Software Design Specification

for

Web application for “Weather Insulations”

Prepared by

Hridaya Annuncio
U101116FCS046

Bollam Sreekar Reddy
U101116FCS020

Ira Mishra
U101116FCS048

NIIT University
Software Engineering course project
12th November 2018

Table of Contents

1. Introduction

1.1. Introduction

1.1.1. The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software 50 should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including use case models, sequence diagrams, collaboration models, object behaviour models, and other supporting requirement information.

1.2. Purpose of this Project

1.2.1. This document will define the design of the one runway simulator. It contains specific information about the expected input, output, classes, and functions. The interaction between the classes to meet the desired requirements are outlined in detailed figures at the end of the document.

1.3. Scope of the development project

1.3.1. We describe what features are in the scope of the software and what are not in the scope of the software to be developed.

1) Inscope:
    a) Ability for a registered user to ask queries regarding the services.
    b) Request to the manager for a service to be provided.
2) Out of Scope:
    a) Online Payment once a work is completed.

1.4. Definitions, acronyms, and abbreviations

1.4.1. IEEE: Institute of Electrical and Electronics Engineers.
1.4.2. SDS: Software Design Specification.

1.5. References

1.5.1. R. S. Pressman, Software Engineering: A Practitioner's Approach, 5th Ed, McGraw-Hill, 2001.
1.5.2. IEEE SDS template

1.6. Overview of document

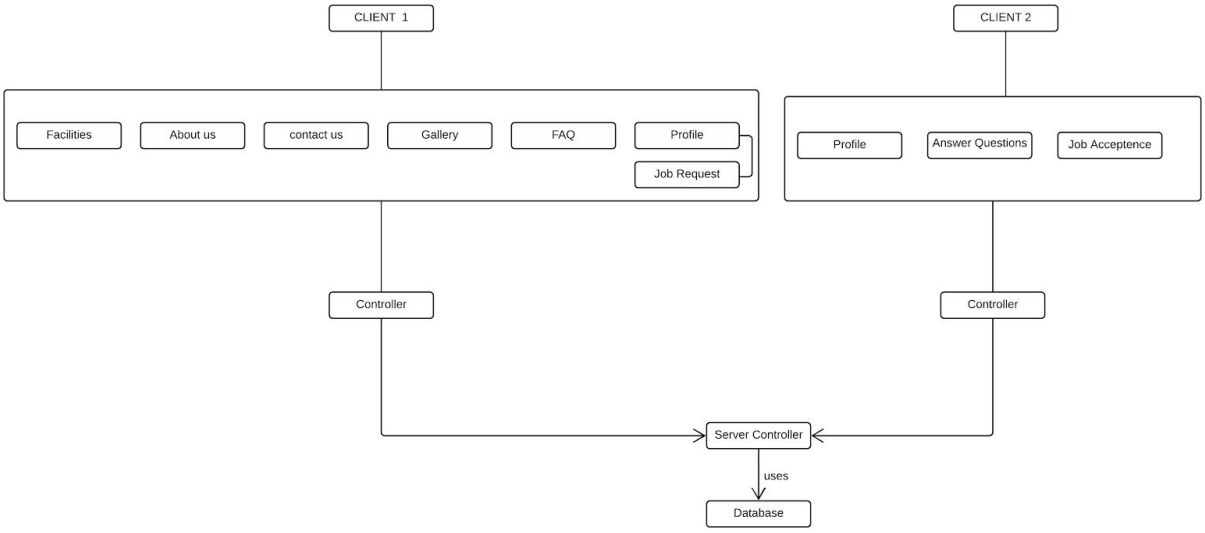
1.6.1. Class Diagram: Static diagram that gives information about the various classes and relationship among them.

1.6.2. Sequence Diagram: Diagramatic representation of an event, tells how the classes communicate with each other in a scenario.

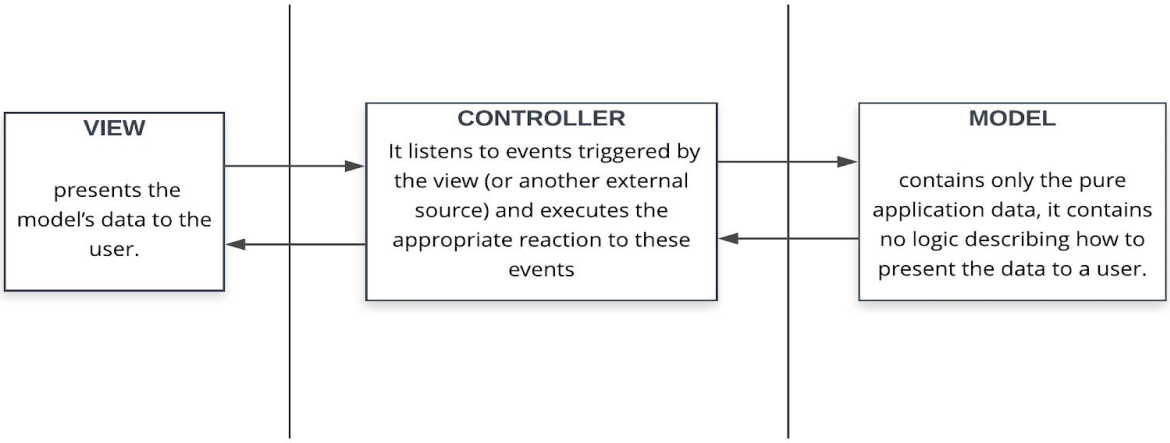
1.6.3. State Diagram: Gives information about how an event changes state of a class here state refers to fields within the class.

2. Conceptual Architecture/Architecture Diagram

Architecture Diagram 1:



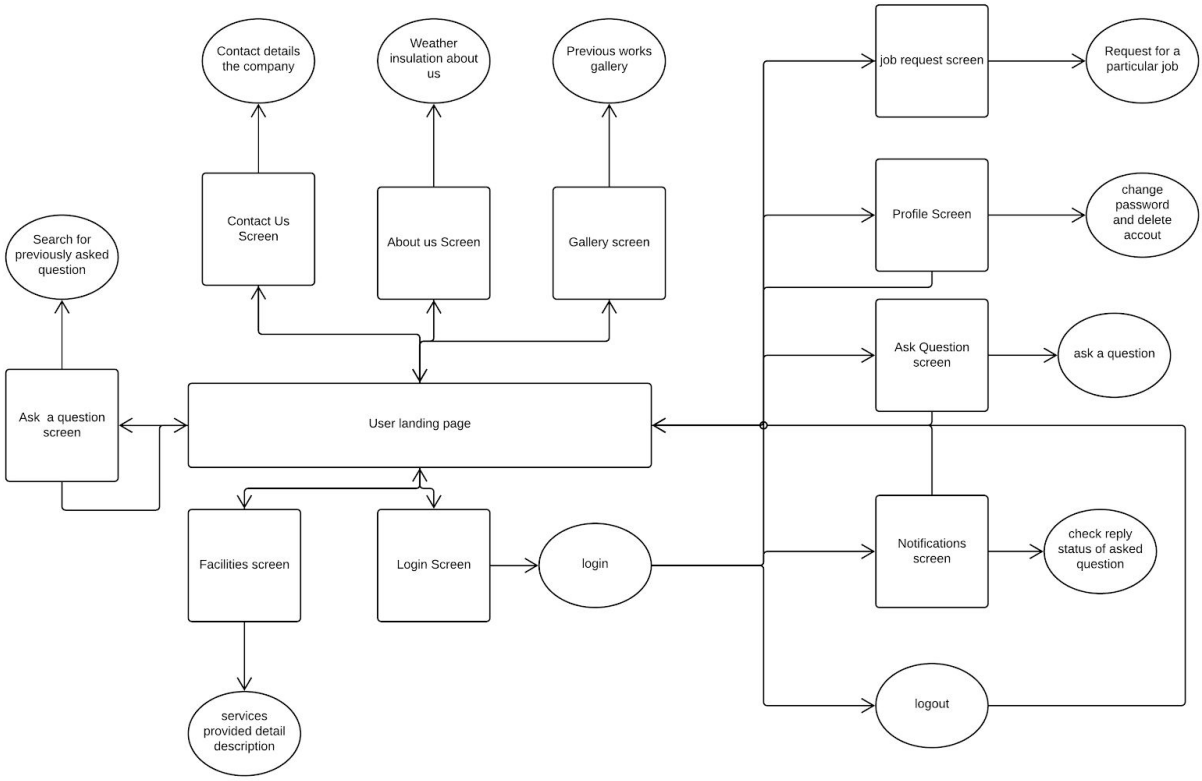
2.1. Overview of modules / components



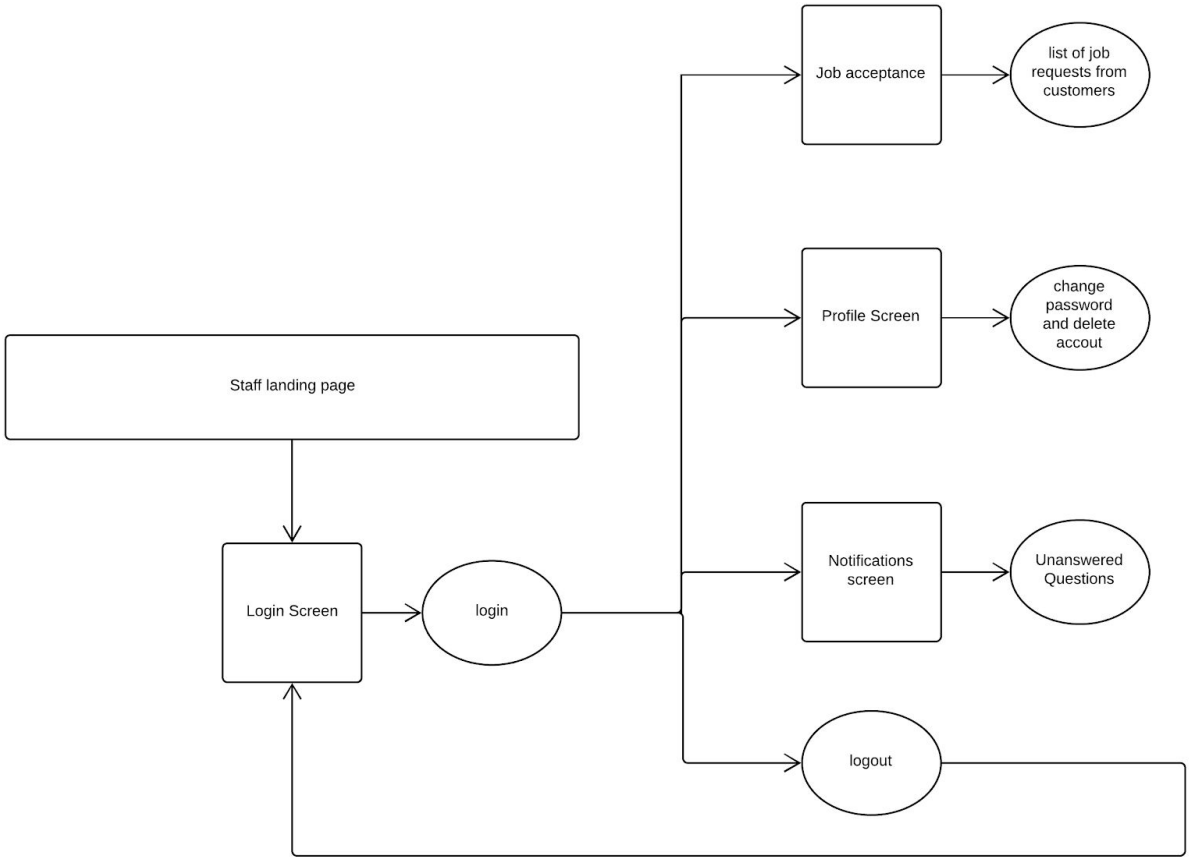
NOTE: The horizontal lines represent the separation of modules.

2.2. Structure and relationships

2.2.1. User side



2.2.2. Staff side



NOTE: The boxes represent individual screens. The circles represent actions that do not have screens. The arrows represent navigation between screens.

3. Logical Architecture

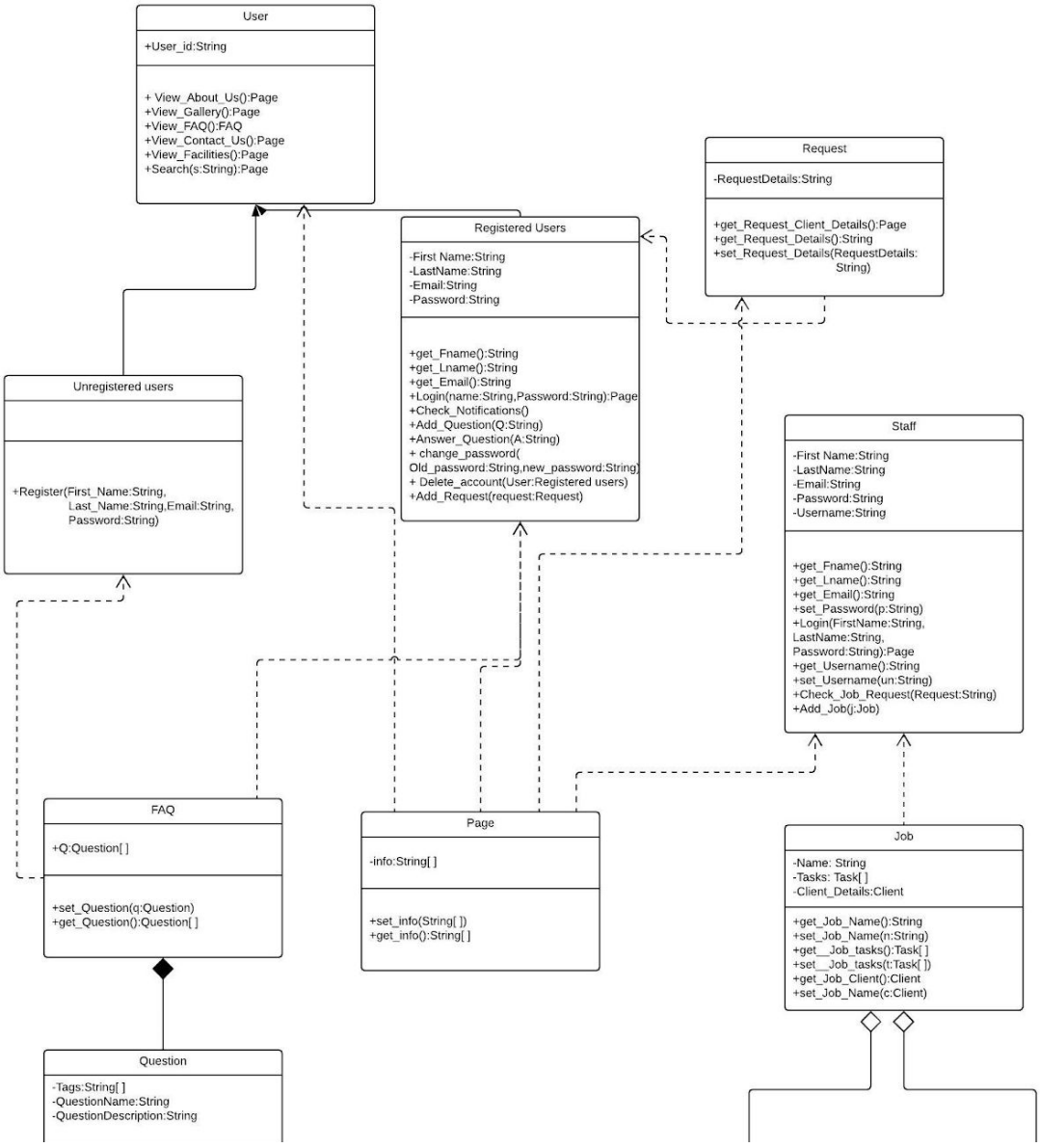
3.1. Class Diagram:

Every job requested is associated with a task that needs to be performed and a client who requested for the task their existence is meaningless without the presence of the of the job class so **aggregation dependency** between client class and job class makes perfect sense similarly between task class and job class.

Once a user is registered he has a ability to post to question on the FAQ page so the entire FAQ section is a composition of questions posted by different users so **composition dependency** between them is perfectly reasonable.

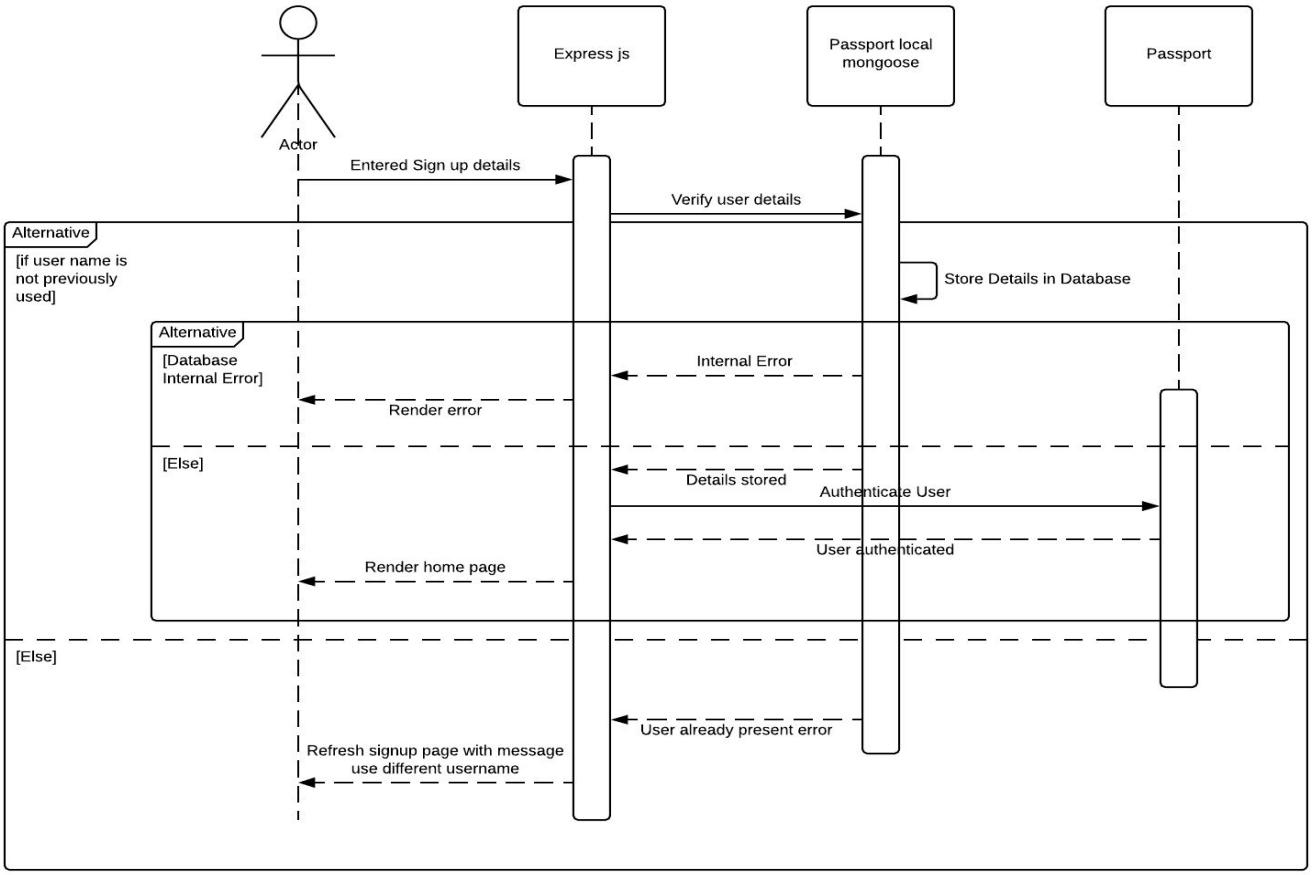
Dependency of Page and FAQ class with Registered_users class says that the registered uses them and there is no dependency between unregistered users and FAQ as they don't get to interact with it.

Both Registered and Unregistered users **inherit** the class User. The User class has functions that can be performed by both the classes



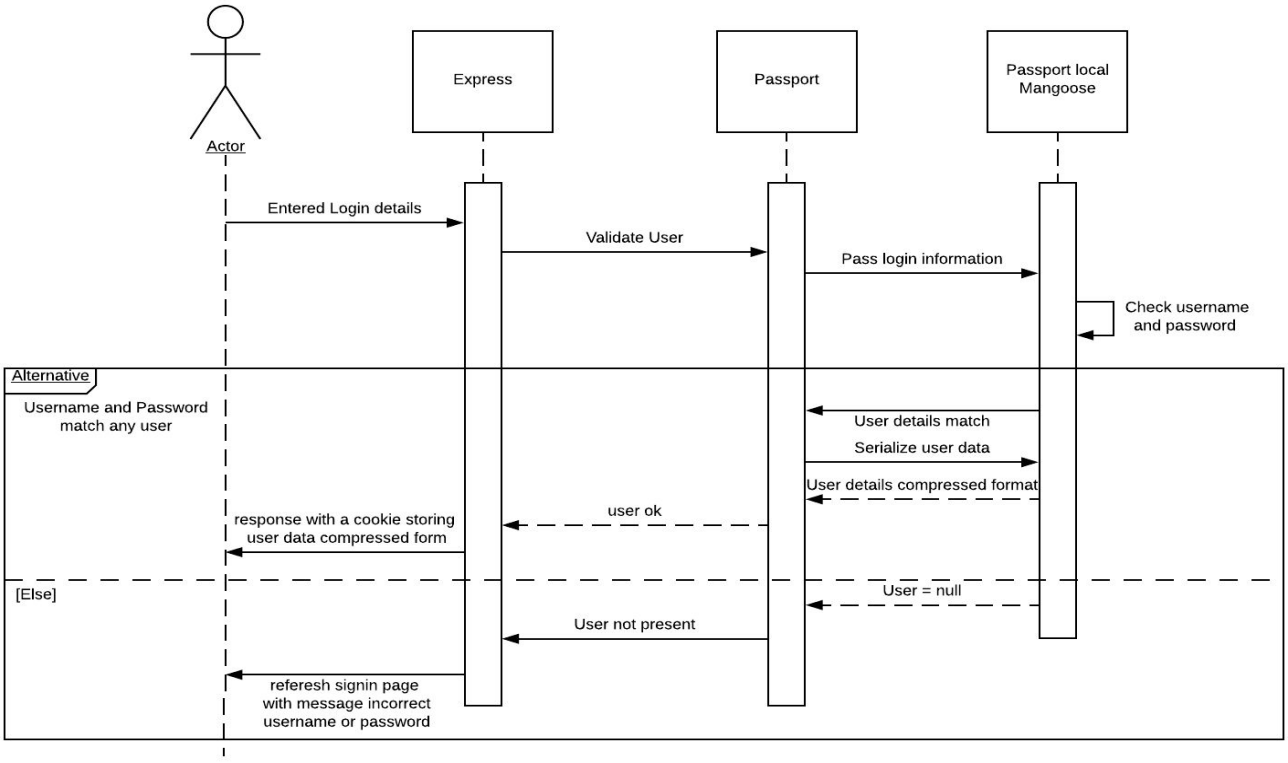
3.2. Sequence Diagram:

3.2.1. Sign up



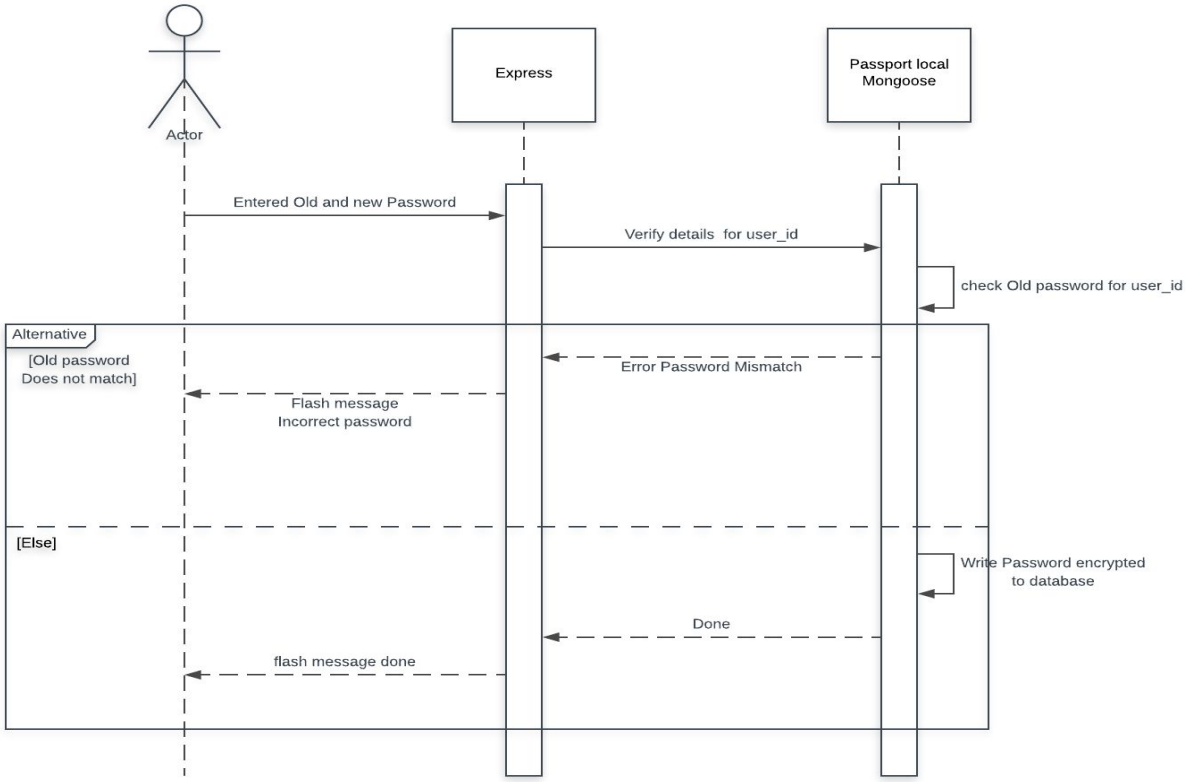
Explanation: To register a user a request for signup page is needed and the user is expected to enter the details in the presented text boxes in a certain pattern like the password should be of min-length 6 and contain at least one letter which is managed by client side javascript these details are routed to passport local mongoose for the verification process here the Object checks if the username entered is unique if not express renders the same “sign in” page with a flash message “try different username”, if the details entered are unique passport local mongoose object communicates with the database to store the new User details were the password is stored in an encrypted format but if the communication with the database fails express if informed to render Internal error page. Once a user is registered he/she will be logged in and then directed to the home page

3.2.2. Sign in:



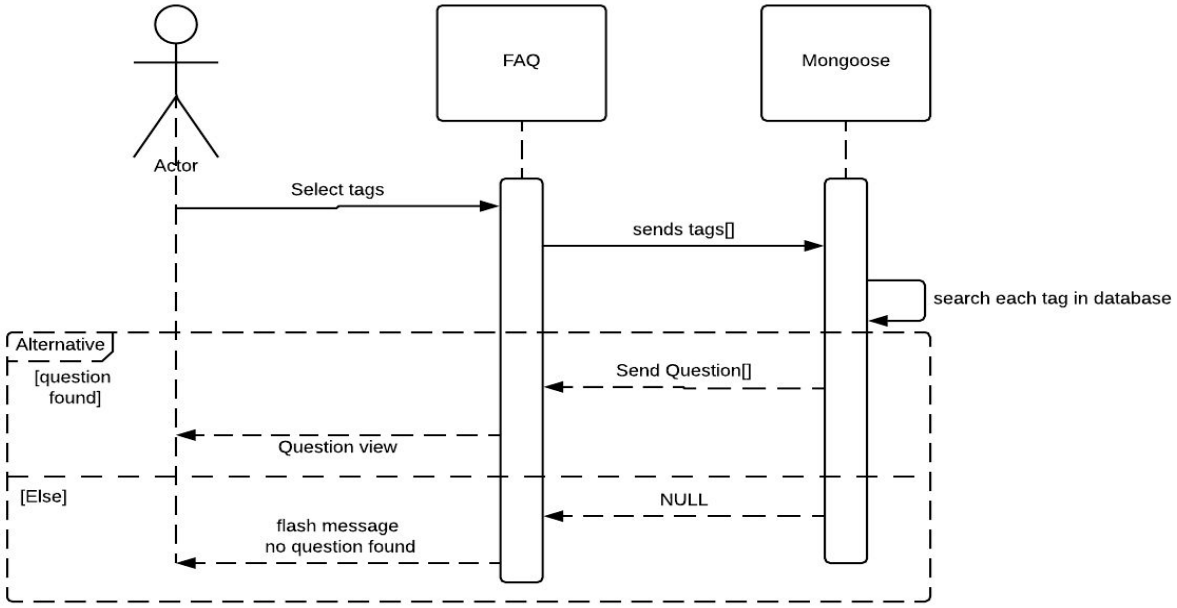
Explanation: To login user should request for route /signin user is then expected to enter the credentials validation information expected from the passport object. Passport object further calls local Strategy method which is given by passport local mongoose which includes details like how should the authentication be done passport-local-mongoose authenticates the user details are valid this message is sent to passport class, which then creates a compressed form of the user details and store them in cookies so as to make data persistent but if the user details are incorrect a flash message with Incorrect username.

3.2.3. Change password:



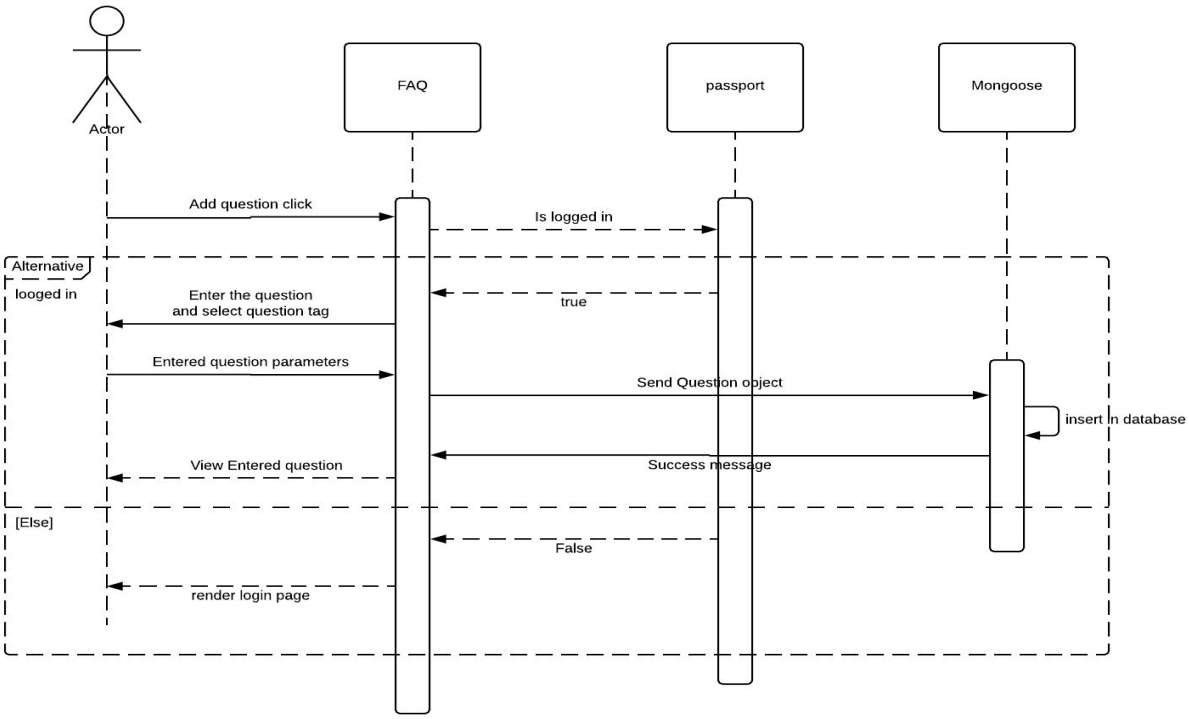
Explanation: Once the user is logged in they have an ability to change password by sending request to the route /:id/settings user is expected to enter the current password and the new password to verify the current the password and the store the new_password in the database call to the change_password method of passport local mongoose is made which checks if the current password entered is correct, the method rewrites it with new_password (in salt and hash form) but if the current password is wrong error message with name “Incorrect Password Error” is thrown express then responds with message wrong password entered.

3.2.4. Search a Question:



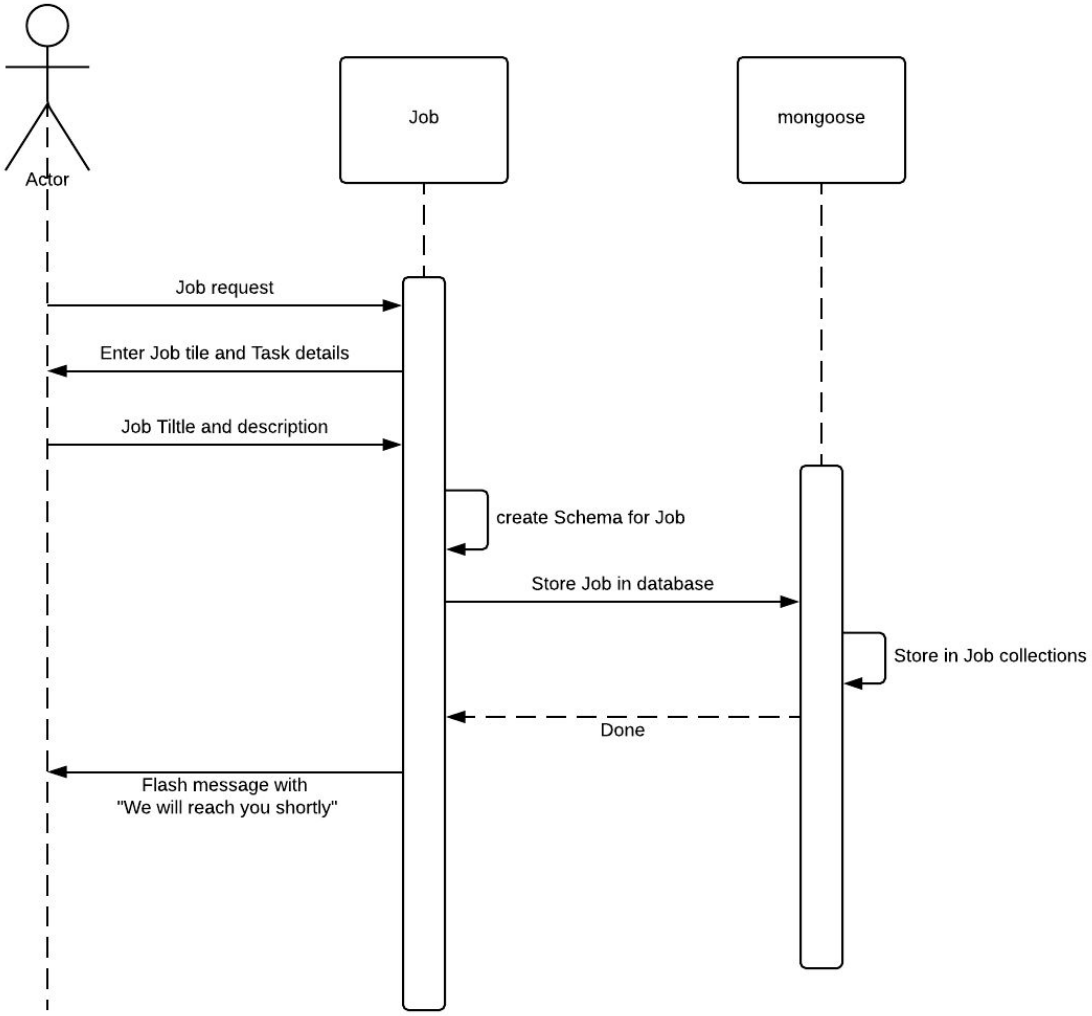
Explanation: User needs route to /questions search for a particular type of question to use this functionality user need not be logged in, user is presented with few checkboxes which are various types of questions present in the database according to the checkbox selected user is presented with various thread with questions and replies various users and staff but if no such question of such type was asked before FAQ page render a message “No Question found”

3.2.5. Add Question:



Explanation: User needs route to /questions search for a particular type of question to use this functionality user need be logged in,which is checked by passport isauthenticated function and then the question tiles description and tags that are suitable for the question once these details are filled Question object is created and stored in the database, this is presented to the user. Each question is attached to the thread object which contains question ID and replies_id[].

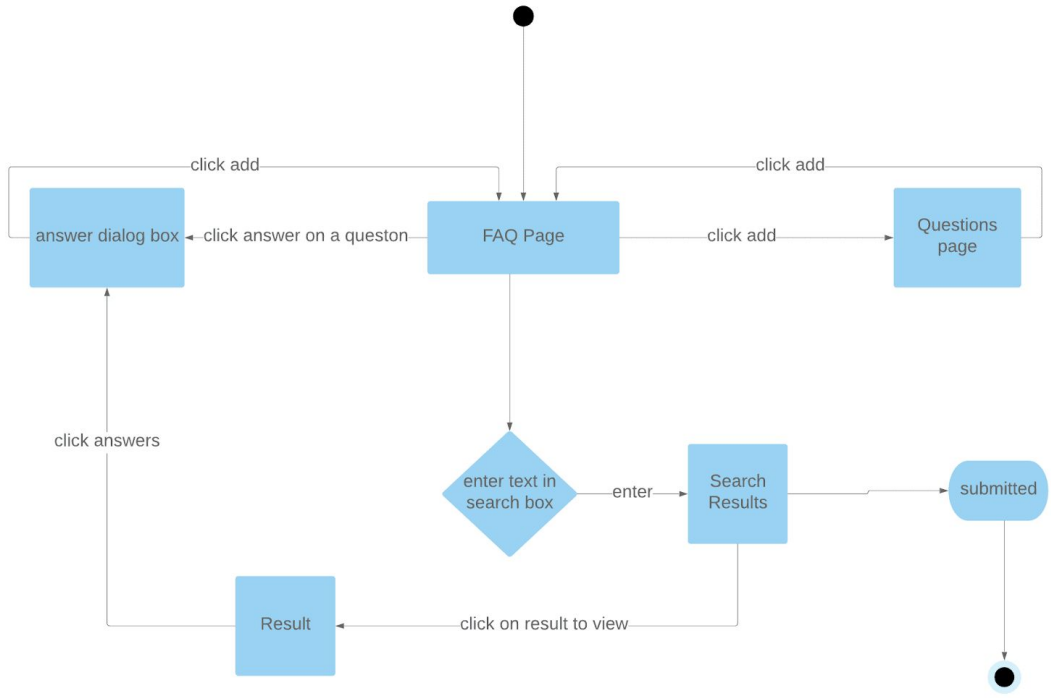
3.2.6. Request for a job to be done



Explanation: A logged in user can request for service to be done from the company they need to route to /job_request enter the job description which are stored in the database and later viewed to the staff member along with the client details from the staff member attends to the user.

3.3. State Diagram

3.3.1. FAQ Page



Explanation: On the FAQ page, there are 3 options.

1) You can click on the button"Add" to add a question. On clicking the button, a dialog box will appear in which the user has to add important words to identify this question i.e. "tags". Then in the next box the user has to add the actual question. On clicking done, the question gets added to the database and will be displayed on the FAQ page.

2) User can, while viewing the questions, click on "Answer" button under a question. Again a dialog box will appear in which the user is to write the answer and subsequently click on "Add". Thus now the Answer would be displayed under that particular question

3) User can enter a word in the search bar. Here, these words will compared with the tags of all questions present on the FAQ page. Any question that has tags similar to the words entered in the search bar will be displayed on a customized FAQ page.

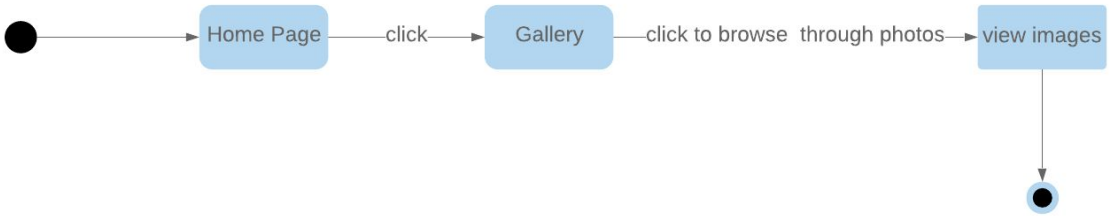
3.3.2. About us

Explanation: On the home page, we can view the task bar on which if we click the button “About Us” we would reach the About Us page.

3.3.3. Contact us

Explanation: On the home page, we can view the task bar on which if we click the button “Contact Us” we would reach the Contact Us page.

3.3.4. Gallery

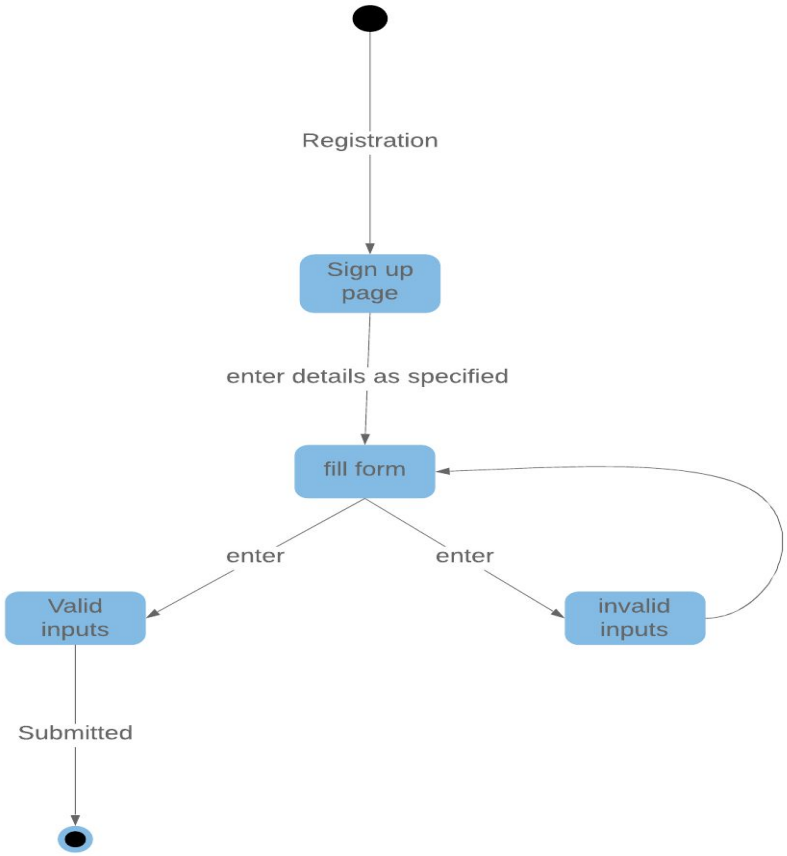


Explanation:On the home page, we can view the task bar on which if we click the button “Gallery” we would reach the Gallery page. Here On clicking a picture we can view it in full screen mode.

3.3.5. Facilities

Explanation:On the home page, we can view the task bar on which if we click the button “Facilities” we would reach the Facilities page.

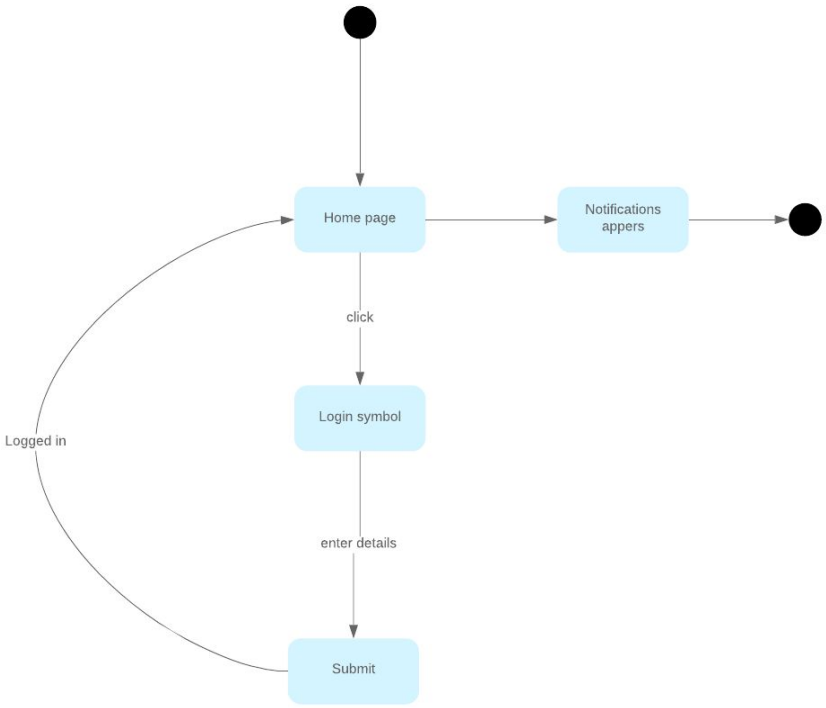
3.3.6. Registration page



Explanation: On the Registration page, you can sign up for an account. The following are the details required.

1. First name (should contain only alphabets)
2. Last name (should contain only alphabets)
3. Gender (only one option between male, female and others)
4. Email (must be a valid email id)
5. Password(alphanumeric characters plus special characters special chars like "@_$#" are allowed but not compulsory , should contain at least 6 characters and at least one numeric)
6. Phone number(must consist of 10 digits only,number must start with 6/9/8/7 only)

3.3.7. Login

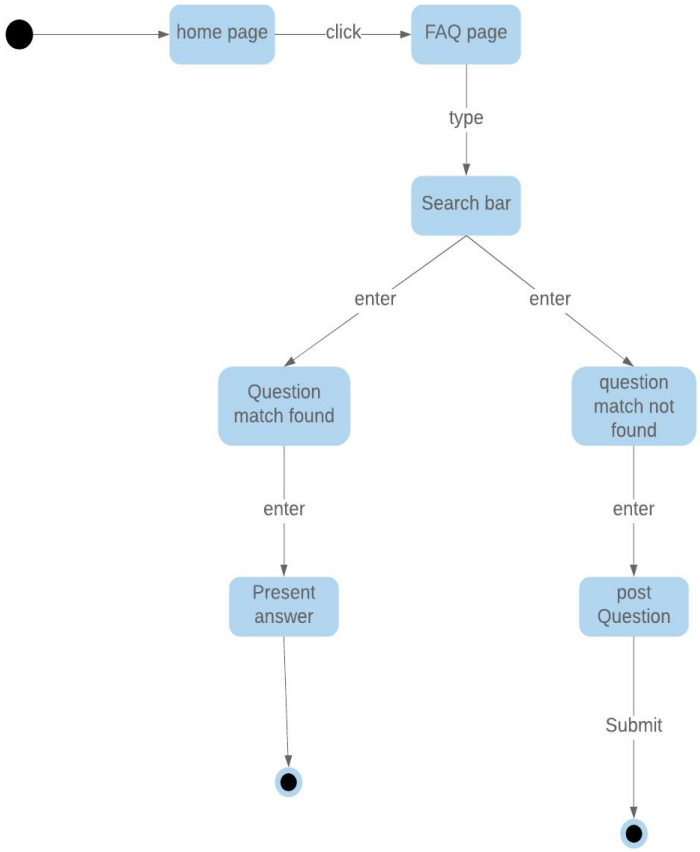


Explanation:In login, the user has to first click a special button present on the right most side of the homepage, it takes the user to the login page where they fill in their details and can get logged into their accounts.
The page from there will be redirected to the homepage where the above mentioned symbol would act as a notifications button and upon clicking that the user will be updated about:-

1) New reviews
2) New products
3) New informations about existing products
4) If queries have been answer

5) Replies to queries posted by user

3.3.8. Searching Question/Writing Question



Explanation: User needs route to /questions search for a particular type of question to use this functionality user need not be logged in, user is presented with few checkboxes which are various types of questions present in the database according to the checkbox selected user is presented with various thread with questions and replies various users and staff but if no such question

of such type was asked before FAQ page render a message “No Question found” and that leads to the option of posting a new question.

4. **Class Details**

4.1 **Unregistered User**

4.1.1 **Variables**

NILL

4.1.2 **Functions**

4.1.2.5 Register(String first_name, String last_name,Username,Phone_no,password)

Lets unregistered user register the password provided should contain at least one numeric and should be min length of 6 chars

4.2 **Registered User**

4.2.1 **Variables**

4.2.1.1 First Name

This is a variable of String type. It is private. It stores the value of the first name of the particular staff member.

4.2.1.2 LastName

This is a variable of String type. It is private. It stores the value of the last name of the particular staff member.

4.2.1.3 Email

This is a variable of String type. It is private. It stores the value of the staff member's email id.

4.2.1.4 Password

This is a variable of String type. It is private. It stores the value of the staff member's account password.

4.2.1.5 Username

This is a variable of String type. It is private. It stores the value of the username of the staff member's account.

4.2.2 **Functions**

4.2.2.1 Search(String s)

Takes input as a string extracts tags[] from the string and sends the request to the database to fetch the results output id the questions displayed on a html page.

4.2.2.2 Login(String email,String password):

Registered user enters email and password they are authenticated using the data stored in the database the output here is the home page if authentication is successful if not out is login page with try again message.

4.2.2.3 Check_notification():

Output is html page with that gives information that contain latest replies for the questions asked by the user.

4.2.2.4 Add_Question(String question):

Output is void question entered in string format is stored in database and later retrieved on the FAQ page waiting for the replies.
delete_accout

4.2.2.5 Answer_Question(String Reply):
Output is void reply made for the question is stored in the database attached to the question for which the reply is made.

4.2.2.6 Change_password(String Old_password,String new_password)
Output is changed password in database if the oldpassword matched else flash message with wrong password entered

4.2.2.7 Delete_account(String user_name)
Taking username as input this method deletes user information and data associated with that user from database

4.2.2.8 Add_Request(request)
Here the registered user can send a request to the company.An object of class Request is made and the request string is taken along with the logged in accounts basic information.
There is no return value.

4.3 FAQ

4.3.1 **Variables**

4.4.1.1 public Question questions[]
Object array of type question contains then list of question asked previously.

4.3.2 **Functions**

4.3.2.1 set_Question(Question [])
Setter function that aids in adding question to list.

4.3.2.2 get_Question(String Question_id)

Fetching a particular question from the list

4.4 Question

4.4.1 **Variables**

4.4.1.1 private String Question_data
Variable that stores data about the question in string format.

4.4.1.2 private String[] tags:
String array that contains list of tags selected by the user

4.4.2 **Functions**

4.4.2.1 set_Question(String Question)
Setter function that aids in adding question.

4.4.2.2 get_Question(String Question_id)
Fetching a particular question using the question_ID.

4.4.2.1 set_Tags(String str[])
Setter function that aids in adding a tag to the question to list.

4.4.2.2 Question[] get_Tags(String Tag_id)
Fetching all question of certain tag.

4.5 Staff

4.5.1: **Variables**

4.5.1.1 First Name

This is a variable of String type. It is private. It stores the value of the first name of the particular staff member.

4.5.1.2 LastName

This is a variable of String type. It is private. It stores the value of the last name of the particular staff member.

4.5.1.3 Email

This is a variable of String type. It is private. It stores the value of the staff member's email id.

4.5.1.4 Password

This is a variable of String type. It is private. It stores the value of the staff member's account password.

4.5.1.5 Username

This is a variable of String type. It is private. It stores the value of the username of the staff member's account.

4.5.2 **Functions**

4.5.2.1 get_Fname()/get_Lname/get_Username()/get_Email

All of these are the getter functions of 4 variables of the class i.e. they help in getting the values of FirstName,LastName, Email and Username which are all private variable and cannot otherwise be got.The return types of all these functions is of String type as the respective variable values are being returned.These functions help in displaying the staff member's personal details if and when required and also to display details on the profile page.

4.5.2.2 Set_Username(String un)/set_Password(String p)

Both these values i.e. that of the Username and Password of the account that the staff member holds, are initially set using these functions. Later on, if the staff member wants to change his/her Username or Password, he/she can do it by calling this function.Both these functions have 1 single parameter of String type and have no return value.

4.5.2.3 Page Login(String name,String password)l
Here a staff member can login by entering their Username and password as the parameters. In return, they will get a display of type Page(will is defined in 3.4). Here the Profile page would be the page that is returned.

4.5.2.4 Check_Job_Request(Request_String)
Here, the parameter comes on clicking on a job that was requested i.e. the request is in the form of a String and this gets passed on as a parameter on calling the function which happens on clicking on that particular request.

4.5.2.5 Add_Job(j:Job)
On the Job List page, if user clicks on “Add Job” a dialog box appears in which all of the job’s details are required to be entered.
On clicking”Add” this function is called passing all these details as one single parameter of Job type(Defined in 4.7).

4.6 Job

4.6.1: Variables

4.6.1.1 Name
This is a variable of String type. It is private. It stores the value of the name of the job (short form details)

4.6.1.2 tasks
This is an array of type Task(Defined in 4.7). It stores the values of all the tasks of that particular Job.

4.6.1.3 Client_Details
This is of Client type(Defined in 4.8). It stores the details of the client who has requested for that particular job to get done.

4.5.2 **Functions**

4.6.2.1 String get_Job_Name()
It is the getter function of the variable Name. It has a return type of String.

4.6.2.2 Task[ ] get_Job_tasks()
It is the getter function of the array variable Tasks. It has a return type of array of Task(Defined in 4.8).

4.6.2.3 Client get_Job_Client()
It is the getter function of the variable Client_Details. It has a return type of Client(Defined in 4.7).

4.6.2.4 set_Job_Name(String n)
It is the setter function of the variable Name. It has a parameter of String type.

4.6.2.5 set_Job_tasks(Task[ ] t)
It is the setter function of the array variable Tasks. It has a parameter of the type array of Task(Defined in 4.8).

4.6.2.6 set_Job_Client(Client c)
It is the setter function of the variable Client_Details. It has a parameter of type Client(Defined in 4.7).

4.7 **Client**

4.7.1 **Variables**

4.7.1.1 Name

This is a variable of String type. It is private. It stores the value of the name of the Client (short form details)

4.7.1.2 Phone No

This is a private variable of String type. It stores the client's phone number.

4.7.1.3 Email

This is a private variable of String type. It stores the client's Email Id.

4.7.2 **Functions**

4.7.2.1 String get_Name()

It is the getter function of the variable Name. It has a return type of String.

4.7.2.2 String get_Phone_No()

It is the getter function of the variable Phone No. Thus its return type is String.

4.7.2.3 String get_Email()

It is the getter function of the variable Email. Its return type is String.

4.7.2.4 set_Name(String n)

It is the setter function of the variable Name. It has a parameter of String type.

4.7.2.5 set_Job_tasks(String d)

It is the setter function of the variable Phone No. It has a parameter of the type String.

4.7.2.5 set_Job_tasks(String d)

It is the setter function of the variable Email. It has a parameter of the type String.

4.8 Task

4.8.1 Variables

4.8.1.1 Name

This is a variable of String type. It is private. It stores the value of the name of the task (short form details)

4.8.1.2 Details

This is a full fledged description of the task. Thus it is of String type. It is a private variable.

4.8.2 Functions

4.8.2.1 String get_Task_Name()

It is the getter function of the variable Name. It has a return type of String.

4.8.2.2 String get_task_details()

It is the getter function of the variable Details. Thus its return type is String.

4.8.2.3 set_Task_Name(String n)

It is the setter function of the variable Name. It has a parameter of String type.

4.8.2.4 set_Job_tasks(String d)

It is the setter function of the variable Details. It has a parameter of the type String.

4.9 Page

4.9.1 Variables

4.9.1.1 info[]

This is an array of String type. It stores the entire information that is supposed to be displayed on the page

4.9.2 Functions

4.9.2.1 set_info(String i[ ])

The setter function of the variable info. It has a parameter of String type array.

4.9.2.2 String[ ] get_info()

The getter function of the variable info. It has a return value of String type array.

4.10 User

4.10.1 Variables

NILL

4.10.2 Functions

4.10.2.1 ViewAbout_us()

Output is a html page with text related to the.

4.10.2.2 View_Gallery()
Output is a html page

4.10.2.3 View_FAQ()
Output is a html page to ask and search for questions.

4.10.2.3 View_Facilities()
Output is a html page to get description about the companies facilities.

4.10.2.4 Search(String s)
Takes input as a string extracts tags[] from the string and sends the request to the database to fetch the results output id the questions displayed on a html page.

4.11 **Request**

4.11.1 **Variables**

4.11.1.1 RequestDetails
Used to store the INput entered by user which is the request made to the company for a service to be provided

4.1.2 **Functions**

4.1.2.1 get_Request_Client_Details()
It has a return type of class Page where in all the details of the logged in user are the return values.

4.11.2.2 set_Request_Details(RequestDetails)
Here the parameter is a String which contains the input request String entered by the logged in user

4.11.2.3 get_Request_Details()

Here the return type is a String. It contains the input request String entered by some particular logged in user

5.0 Execution Architecture

Any modern web browser will suffice tested on Google chrome version 70.0.3 and firefox 63.0

5.1 Reuse and relationships to other products

NIL

6.0 Design decisions and tradeoffs

We have designed our website in such a way that there are 2 views. These are of the User/Client side and the other is of the Staff member side. We have made two views so as to ease things up for a staff member. For Example, a manager should only have to take care of the jobs in hand that have to be completed as well as the jobs requested. Thus in our class diagram the staff set of entities and the user entities are not connected as such. The only thing common is that they all have a function with the return type of Class Page.

Also, on the FAQ page, we have tried to design it in the most user-friendly way possible. We tried to design it the way the home page of facebook is. That design, we thought was a well liked one. Also having a search box in the design helps to not have to scroll through a whole lot of data in order to look for a specific thing. Thus, we also designed the process of adding a question in a similar way.

A question, along with its data, has to have a few word tags that describe it in a short,crisp way which is directly related to the main topics that that question is related to. This is similar to hashtags used on Instagram and Youtube.

On the Staff view, we realized that there is supposed to be a clear cut view for the staff members to view the jobs,tasks related to them and their status.

7.0 Pseudocode for components

7.1 Class User

Method1:**View_About_Us( )**
Input: hyperlink to /AboutUs
Output: /AboutUs page

1. View_About_Us(){
2. Click on “About Us” on navigaton bar
3. Redirect to /AboutUs page
4. }

Method2:**View_Contact_Us( )**
Input: hyperlink to /AboutUs
Output: /ContactUs page

1. View_Contact_Us(){
2. Click on “Contact Us” on navigation bar
3. Redirect to /ContactUs page
4. }

Method3:**View_Gallery( )**
Input: hyperlink to /Gallery
Output: /Gallery page

1. View_Gallery(){
2. Click on “Gallery” on navigation bar
3. Redirect to /Gallery

4. }

Method4:**View_FAQ( )**
Input: hyperlink to /FAQ
Output: /FAQ page

1. View_FAQ(){
2. Click on “FAQ” on navigation bar
3. Redirect to /FAQ page
4. }

Method5:**View_Facilities( )**
Input: hyperlink to /Facilities
Output: /Facilities page

1. View_Facilities(){
2. Click on “Facilities” on navigation bar
3. Redirect to /Facilities page
4. }

Method6:**Search(Tags[])**
Input: array of tags
Output: refresh the page with appropriate threads

```
1. Search(Tags[]){
2.   foreach(tag: tags){
3.    // use find method of Question model
4.    Question.find({tag: tag},function(err,result){
5.     if(!result)
6.     threads.concat(result)
7.    })
8.         Redirect to ask a question page with threads as data
9. }
```

7.2 Class Registered Users

Method1: **Login(String Username,String Password)**
Input: Username, Password
Output: Home page of login successful

```
1.  login(){
2.    passport.authenticate(local_strategy, function(err, user, info) {
3.      if (err)
4.            Internal error
5.      if (!user)
6.            redirect with flash message Wrong_crendentials
7.      req.logIn(user, function(err) {
8.        if (err)  let the express catch the error
9.        If no error redirect to home page
10.         });
11.       })
```

Method2: **change_password(String Old_password,String new_password)**
Input: Old password,new password
Output: redirect to home page and flash password saved status

```
1.  change_password(){
2.  // get user object from session
3.  req.user.changePassword(old_password,new_password,function(err,user)
    {
```

```
4.            if(err.name=="Incorrect Password Error")
5.              Old password mismatch flash mismatch
6.            Else
7.              Success redirect to home page with saved status
8.       })
9.  }
```

Method3: **Delete_account(String user_name)**
Input: User object reference
Output: redirect to home page

```
1.  Delete_account(req.user.username){
2.    // find a user with that username and remove from database
3.       User.deleteOne({username:req.user.username},function(err){
4.         if(err)
5.            return console.log(err);
6.       })
7.       Logout user this removes user data from session
8.       Redirect user to home page
9.  }
```

Method 5:**Add_Question(Q)**
Input: Name of question, question description , tags(5 maximum), hyperlink to /AddQ on clicking “Add Question”
Output: redirect to FAQ page with Question added to page

```
1.  AddQuestion(Q){
2.  Click on “Add Question”
3.  // on page /AddQ
4.  Enter all inputs
```

a.Q.set_Tags(t)

b.Q.set_Question_Name(qn)

c.Q.set_Question_Description(qd)

5. FAQ.set_Question(Q)
6. Click on submit
7. FAQ.get_Question(Q)

Method 6:**Answer_Question(A)**
Input:A string (Answer), hyperlink to /addA on clicking “Add Anwer”
Output: redirect to FAQ page with Answer added to that question

1. Answer_Question(A){
2. Click on “Add Answer”
3. // on page AddA
4. Enter input
5. Click “Add”
6. Input added as element to the collection of that particular question

Method7:**Add_Request(r)**
Input:hyperlink to page with form on clicking the button "Add",Request Details(i)

1.Add_Request(r){
2.//r is an object of type Request
4.String i = Input of request details by user
4.r.set_Request_Details(i)
5.}

7.3 Class Unregistered User

Method1:**Register(Registered_User user,String password)**
Input: An instance of registered user class and password
Output: save new user details to database

1. Register(User user,String password){
2. // pass user model to function register which is received from form

```
3.          User.register(new User({
4.          username:username,
5.          first_name:first_name,
6.          last_name:last_name,
7.          phone_number:phone_number}),
8.          password,function(err){
9.          if(err.name=="User Exists Error"){
10.         If user already exists render same page with information Try
11.         different username
12.         }
13.        // if everything goes well login user and redirect to home page
14.         passport.authenticate("local")(req,res,function(err){
15.                 res.redirect("/");
16.     })
17.         })
```

7.4 Class FAQ

Method1: **set_Question(q)**

Input: An object of class Question
Output:Nil

1. set_Question(q){
2. Create variable of type faqSchema
   a.q.get_Tags(t)
   b.q.get_Question_Name(qn)
   c.q.get_Question_Description(qd)
3. Add variable to collection

Method2: **get_Question(q)**

Input: Nil

Output:array of Questions

1. get_Question(q){
2. Redirect to /FAQ
3. if(!error){
4. response.render("/faq",{faqSchema:allFaq});
5. }

7.5 Class Question

Method1: **set_Tags(t)**

Input: Upto 5 strings(i.e. tags)
Output:Nil

1. set_Tags(t){
2. Initialize var i=0
3. For loop(i<5)
    a. t[i]=Enter input
    b. If user does not click submit
        i. Continue
    C. else
        I. break
4. }

Method2: **get_Tags()**

Input: Nil
Output:Nil

1. get_Tags(t){
2. return t
3. }

Method3: **set_Question_Name(qn)**

Input: Question Name
Output:Nil

1. set_Question_Name(qn){
2. qn= Enter Input
3. }

Method4: **get_Question_Name()**

Input: Nil
Output:Nil

1. get_Question_Name(){
2. return qn
3. }

Method5: **set_Question_Description(qd)**

Input: Question Description
Output:Nil

4. set_Question_Description(qd){
5. qd= Enter Input
6. }

Method6: **get_Question_Description()**

Input: Nil
Output:Nil

4. get_Question_Description(){
5. return qd
6. }

7.6 Class Page

Method1: **set_info(i)**
Input: a String array
Output:Nil

1. set_info(i){
2. i=Admin enters all information
3. }

Method1: **get_info()**
Input: Nil
Output:a String array

1. get_info(){
2. return i
3. }

7.7 Class Staff

Method1:**get_Fname()**
Input: Nil
Output:Nil

1. get_Fname(){
2. return FirstName
3. }

Method2:**get_Lname()**
Input: Nil
Output:Nil

1. get_Lname(){
2. return LastName
3. }

Method3:**get_Email()**
Input: Nil
Output:Nil

1. get_Email(){
2. return Email
3. }

Method4:**get_Username()**
Input: Nil
Output:Nil

get_Username(){

return un

}

Method5:**set_Username()**
Input: username
Output:Nil

1. set_Username(un){
2. un = Name entered by user
3. }

Method6:**set_Password(p)**
Input: password
Output:Nil

1. set_Password(){
2. return p
3. }

Method7:**Add_Job(j)**

Input:Object of Job class(inputs being job name,tasks(name and details) and client(name,Phone Number and Email))
Output:Nil

```
1.Add_Job(j){
2.j=User enters input
        a.set_Job_Name(n)
        b.set_Job_tasks(t)
        c.set_Job_Client(c)
3.}
```

Method8:**Check_Job_Request(Request r)**
Input:Hyperlink to a page displaying the request details
Output:Page showing details of the request

```
1.Check_Job_Request(r){
2.
        a.r.get_Request_Client_Details()
        b.r.get_Request_Details()
3.}
```

7.8 Class Job

Method1: **set_Job_Name(n)**

Input:Job name(n)
Output:Nil

```
1. set_Job_Name(n){
2. n =  Input entered by staff member
3. }
```

Method2: **get_Job_Name()**

Input:Nil
Output:Nil

```
1. get_Job_Name(){
2. return n
3. }
```

Method3: **set_Job_tasks(t)**

Input:Job name(n)
Output:Nil

```
1. set_Job_tasks(t){
2.   while(click Add){
3.         t[i] = task entered by user
        a. t[i].set_task_Name(n)
        b. t[i].set_task_details(d)
4.     }
5. }
```

Method4: **get_Job_tasks()**

Input:Nil

Output:Nil

1. get_Job_tasks(){
2. return t
3. }

Method5: **set_Job_Client()**

Input:All Client Details
Output:Nil

1. set_Job_Client(client){
2. client = Input entered by staff member
    a. client.set_Name(n)
    b. client.set_Phone_No(p)
    c. client.set_Email(e)
3. }

Method6: **get_Job_Client()**

Input:Nil
Output:Nil

1. get_Job_Client(){
2. return c
3. }

7.9 Class Client

Method1: **set_Name(n)**

Input:Client's name(n)
Output:Nil

1. set_Name(n){

2. n = Input entered by staff member
3. }

Method2: **get_Name()**

Input:Nil
Output:Nil

1. get_Name(){
2. return n
3. }

Method3: **set_Phone_No(p)**

Input:Client's Phone Number
Output:Nil

1. set_Phone_No(p){
2. p=Input entered by staff member
3. }

Method4: **get_Phone_No()**

Input:Nil
Output:Nil

1. get_Job_tasks(){
2. return p
3. }

Method5: **set_Email()**

Input:Client's Email
Output:Nil

1. Set_Email(e){
2. e=Input entered by user
3. }

Method6: **get_Email()**

Input:Nil
Output:Nil

1. get_Email(){
2. return e
3. }

7.10 Class Task

Method1: **set__Task_Name(n)**

Input:Task's name(n)
Output:Nil

4. set__Task_Name(n){
5. n = Input entered by staff member
6. }

Method2: **get__Task_Name()**

Input:Nil
Output:Nil

4. get_Task_Name(){
5. return n
6. }

Method3: **set_task_details(d)**

Input:Task's details
Output:Nil

4. set_task_details(d){
5. d =Input entered by staff member
6. }

Method4: **get_task_details()**

Input:Nil
Output:Nil

4. get_task_detailsd){
5. return d
6. }

7.11 Class Request

Method1: **set_Request_Client_Details()**

Input:Nil

Output:Nil

1.get_Request_Client_Details(){
2. //The details of the user that had logged in when writing the request
// Let it's object be called ru
a.ru.get_Fname()

b.ru.get_Lname()

c.ru.get_Email()

3.}

Method2: **set_Request_Details()**

Input:

Output:Nil

1.set_Request_Details(r){

2. r=input by user

3.}

Method3: **get_Request_Details()**

Input:Nil

Output:Nil

1.get_Request_Details(){

2. return r

3.}