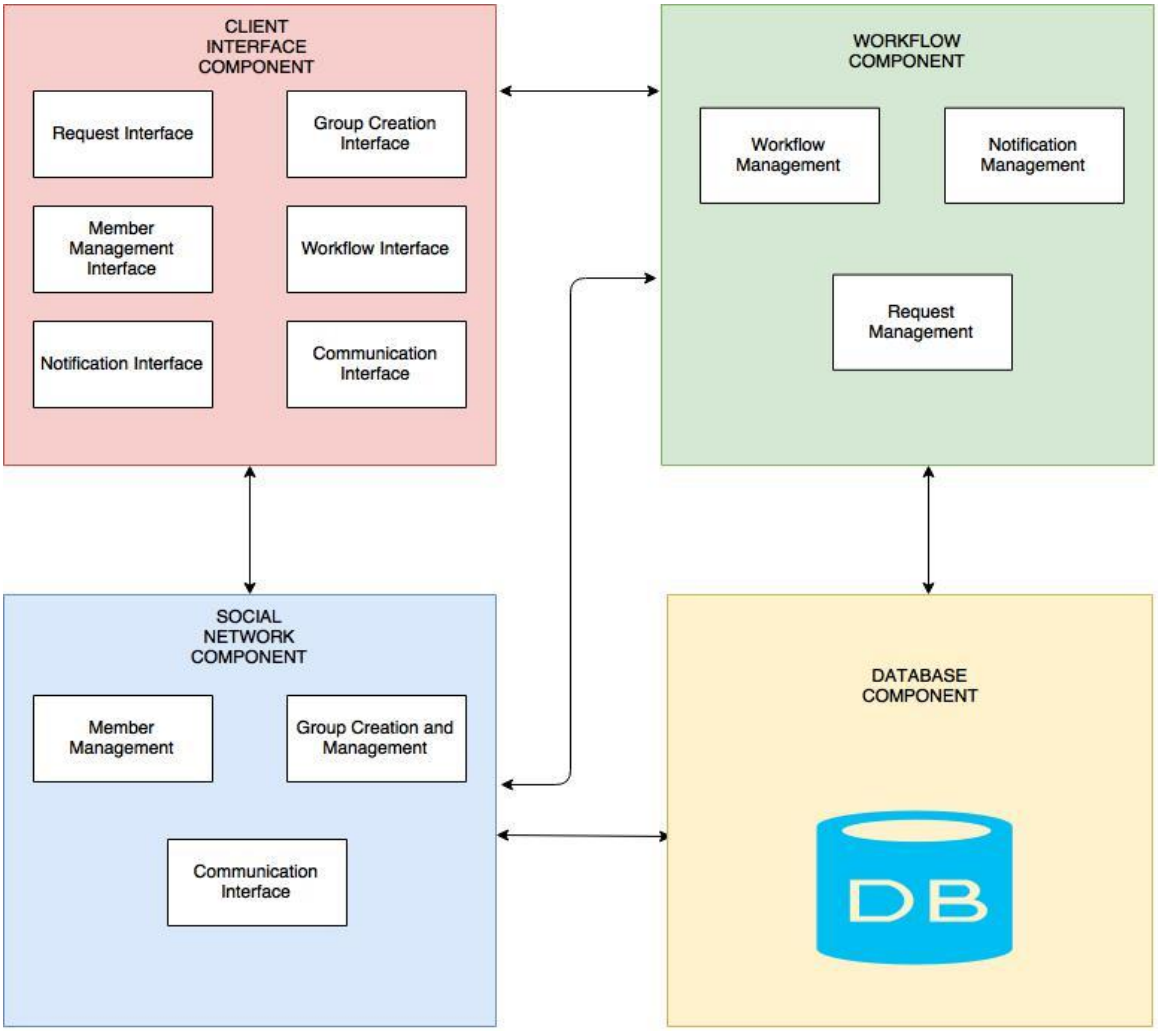


Fig 6. System Component Diagram

5.4 System Quality Attributes

The solution includes various aspects of a smart city ecosystem such as the smart communication, smart operation, smart infrastructure and smart service. Below are few of the quality attributes the system focusses on.

- **Scalability:**
  The system, a SaaS application, is designed to scale out to accommodate the growing number of users, and provide on-demand services and connectivity. The open-source nature of the stack components like OSSN, ProcessMaker, PHP and MySQL also contributes to fast and efficient system scalability.

- **Sustainability:**
  Flexible configuration of the system components and efficient workflow management helps the system evolve quickly, in order to adapt to any changes in the technical environment or business use cases.

- **Security & Privacy:**
  Identity and Access Management and Web Application Firewall services of AWS are leveraged for agile security configuration of the application. This also makes the security and privacy aspects cost effective.

- **Communication:**
  This communication platform is supported by the open source social network component for communication within a cluster and the REST API for communication across the clusters.

Chapter 6: Components

The system supports the following components:

- Community Management
- Workflow Management
- Messaging
- Dashboard
- Database

6.1 Community Management Component

Community management component supports the management of different communities. The functions provided by this component includes creation, update, deletion and viewing of different communities. Community manager can initiate requests for creating a new community/department. The requests are reviewed by the system admin who approves or refuses the request. This workflow applies to other processes like update and deletion of nodes.

Community management component provides the admin with a Graphical User Interface that makes the management of the nodes easier. Basic details about a node is captured through this GUI, which can be used to create nodes from the snapshot of previous configuration of created nodes.

6.1.1 Basic Functions

Function	Description
Create Community	This component allows the admin to create a new node upon request and add it to the cluster. User will be notified upon instance creation through email.
Configure Community	This component enables the admin to configure and update the community.
View Community	This component provides admin with an interface to view details of existing communities.
Delete Community	This component allows admin to terminate and delete a node upon request, and free the resources associated with it. User will be notified upon instance deletion through email.

6.1.1.1 Create Community

This component creates a new community and add it to the cluster. Different communities are connected to each other through messaging component. Community managers who want to create a new community are provided with a GUI where they can enter the specifications. Once the request gets submitted, admin can view it and approve/refuse the request.

The community instance is launched and configured through an automatic process that has been developed as part of the community management component. Upon successful creation, moderator is notified through an email which contains the information to log in to the newly created community instance. The moderator can sign in to add users/staffs and create groups using the social networking component provided in the community instance.

6.1.1.2 Update Community:

The software component allows customizing the configuration of different communities according to the community manager's requirements. Admin is authorized to perform all the configurations.

6.1.1.3 View Community:

The GUI for community management component provides an interface where the admin can view the existing community and the status of it. This would help the admin for monitoring and request approval.

6.1.1.4 Delete Community:

This component facilitates deleting or terminating a community. Community manager places a request for deleting a community. The admin executes the process of instance termination through the community management GUI. This process is automated with the help of Amazon API for EC2. All the allocated resources and data will be removed on instance termination

6.1.2 Users and Use cases

There are three kinds of users for the community management component.

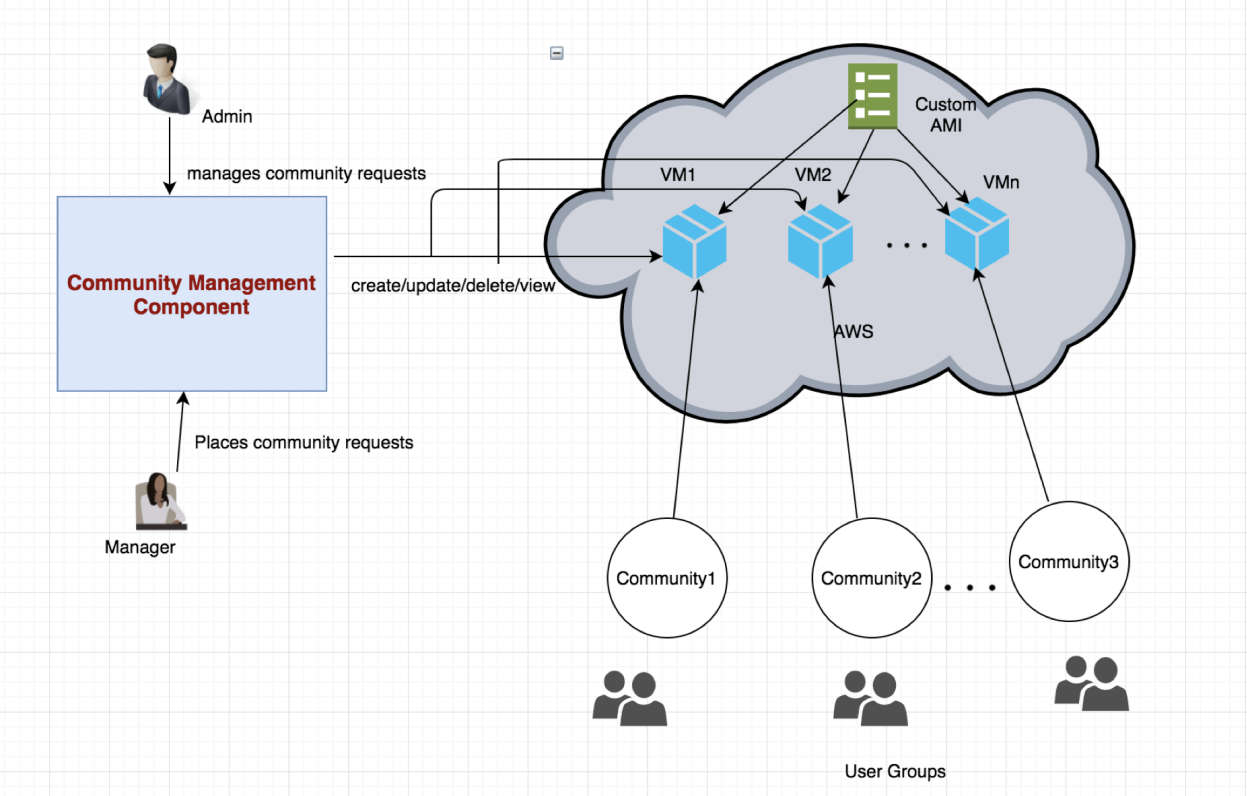
User	**Use Cases**
Administrator	Admin performs create, delete, view and configure functions on receiving the requests from the community manager.
Community Manager	Community manager is responsible for placing requests to create or delete a community instance.
Moderator	Moderator will be notified upon success creation of node. Moderator performs the tasks like adding staffs/users and creating groups after the launch of a new community.

6.1.3 Component Design

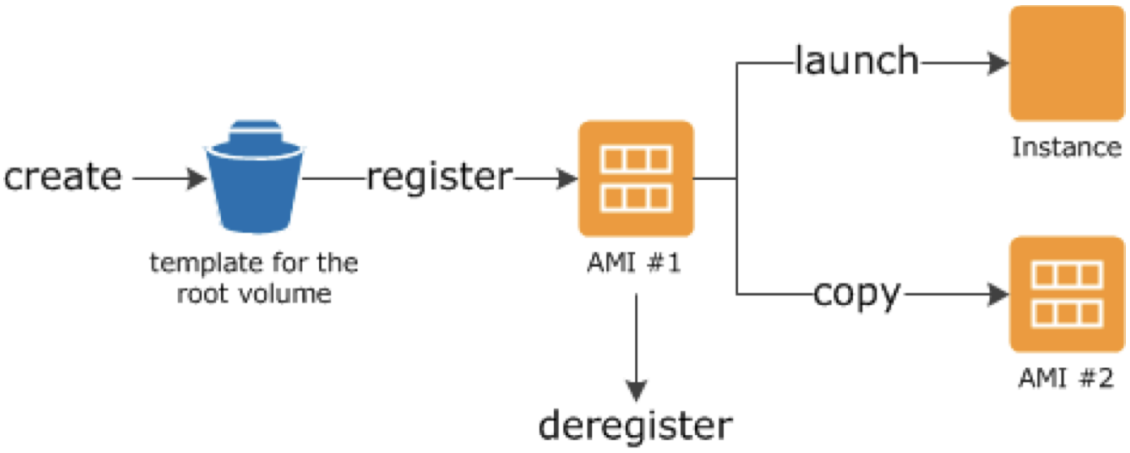
The community management component automatically creates one virtual machine per community in the chosen cloud infrastructure, AWS. This provides on-demand services with significant scalability and resiliency.

6.1.3.1 Component Function

Each new community is automatically launched in the cloud environment(AWS) using the snapshots of basic configuration of previously created nodes. Necessary custom configurations are provided during the launch to suit the needs of different communities. Amazon Machine Images(AMI) are used to automate the community creation. AMIs acts as a template containing all necessary information to launch a community.



A custom AMI has been created for the community management component, which contains different software components like social networking component, database component, web server and messaging component. This custom AMI is used by the community management software to automate the virtual machine (community) creation process on-demand. A new instance is created for each community.



6.1.3.2 API and Algorithms

Amazon EC2 API is used to implement the automatic instance management for this component. MySQL is used to store details such as user login, user details, community requests and community instance details. The Graphical User Interface for community management is developed using PHP, ProcessMaker and Node.js. AWS SDK for JavaScript enables the software component to connect to the AWS services programmatically to create and terminate an instance. Node.js provides an efficient cross platform runtime.

6.1.3.2.1 Algorithm to implement automatic creation of instance using EC2 API

```
var AWS = require('aws-sdk');
AWS.config.update({
    accessKeyId: "AKIAIBBLJ2GN7MXLBMQ",
    secretAccessKey: "****************",
    "region": "us-west-2"
});

var ec2 = new AWS.EC2({apiVersion: '2016-11-15'});

var commands = [
    '#!/usr/bin/env bash',
    'mkdir /home/ubuntu/test',
    'sudo chmod 777 /var/www/html/ossn/configurations',
    'sudo rm /var/www/html/ossn/configurations/ossn.config.site.php'
];

var params = {
   ImageId: 'ami-84ec3ffc',
   InstanceType: 't2.micro',
   MinCount: 1,
   MaxCount: 1,
   KeyName: 'ossnkey',
   SecurityGroupIds: ['sg-951814e8'],
   UserData: new Buffer(commands.join("\n")).toString('base64')
   };
```

```
// Create the instance
var insId;
var dns;
ec2.runInstances(params, function(err, data) {
   if (err) {
      console.log("Could not create instance", err);
      return;
   }
   else
   {
   insId = data.Instances[0].InstanceId;
   //console.log("Created instance", insId);
   //wait for instance creation
   var sleep = require('sleep');
   sleep.sleep(10);
   ec2.describeInstances({}, function(err, data) {
        if(err) {
            console.error(err.toString());
        } else {
            var currentTime = new Date();
            //console.log(currentTime.toString());
            for(var r=0,rlen=data.Reservations.length; r<rlen; r++) {
                var reservation = data.Reservations[r];
                for(var i=0,ilen=reservation.Instances.length; i<ilen; ++i) {
                    var instance = reservation.Instances[i];

                    var name = '';
                    for(var t=0,tlen=instance.Tags.length; t<tlen; ++t) {
                        if(instance.Tags[t].Key === 'Name') {
                            name = instance.Tags[t].Value;
                        }
                    }
                    if(instance.InstanceId ==insId){
                    //console.log("IP: " + instance.PublicIpAddress);
                    console.log("DNS: " + instance.PublicDnsName);
                   // console.log("STATE: " + instance.State.Name);
                    //wait before connect
                    var sleep = require('sleep');
                    sleep.sleep(60);
                    //console.log("STATE After Wait: " + instance.State.Name);
                    const exec = require('child_process').exec;
                    var yourscript = exec('sh awscon.sh ' + instance.PublicDnsName,
                        (error, stdout, stderr) => {
                        //console.log(`${stdout}`);
                        //console.log(`${stderr}`);
                        if (error !== null) {
                        //console.log(`exec error: ${error}`);
                }
        });
```

6.1.3.2.2 Algorithm to implement automatic deletion of instance using EC2 API

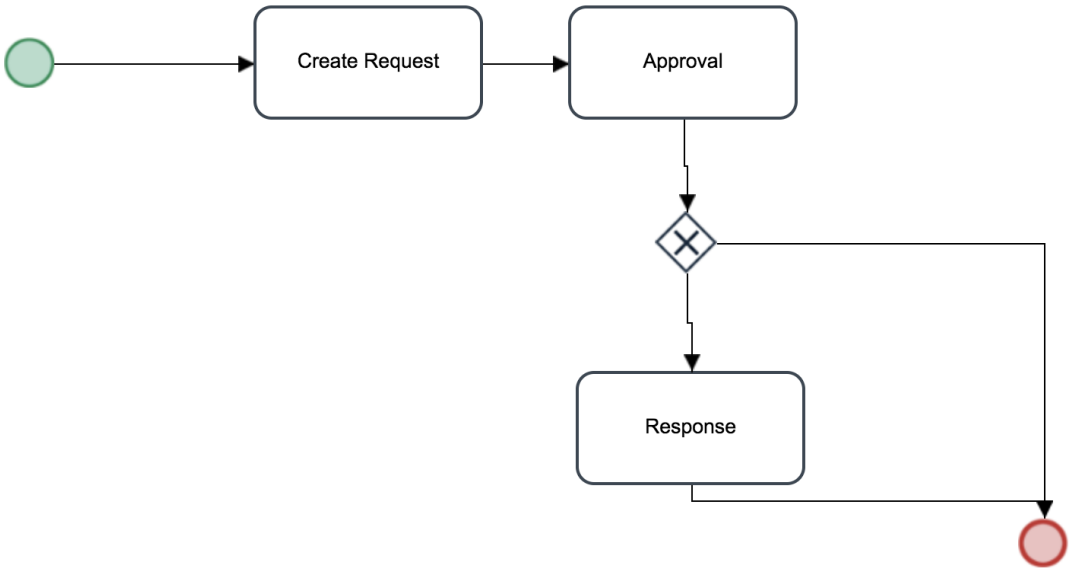
```
var AWS = require('aws-sdk');
AWS.config.update({
    accessKeyId: "AKIAIZWS2QEM3QYYSA",
    secretAccessKey: "******",
    "region": "us-west-2"
});

var ec2 = new AWS.EC2({apiVersion: '2016-11-15'});
var ins;

    ec2.describeInstances({}, function(err, data) {
        if(err) {
            console.error(err.toString());
        } else {
            var currentTime = new Date();
            for(var r=0,rlen=data.Reservations.length; r<rlen; r++) {
                var reservation = data.Reservations[r];
                for(var i=0,ilen=reservation.Instances.length; i<ilen; ++i) {
                    var instance = reservation.Instances[i];

                    var name = '';
                    for(var t=0,tlen=instance.Tags.length; t<tlen; ++t) {
                        if(instance.Tags[t].Key === 'Name') {
                            name = instance.Tags[t].Value;
                        }
                    }
                    console.log('\t'+name+'\t'+instance.InstanceId+'\t'+instance.PublicIp

                }

            }
        }

    });
```

6.1.3.2.2 Algorithm to implement view community details function

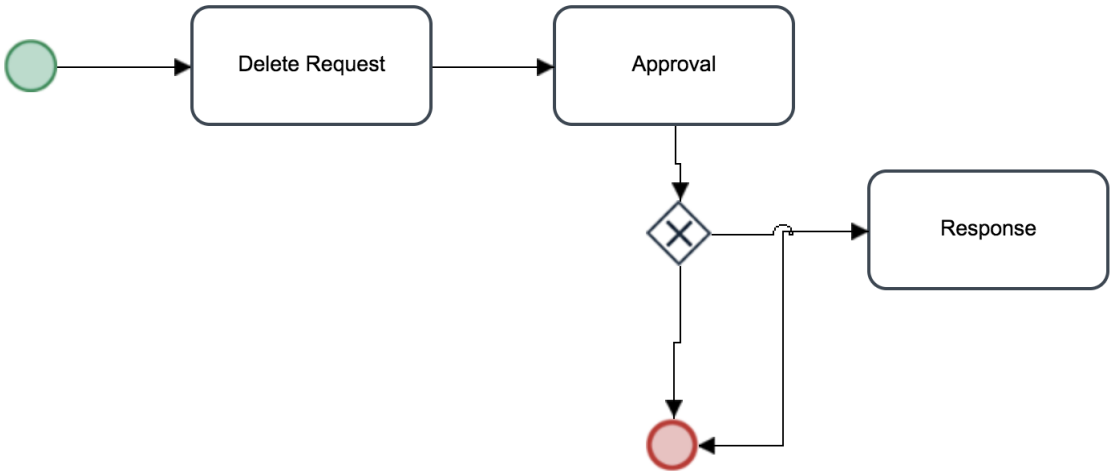
```
var AWS = require('aws-sdk');
AWS.config.update({
    accessKeyId: "AKIAIBBLJ2GN7MBMTQ",
    secretAccessKey: "************",
    "region": "us-west-2"
});

var ec2 = new AWS.EC2({apiVersion: '2016-11-15'});
const args = process.argv;
instanceId =args[2]
console.log (instanceId)
ec2.terminateInstances({ InstanceIds: [instanceId] }, function(err, data) {
        if(err) {
            console.error(err.toString());
        } else {
           for(var i in data.TerminatingInstances) {
                var instance = data.TerminatingInstances[i];
                console.log('TERMINATING:\t' + instance.InstanceId);
            }
        }
    });
```

6.1.4 Component Workflow

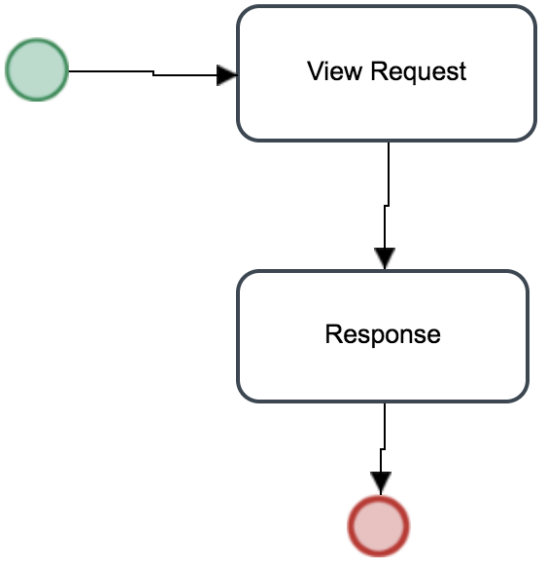
6.1.4.1 Create and configure community workflow



6.1.4.2 Delete community workflow



6.1.4.3 View community workflow

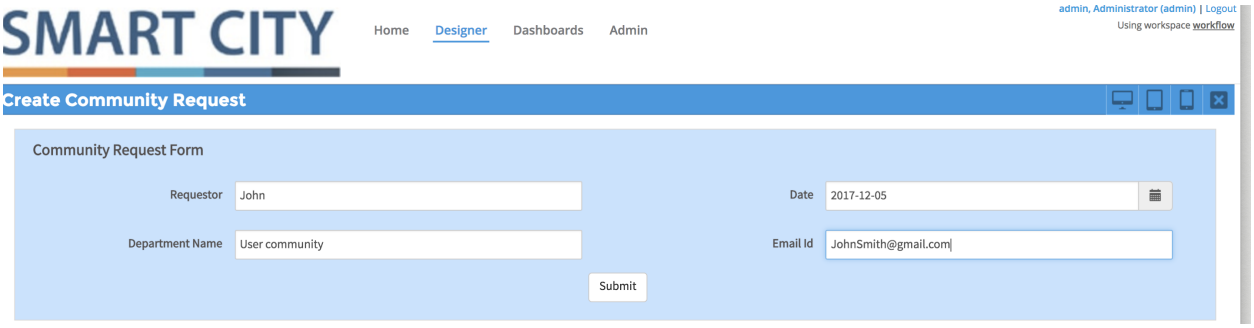


6.1.5 Component GUI

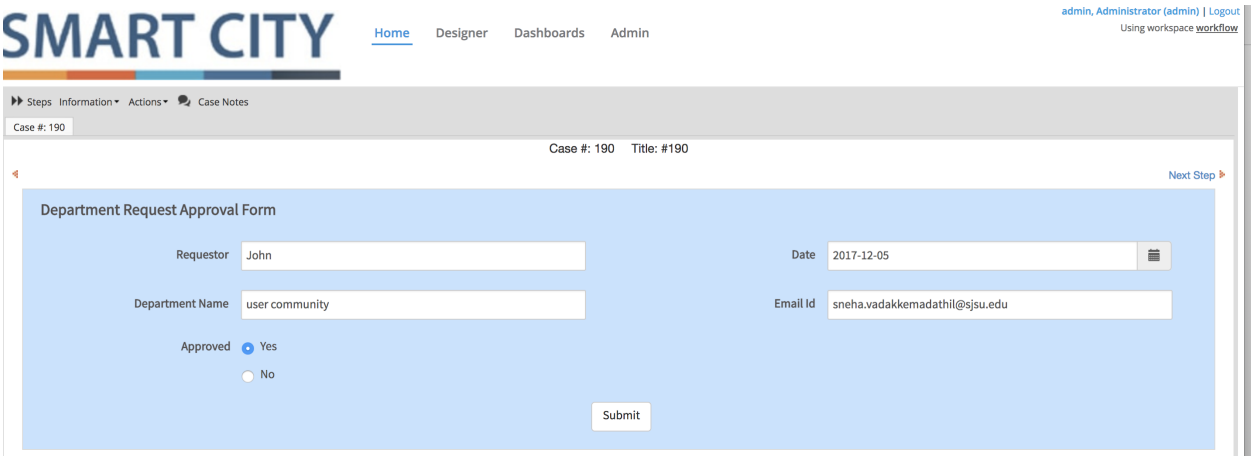
Community managers can log in to use the community management GUI to create requests by filling up the basic details. These requests are then routed to the administrator. Component GUI allows the administrator to log in and see the pending requests and take appropriate actions. This GUI also allows admins to view various departments in the cluster.

6.1.5.1 Community management login

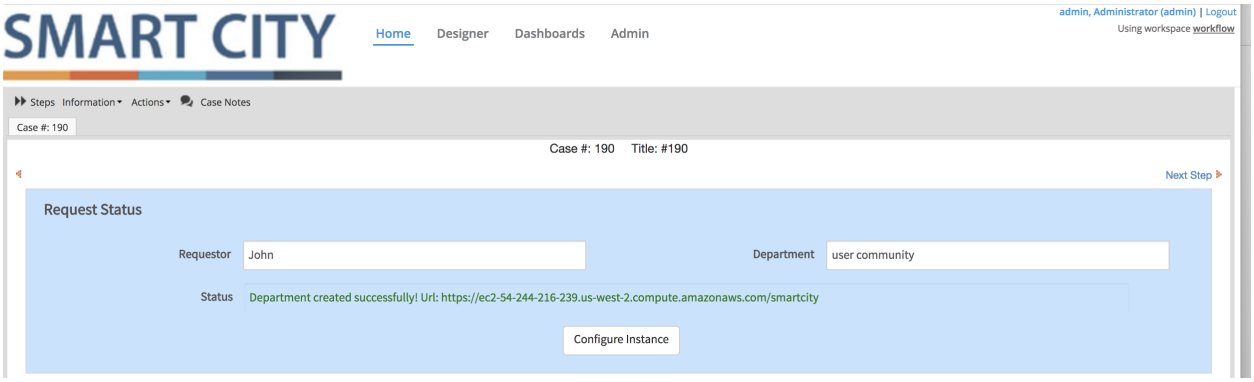
6.1.5.2 Create community request



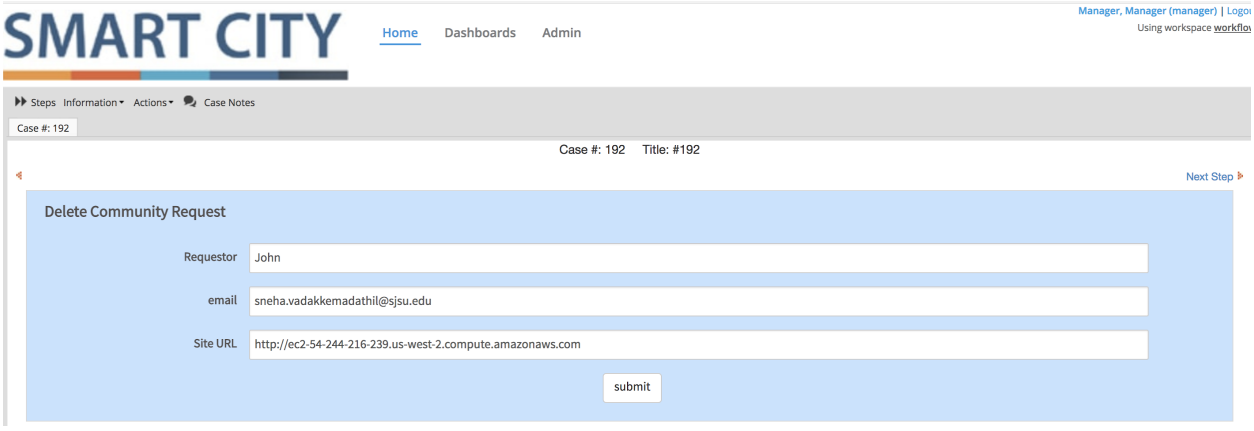
6.1.5.3 Community request approval



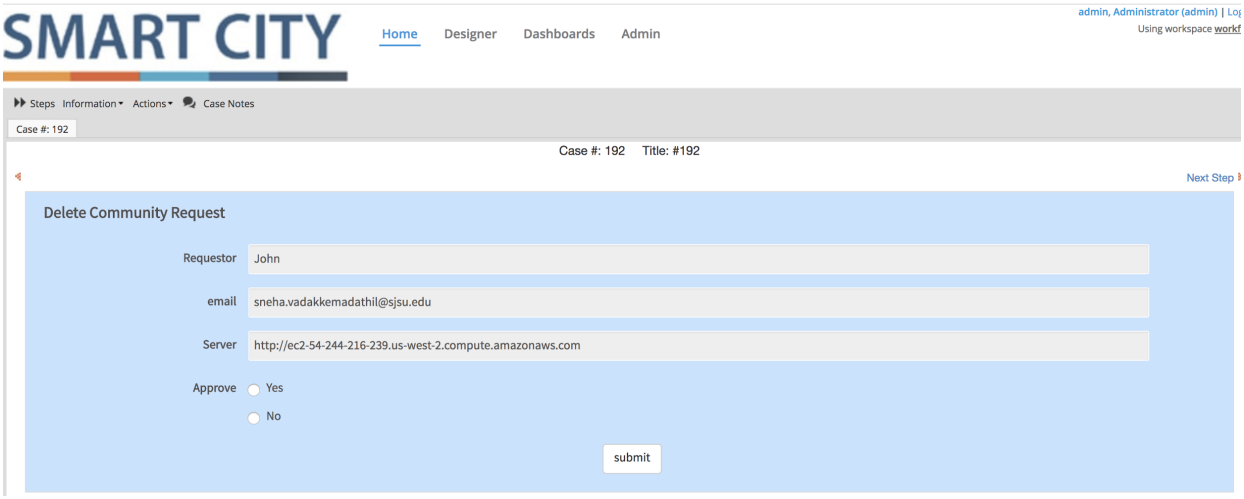
6.1.5.4 Configure community instance



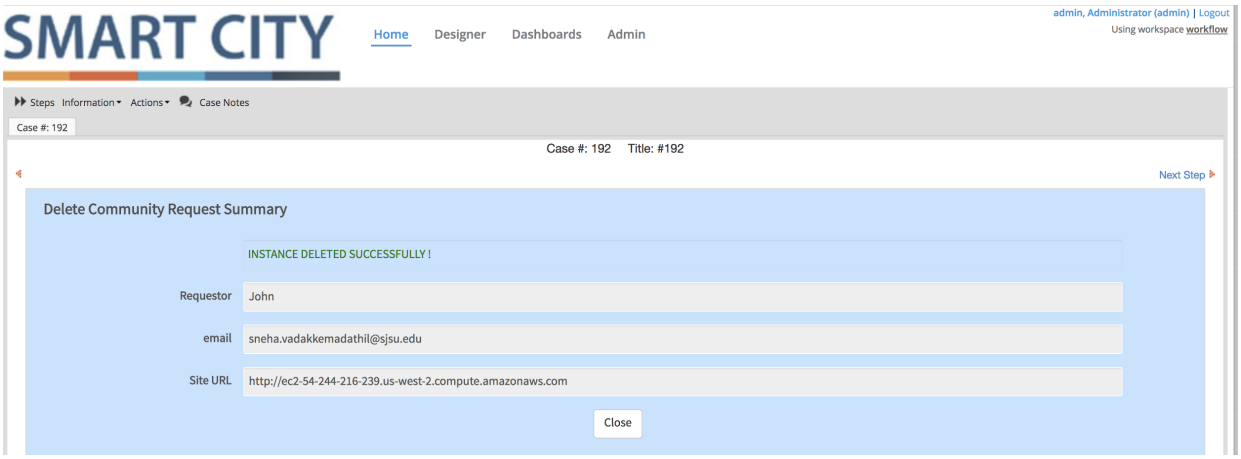
6.1.5.5 Delete community request



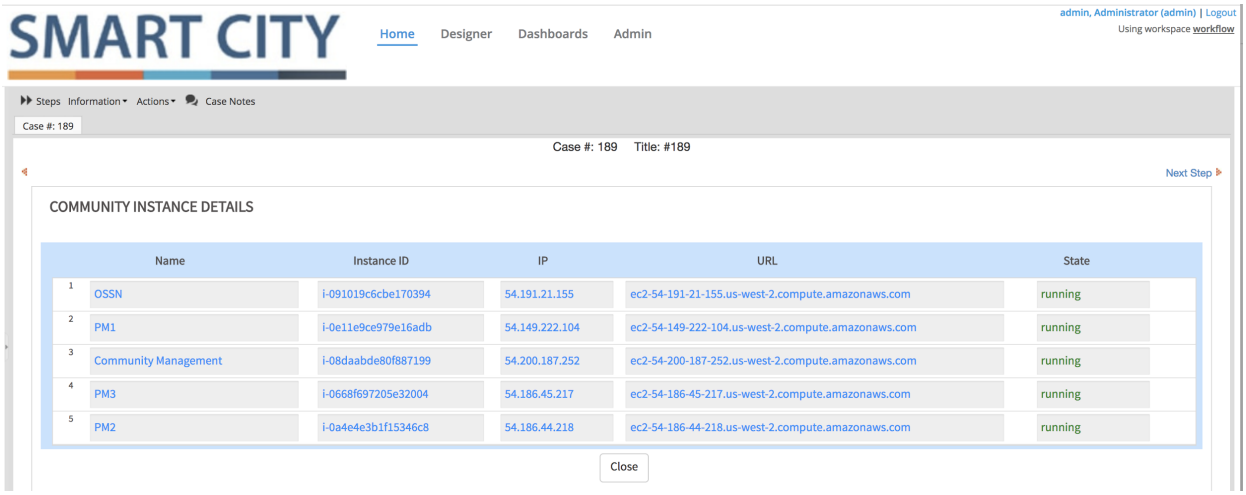
6.1.5.6 Delete community approval



6.1.5.7 Delete request summary



6.1.5.8 View community details



6.2 Workflow Management Component

Smart City is a service organization with citizens as its customers and it takes care of providing services to its citizens. This is a smart, effective and efficient way of managing the communities and departments.

In order to improve the case management, operations, workflows, cross-department collaborations, regulatory enforcements, ProcessMaker open Source BPM and workflow solution is used.

ProcessMaker workflow software features an extensive toolbox which provides the ability to easily create digital forms and map out fully functioning workflows. The software is completely web based and accessed via any web browser, making it simple to manage and coordinate workflows throughout an entire organization - including user groups and departments. Instead of running around to get approvals and request routing, processmaker workflow management is used in the system.

6.2.1 Benefits of using ProcessMaker

- It is an open source
- Active community support of users
- Flexible workflow management
- Easier tracking
- Streamlines process
- Drag and drop interface
- Automate notifications
- Easy and fast development and deployment.

- Dashboard feature

6.2.2 Basic Functions

The workflow management service is used for effective communication and request routing between the departments and staff. Following is the list of functions provided by the workflow component.

Application Area	Function	Description
Department Creation Request	1. Request Routing	On the request of Application Manager for a new department, the request will be routed to the Admin for approval.
	2. User/Role management	Admin can create the users and assign them to specific departments.
	3. Request approval/disapproval	The admin will be able to approve or disapprove the request.
	4. Information Validation	The information provided by the manager will be validated by a trigger.

	5. Check for an existing request	A check will be performed before every request submission for similar existing application to avoid request redundancy.
	6. Notification	Notify users when they have been assigned a task or case. Notify the Manager about the new Department created with the link to URL.
	7. Case tracking	Users will be able to manage the tasks assigned to them
Raise alerts and requests	1. Parse the reported issue	Trigger the script to parse the issue posted by the citizens.
	2. Classify the issue	Classify the issue category based on keywords.
	3. Route to respective department	Route the request to specific department.

	4. Notification	Notify all the staff belonging to a group about the new request reported
	5. Database update	Update the status of the request in the backend.
	6. Request Rerouting	Request can be rerouted to other departments.
Request Handling within a Department	1. Check for staff availability	Query the database for the staff available using a trigger.
	2. Request routing	Route the request to the available staff.
	3. Make decision	Staff can mark the request complete, pending or rejected based on the request status.
	4. Database update	Update the status of the request in the database.
	5. Notification	Notify the end user about the status.

6.2.3 Users and Use-Cases

User	**Use Cases**
Application Admin	· Creation of users/groups. · Approve Department Creation Requests · Design the process · Monitor the dashboard · Send and receive notifications
Application Manager	· Places Department Creation requests on behalf of Moderators. · Send and receive notifications
Department Moderator	· Receive notifications when new departments created.
Department Staff	· Manage user incidents · Route user requests · Respond to users
Citizens/Users	· Report incidents · Receive response · Check request status

6.2.4 Component Design

The steps below explain how a process is created and designed in ProcessMaker. The intuitive drag and drop interface makes the process easier.

6.2.4.1 Project creation

1. Install ProcessMaker.
2. Create a new BPMN Project.
3. Assign a name provide a description for the project.

6.2.4.2 User/Group/Department Creation

1. Create the users/groups/departments as needed.

6.2.4.3 Workflow Design

1. Drag the elements from the shapes and tasks toolbar into the process map.
2. Name the tasks.
3. Add the exclusive gateway and connect the tasks.
4. Create the Dynaforms which acts as an interface for user interaction. It is used for both entering the data and for user interaction.
5. Create the variables and link it to the objects on the Dynaform.
6. Add the conditions to the gateways.
7. Set the assignment rules.

6.2.5 Algorithms

6.2.5.1 Parse User Input

As per the current approach, the user's posts from OSSN community will be parsed and scanned and in case of any concern or issues, the respective department will be alerted.

Step 1: Define the list of keywords for all three categories.
Step 2: Get User Input
Step 3: Parse the user input for key words.
Step 4: If match found, identify the request category.
Step 5: Return the Department name where the request should be routed to.

6.2.5.2 Request Re-Routing

If a request assigned to a department, requires another department's assistance or the request is wrongly assigned to a department, the request can be routed to the concerned department.

Step 1: Set the reroute department id.

Step 2: Check for the request status.
Step 3: If request is pending and reroute indicator is set, then reroute the request.
Step 4: If request is Completed or Rejected, notify the user about the status.

6.2.5.2 Request count balancer

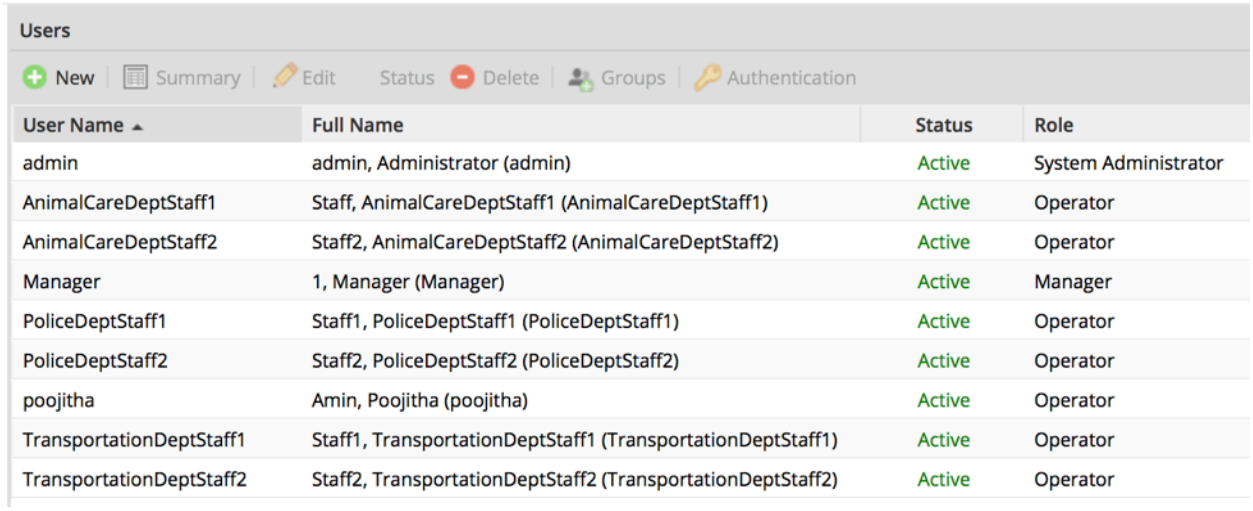
For efficient utilization of staffs, the cases will be allocated based on the existing cases in the Staff's bucket. The Staff with the least number of cases, will be assigned the next request.

Step 1: Get the Department Name.
Step 2: Check the Case count of all the staffs under that department.
Step 3: Get the staff name with least amount of cases.
Step 4: Assign the case to that Staff.
Step 5: Increment the case count by one in the database.
Step 6: Once the case is processed, decrease the case count by one for that staff.

6.2.6 GUI

6.2.6.1 Login Page

6.2.6.2 List of Users

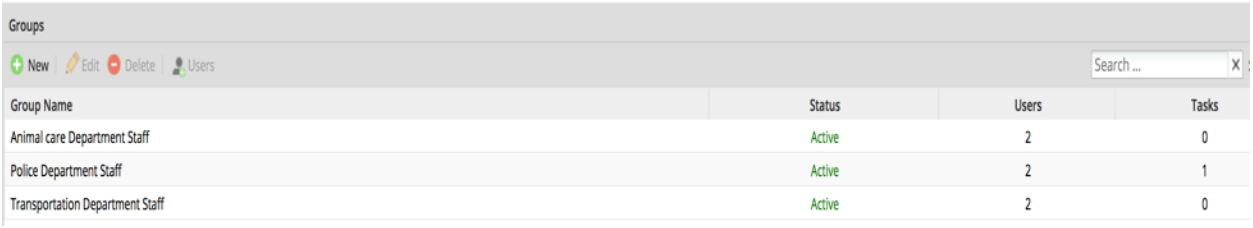


Users

New | Summary | Edit | Status | Delete | Groups | Authentication

User Name	Full Name	Status	Role
admin	admin, Administrator (admin)	Active	System Administrator
AnimalCareDeptStaff1	Staff, AnimalCareDeptStaff1 (AnimalCareDeptStaff1)	Active	Operator
AnimalCareDeptStaff2	Staff2, AnimalCareDeptStaff2 (AnimalCareDeptStaff2)	Active	Operator
Manager	1, Manager (Manager)	Active	Manager
PoliceDeptStaff1	Staff1, PoliceDeptStaff1 (PoliceDeptStaff1)	Active	Operator
PoliceDeptStaff2	Staff2, PoliceDeptStaff2 (PoliceDeptStaff2)	Active	Operator
poojitha	Amin, Poojitha (poojitha)	Active	Operator
TransportationDeptStaff1	Staff1, TransportationDeptStaff1 (TransportationDeptStaff1)	Active	Operator
TransportationDeptStaff2	Staff2, TransportationDeptStaff2 (TransportationDeptStaff2)	Active	Operator

6.2.6.3 List of Departments /Groups



Groups

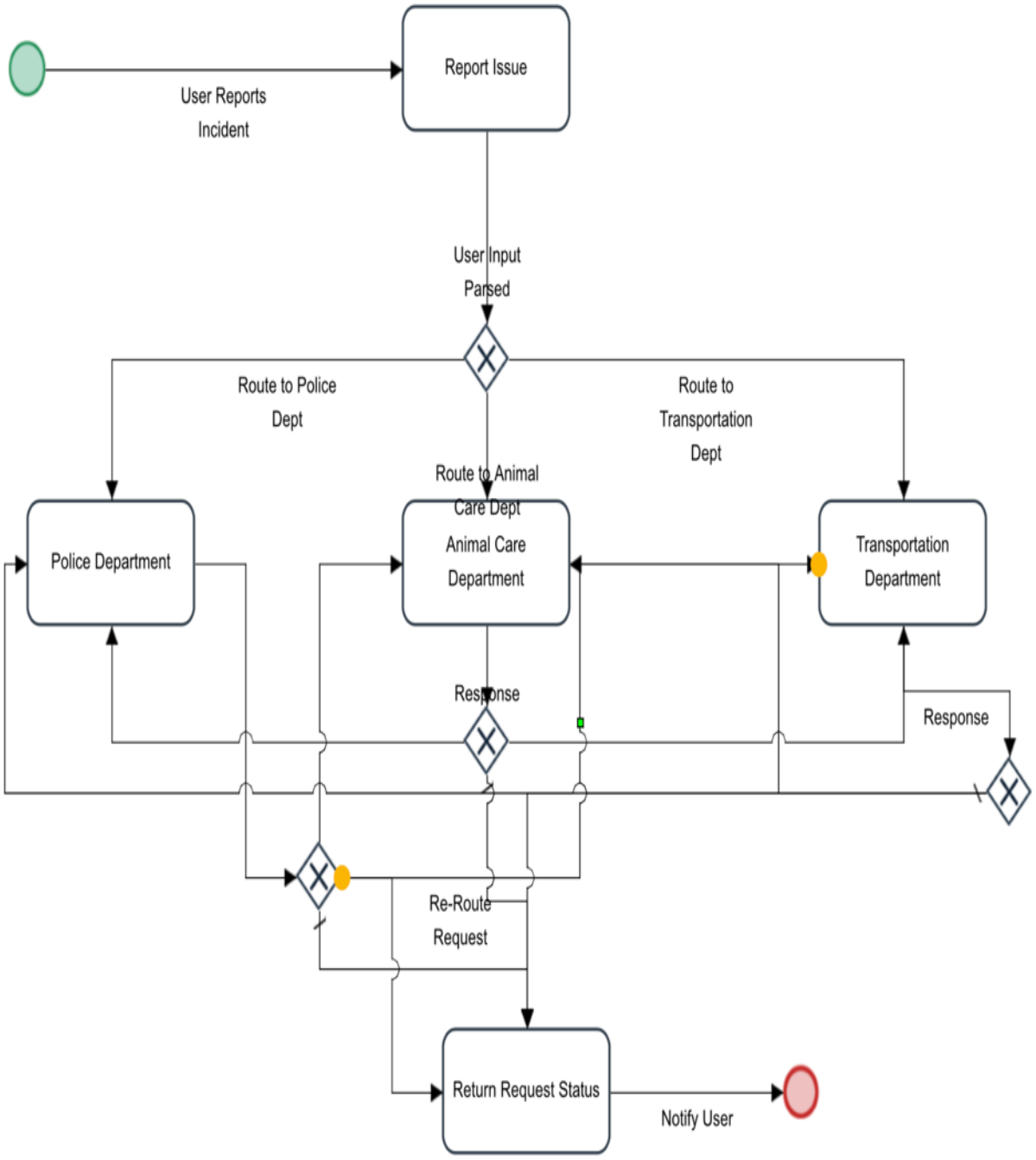
New | Edit | Delete | Users

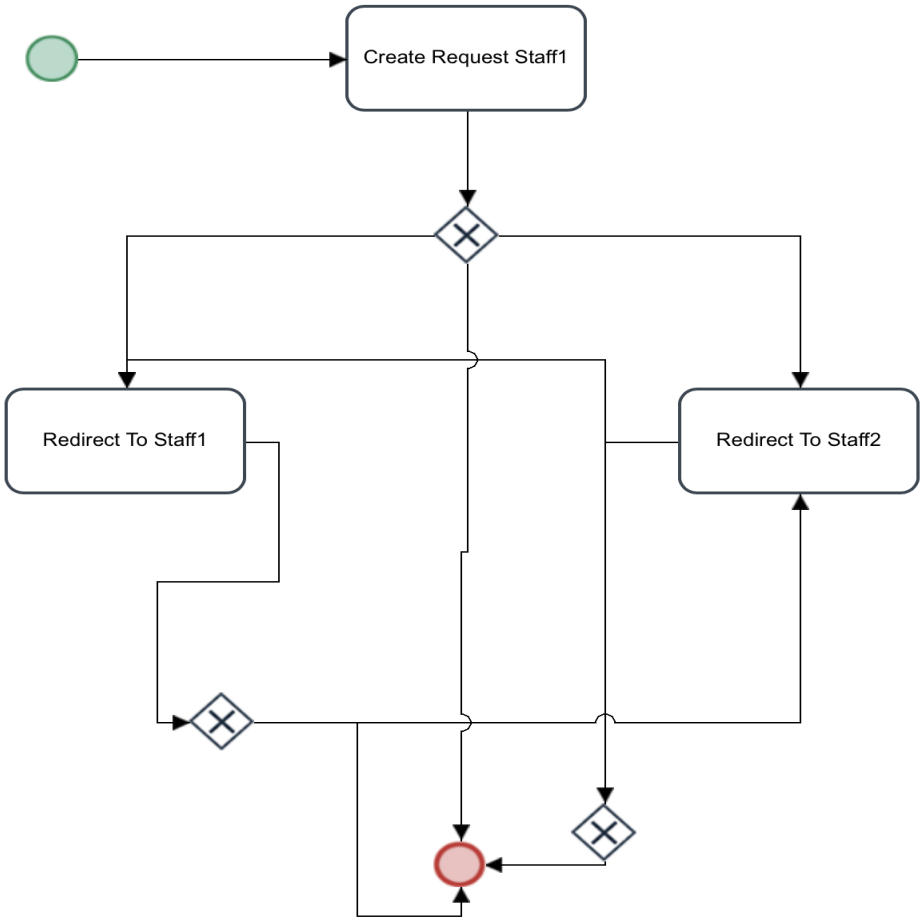
Search ... X

Group Name	Status	Users	Tasks
Animal care Department Staff	Active	2	0
Police Department Staff	Active	2	1
Transportation Department Staff	Active	2	0

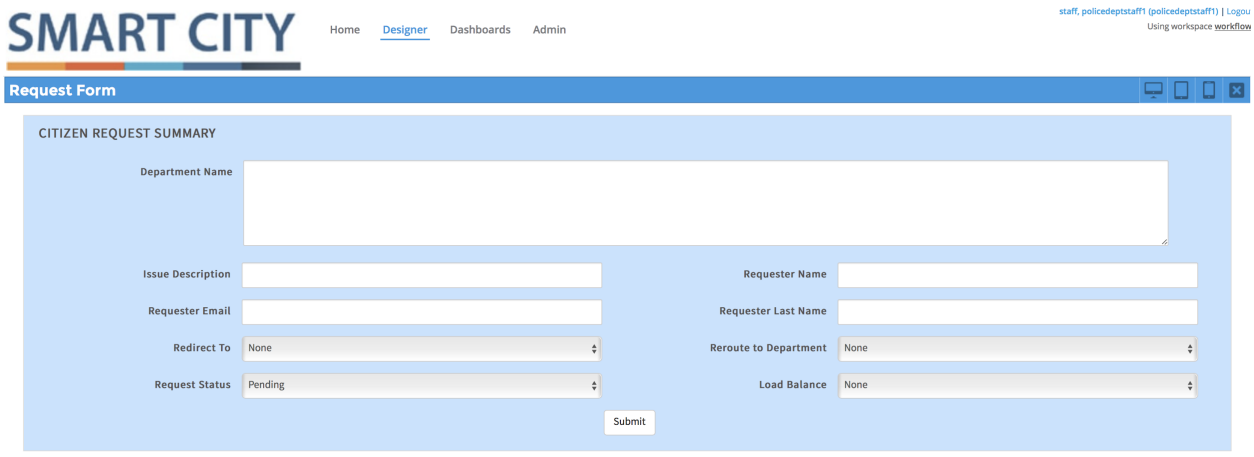
6.2.6.4 Incident Reporting Module

6.2.6.4.1 Workflow

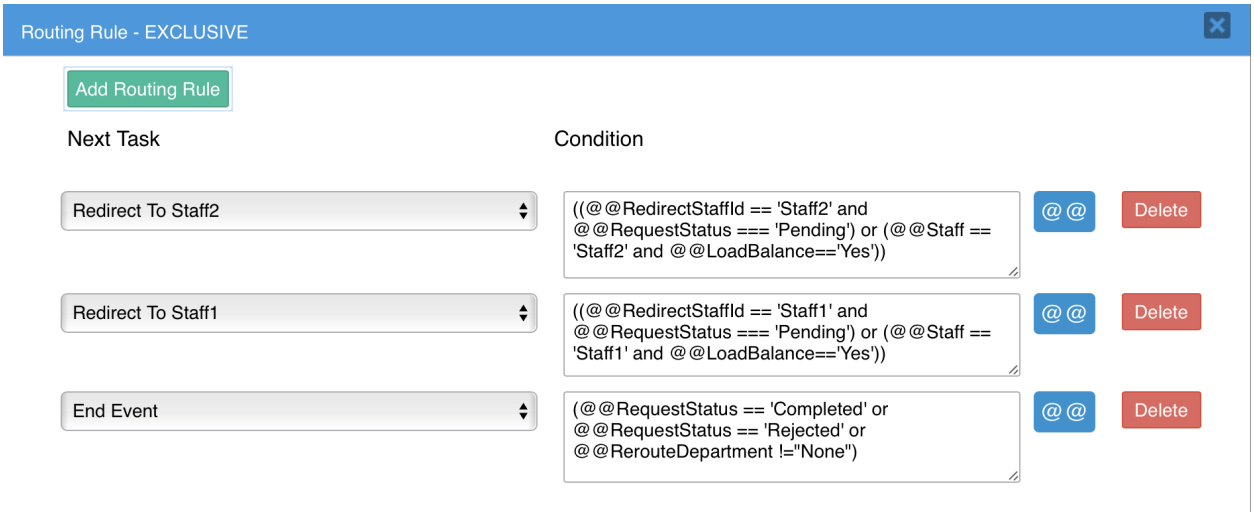




6.2.6.4.2 Dynaform



6.2.6.4.3 Exclusive Route Conditions



6.2.6.4.4 Increment Case Count Trigger

```
$db = "9873300425a23b9d57f5d54036452481";
$StaffName = @@RedirectStaffId;
$StaffName2 = @@Staff;
$DepName = @@DepartmentName;
$query ="Update Staff set Case_Count=Case_Count+1 where Staff_Name=
'$StaffName'";
executeQuery($query, $db);
$query2 ="Update Staff set Case_Count=Case_Count+1 where Staff_Name=
'$StaffName2'";
executeQuery($query2, $db);
```

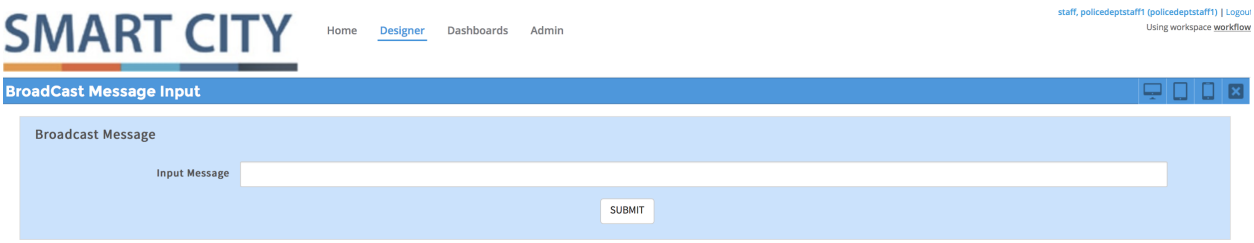
6.2.6.4.5 Get Least Loaded Staff Trigger

```
@@Staff = 'Police';
$db = "9873300425a23b9d57f5d54036452481";
$DepName = @@DepartmentName;
$query ="Select Staff_Name from Staff order by Case_Count Asc limit 1";
$res=executeQuery($query, $db);
@@Staff = $res[1]['Staff_Name'];
@@RedirectStaffId == $res[1]['Staff_Name'];
```

6.2.6.5 Broadcast Messaging Module

6.2.6.5.1 Workflow

6.2.6.5.2 Dynaform



6.2.6.5.3 Broadcast Trigger

```
$c = mysql_connect("ec2-54-191-21-155.us-west-2.compute.amazonaws.com",
"root", "abcd", 3306);
mysql_select_db("ossndb");
$Message = @@Message;
$result = mysql_query("insert into
ossn_messages(message_from,message_to,message,viewed,time)
select t1.message_from, t2.message_to, t1.message,t1.viewed, t1.time from
(select 1 as message_from ,'$Message' as message ,0 as viewed ,1512370573
as time) t1
CROSS JOIN (select guid as message_to from ossn_users where guid<>1) t2;");
$row = mysql_fetch_assoc($result);
echo htmlentities($row['_message']);
```

6.3 Messaging Component

6.3.1 Uses Of Messaging Component

The messaging component is an important part of a Social Networking Community website. It delivers the messages between community members and across different communities too. Here is a general overview of this component. When messages are sent using the front-end UI by any user or admin within a community, it is sent as an HTTP/HTTPS request to the server, like Nginx. The messages are stored in the database at the backend based on the schema of the database. The database is necessarily in Cloud, Amazon RDS. Anything from friend requests, to user login/logout, password change to Creating new user requires a proper DB for storing all user and website related data. Hence, the storage pushed to Cloud will be more useful here. All issues can be tracked with issue tracker. The database is queried to retrieve user requested data and is then sent back in valid viewable format to the front end again which the user finally sees. All this should happen keeping very good scalability and impeccable infrastructure in mind. Hence, Components are individually designed and integrated together to bring a great infrastructure. All these components are encapsulated from user and all that the user sees is the integrated environment that dynamically scales up and down according to the load and needs.

6.3.2 Basic Functions

Functions	Description
Send Message	Messaging Sends messages to Groups(Fire Dept, Animal Care Dept and Police Dept)
Receive Message	Receives messages from Groups
Publish Group channel	Publishes the group messages to the group channel using the redis pub/sub model
Subscribe Group channel	Subscribes the group messages received from redis using the subscriber module in redis.

6.3.3 Users And Use Cases

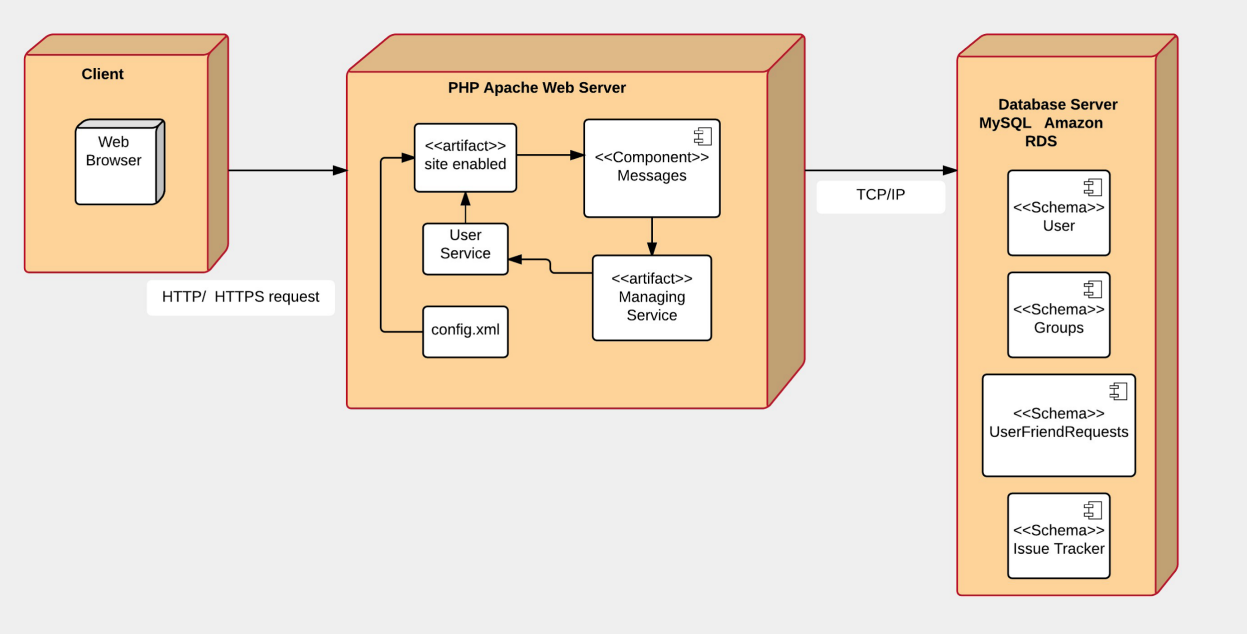
These are the users and use cases involved in messaging component. The use case flow is given below for each of the scenarios as follows:

Users	Use Cases
Department Users	1. User logins in 2. User composes message to police department 3. User clicks send message. 4. User logs out.
Police Department	1. Police Department admin logs in. 2. Admin issues or posts warning message to users within community. 3. Admin logs out.
Fire department	1. Fire Department admin logs in. 2. Admin issues or posts warning message to users within community. 3. Admin logs out.

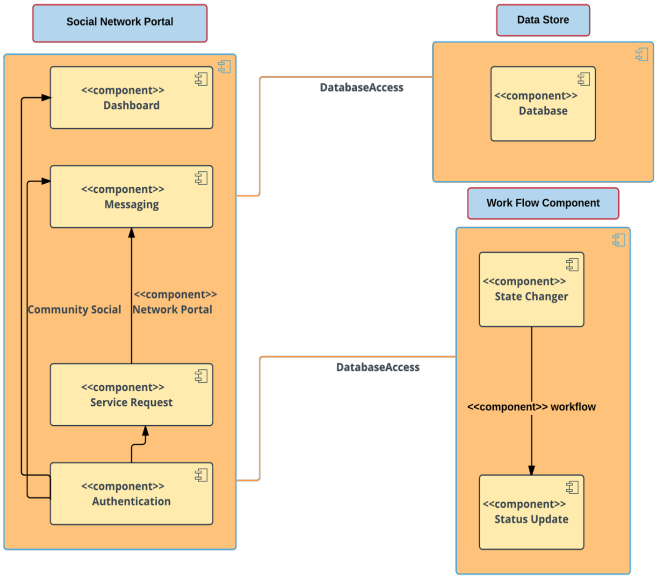
6.3.4 Component Design

The diagram below shows how the messaging component interacts with the rest of the artifacts. The client requests through a web browser and the web server responds to the request. The web server using TCP/IP protocol accesses the Database to query the backend DB that contains the database schema of user information, group information related to the users,

friends of users and issue tracks of records that contains issue lists. By querying appropriately according to the sql commands fired, the information is retrieved and sent back to the user.



COMPONENT DIAGRAM FOR MESSAGING



6.3.4.1 Publishing Method for the channel

```
 */
class OssnWall extends OssnObject {

        /**
         * Publish this wall post to other OSSN isntances using RDIS Pub/Sub
         *
         * @params $channel : Channel ID for posting to other REDIS subscribers
         *         $post: Post text
         *         $friends: Friend guids
         *         $location: Post location
         *         $access: (OSSN_PUBLIC, OSSN_PRIVATE, OSSN_FRIENDS)
         * @param string $post
         *
         * @return bool;
         */

        public function Publish($channel, $group, $post, $friends = '', $location = '', $access = '') {
            $redis = new Redis();
            $redis->pconnect('localhost',6378);
            $redis->publish($channel, 'GROUP='.$group.';POST='.$post.';FRIENDS='.$friends.';LOCATION='.$location); // send message to channel 1.

            $redis->close();

        }


        /**
         * Post on wall
         *
         * @params $post: Post text
         *         $friends: Friend guids
         *         $location: Post location
         *         $access: (OSSN_PUBLIC, OSSN_PRIVATE, OSSN_FRIENDS)
         * @param string $post
         *
         * @return bool;
         */
```

Problems Tasks Console Servers Search

6.3.4.2 Subscribing Algorithm For the Channel

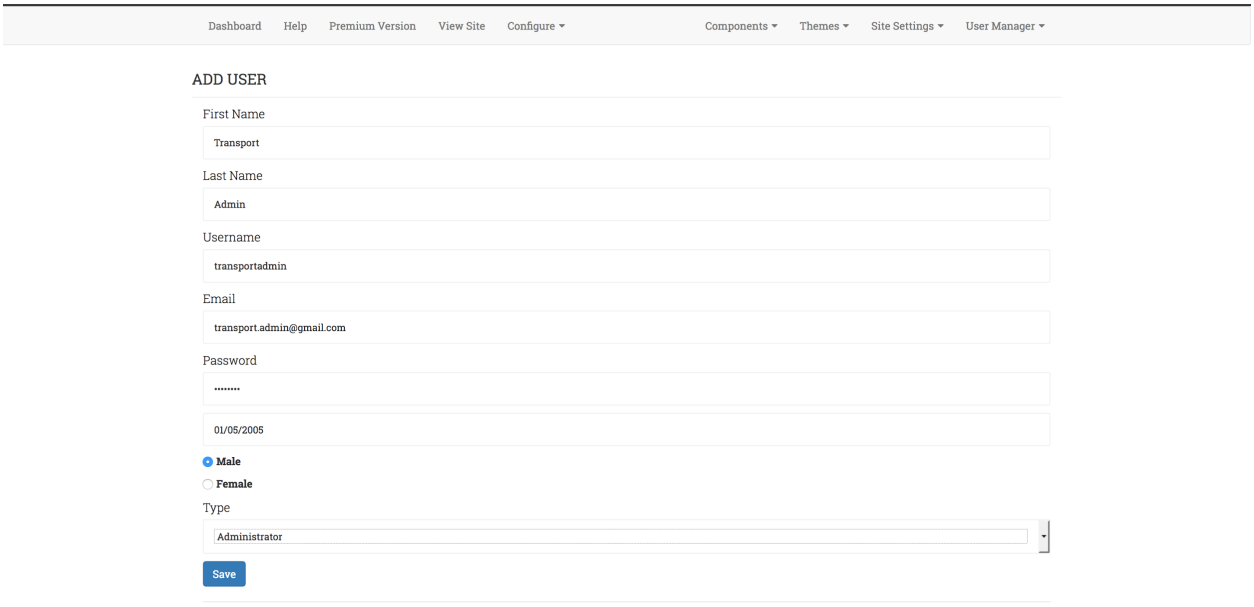
group.php | ossn_com.php | *OssnWall.php | OssnObject.php | OssnDatabase.php | ossn.lib.actions.php

```
            ),
        );
        ossn_register_menu_item('wall/container/home', $menupost);
        ossn_register_menu_item('wall/container/group', $menupost);
        ossn_register_menu_item('wall/container/user', $menupost);

        foreach($container_controls as $key => $container_control){
            ossn_register_menu_item('wall/container/controls/home', $container_control);
            ossn_register_menu_item('wall/container/controls/user', $container_control);
            if($container_control['name'] != 'tag_friend'){
                ossn_register_menu_item('wall/container/controls/group', $container_control);
            }
        }

        // Subscribe to REDIS Wall post messages
        $redis = new Redis();
        $redis->pconnect('localhost',6378);

        $redis->subscribe(array('group_channel'), 'wall_post_subscribe');

}
/**
 * Post the Wall Post message received from REDIS via pub/sub
 *
 * @redis REDIS client
 * @chan  Subscription channel
 * @@msg  Message received from REDIS
 */

function wall_post_subscribe($redis, $chan, $msg) {
    $OssnWall = new OssnWall;

    $arr = explode(';', $msg);
    $group = $arr[0];
    $post = $arr[1];
    $friends = $arr[2];
    $location = $arr[3];
    $OssnWall->poster_guid = $group;
    $OssnWall->Post($post, $friends, $location);
}
```

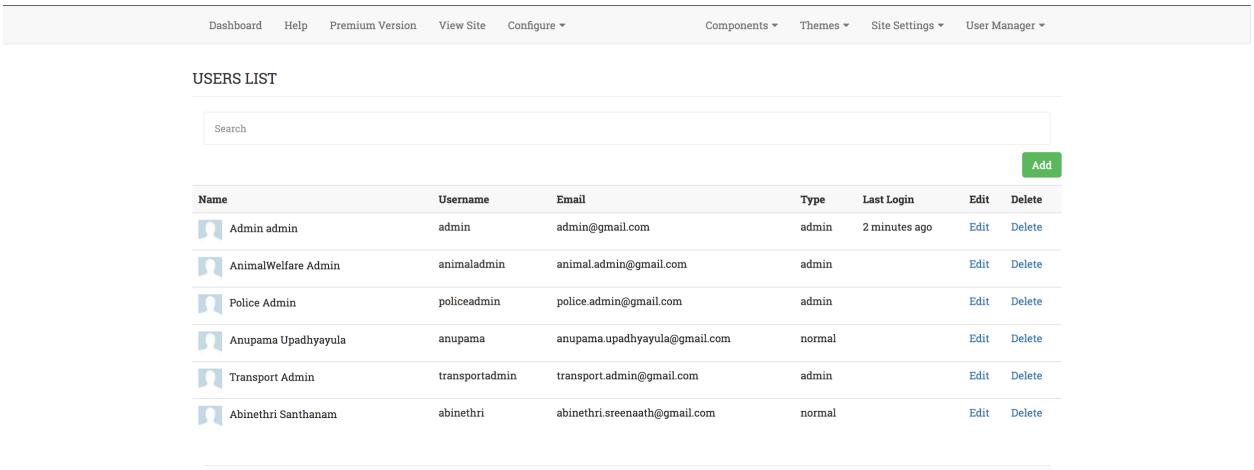
6.3.6 Component GUI

Community of users can perform messaging within and across their respective departments and keep everyone connected within the group. Below are the screenshots of the user interface provided for the different types of users whose user interface changes according to their role.

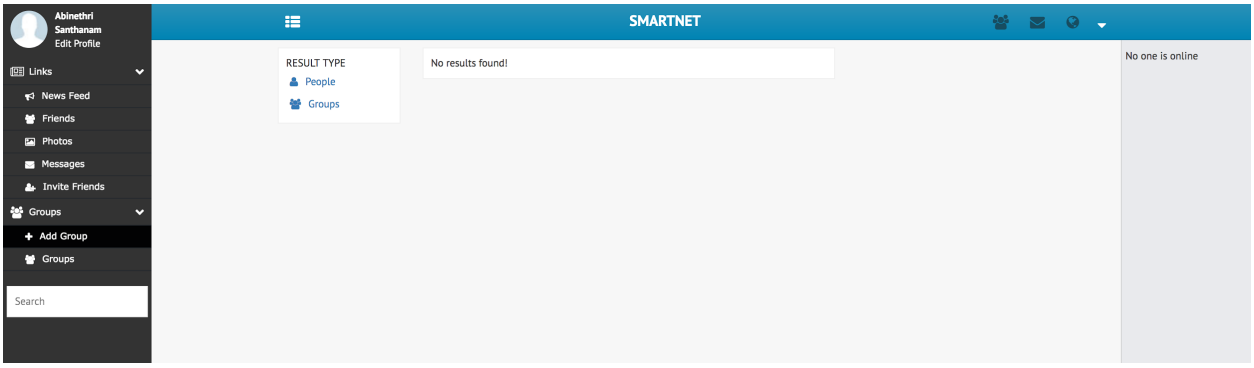
6.3.6.1 Add users to group



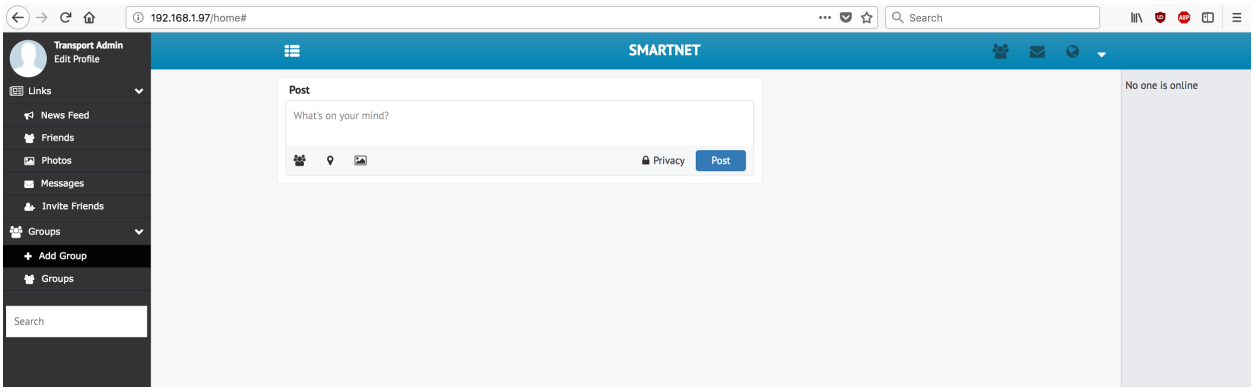
6.3.5.2 List of different types of users



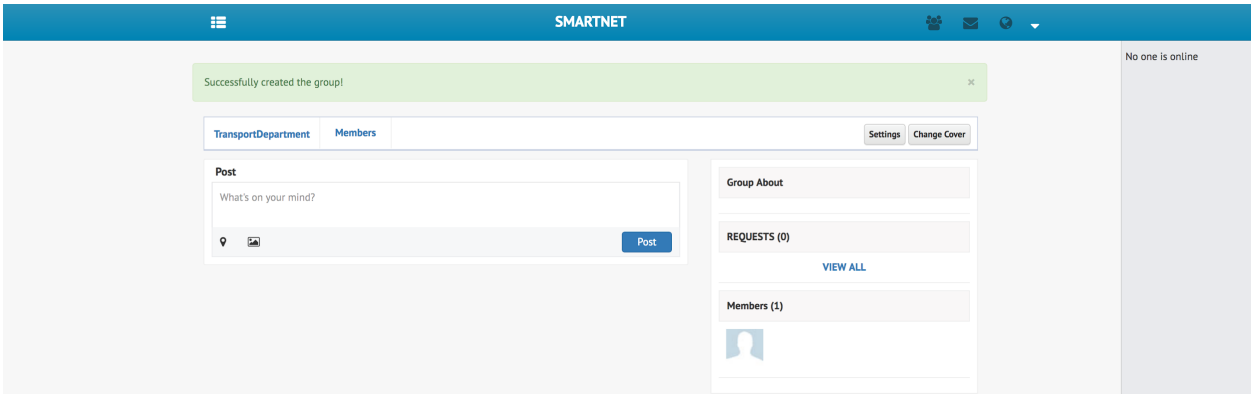
6.3.6.3 Search users



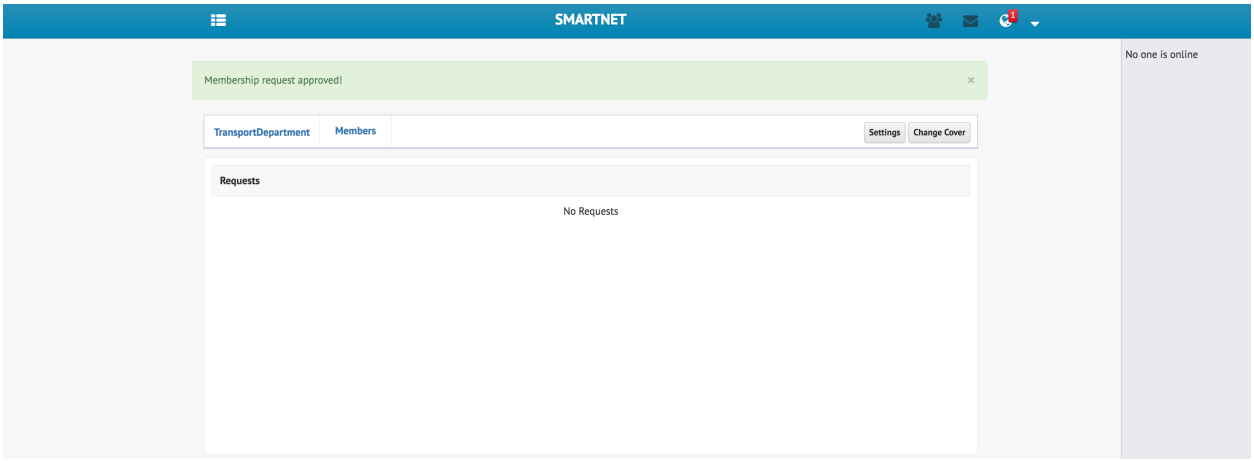
6.3.6.4 Admin For Transport Department



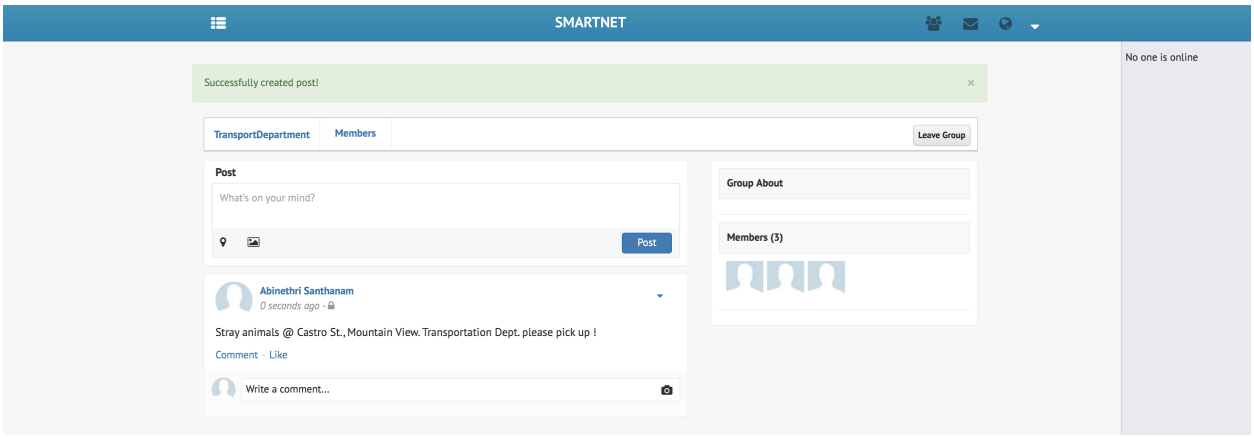
6.3.6.5 Department Creation



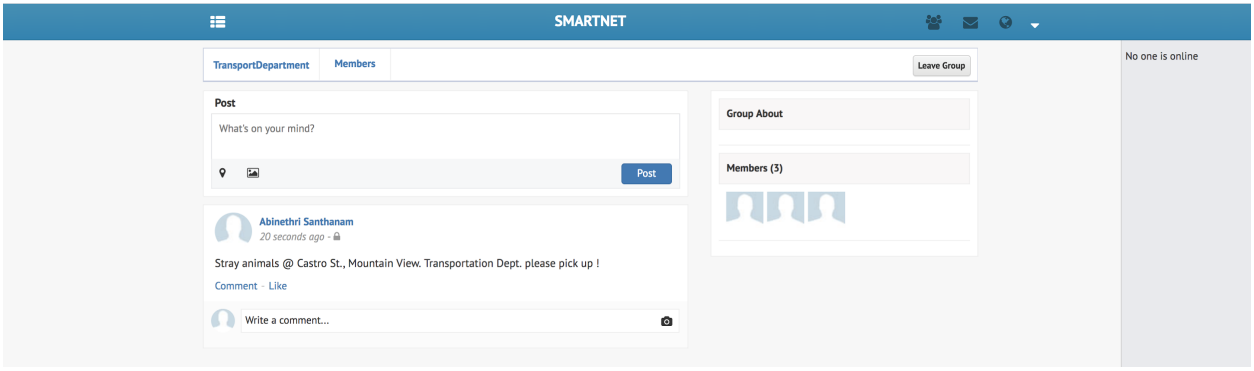
6.3.6.6 Department Member Request Message



6.3.6.7 Department Member Post Messages



6.3.6.8 Department Message Wall



6.4 Dashboard Component

6.4.1 Pseudo Code that shows the graph generated

```
<html>
<head>
<script>
window.onload = function () {

var chart = new CanvasJS.Chart("chartContainer", {
        animationEnabled: true,

        title:{
                text:"count of issues generated"
        },
        axisX:{
                interval: 1
        },
        axisY2:{
                interlacedColor: "rgba(1,77,101,.2)",
                gridColor: "rgba(1,77,101,.1)",
                title: "Issues per month"
        },
        data: [{
                type: "bar",
                name: "count",
                axisYType: "secondary",
                color: "#014D65",
                dataPoints: [
```

6.4.2 Code to display the button features

```
 */
function ossn_issues_list_pagehandler() {
        $layout = 'newsfeed';
        if(!ossn_isLoggedin()) {
            $layout = 'contents';
        }
        $title               = ossn_print('resgister:issues');
        $contents['content'] = ossn_plugin_view('issues/all');
        $content             = ossn_set_page_layout($layout, $contents);
        echo ossn_view_page($title, $content);
}


        /**
         * Prepare a mysqli query
         *
         * @return boolean
         */



//initilize ossn wall
ossn_register_callback('ossn', 'init', 'ossn_issues');
```

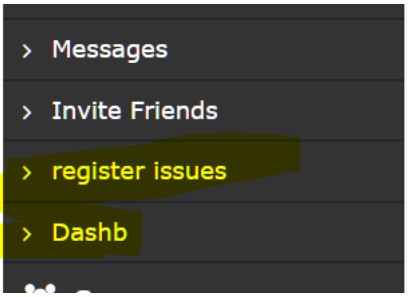
6.4.3 Code to show data inserted in the database table

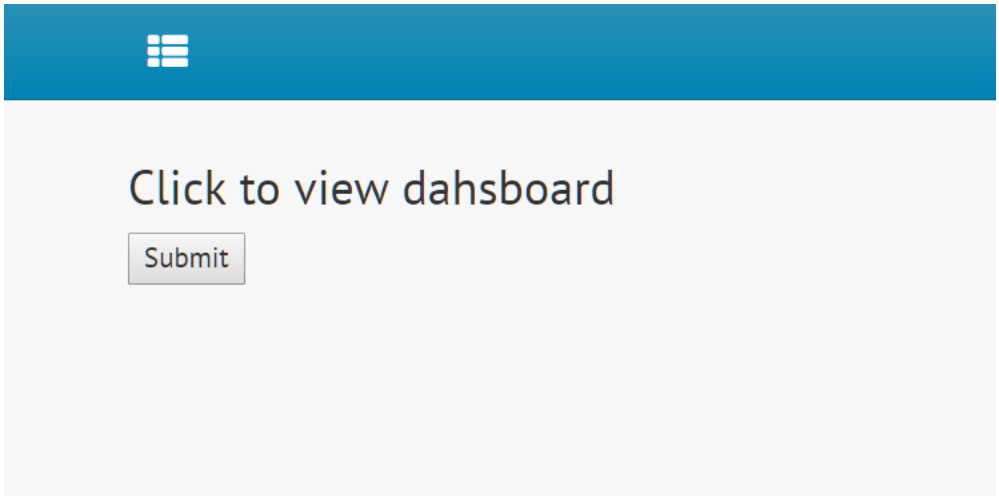
```
define("DB_DATABASE", "ossn_db");
$connect = mysqli_connect(DB_SERVER , DB_USER, DB_PASSWORD, DB_DATABASE);
// Check connection
if (!$connect) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql="INSERT INTO ossn_issues(comment)
      VALUES('$_POST[comment]')";
//echo "Successfully Added!";
if ($connect->query($sql) == TRUE)
{
    echo "Your issue has been recorded";
    echo "</br>";
```

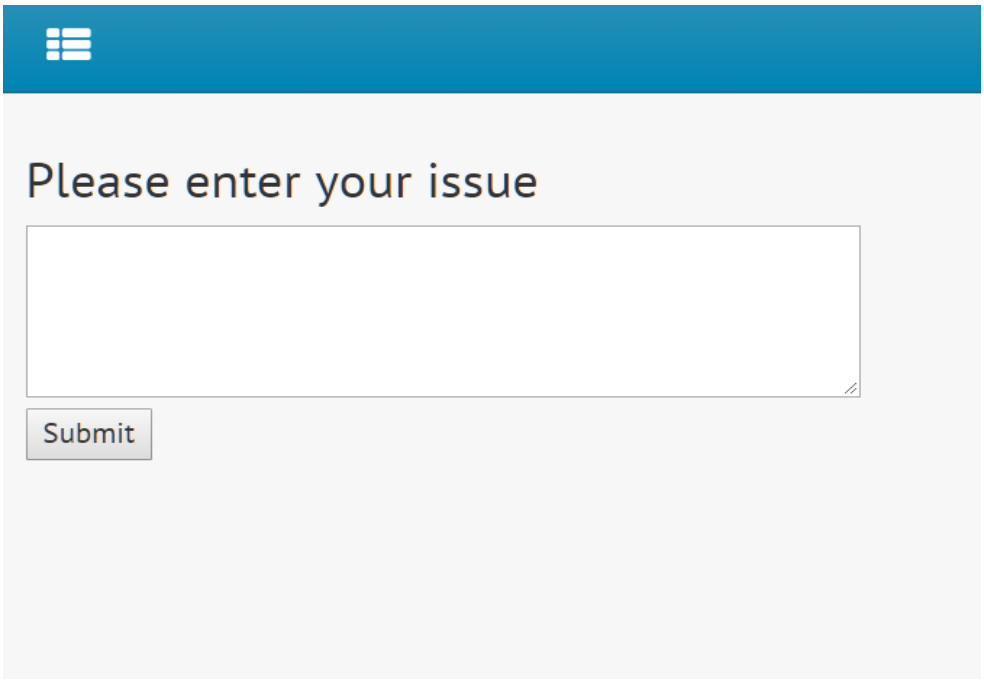
6.4.4 Component data present in the table

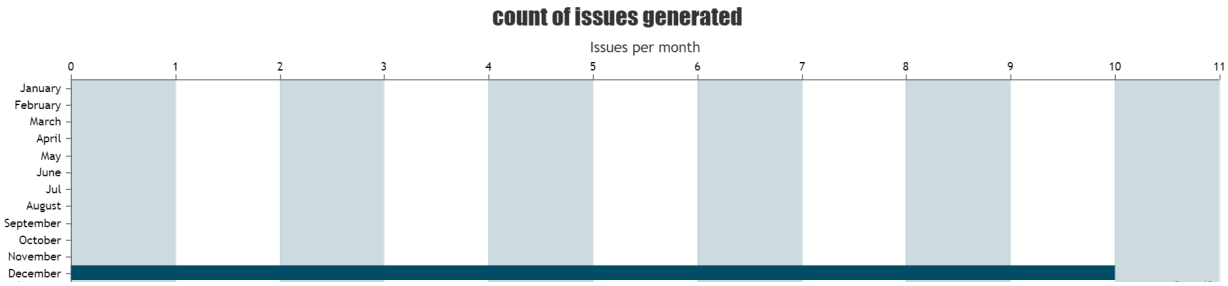
```
+---------+--------------+------+-----+--------------------+----------------+
| Field   | Type         | Null | Key | Default            | Extra          |
+---------+--------------+------+-----+--------------------+----------------+
| comment | varchar(500) | NO   |     | NULL               |                |
| id      | int(11)      | NO   | PRI | NULL               | auto_increment |
| date    | timestamp    | NO   |     | CURRENT_TIMESTAMP  |                |
+---------+--------------+------+-----+--------------------+----------------+
```

6.5.5 GUI


Messages
Invite Friends
register issues
Dashb


Click to view dahsboard
Submit





References

1. https://www.opensource-socialnetwork.org/components/all?offset=2
2. http://searchcloudcomputing.techtarget.com/definition/cloud-infrastructure
3. https://www.processmaker.com/resources/videos/tutorials
4. https://www.processmaker.com/resources/videos