Software Design Specifications

"Code Utility Web App"

Guidance by -
Prof. Amit Kumar sir

Project by -
Ganesh Singh - U101115FCS087 - S5
Sangamesh Kotalwar - U101115FCS210 - S5
Shivam Ratnakar - U101115FCS147 - S5
Shivangi Tak - U101115FCS213 - S5

1. Introduction

1.1 Purpose of the SDS

The purpose of this document is to provide a high-level design framework around which we'd build our Code Utility Web Application. The document also provides a list of requirements against which we'd be testing our final project and determine whether we were able to successfully implement the application according to design.

1.2 Scope of the project

In scope:

1. User will be able to find all the assistance he needs regarding the entered coding language on any entity or operation and also save (if registered) his experiences as code snippets for future reuse.
2. Creating and managing a personal account of his own.
3. The registered user will be able to store all his learning and building experiences at one place along with their description and retrieve them from anywhere in future.
4. (if registered) User will be able to share his learning and building experiences with any other existing user he wants.
5. The code snippets stored are in sorted order according to the tags and category so that the searching time is less.
6. User will be able to specify whether the snippet he is entering is to be kept private or public.

Out of scope:

1. Any user cannot know about the private code snippets of the other users and request for it.
2. The entered code snippet or description cannot be checked for correctness.
3. Users can't rate one another's public data for better references.
4. User won't be able to get the detailed information/documentation about the entity searched by him/her.

1.3 Overview

The *system architecture description* section is the main focus of *Section 2* of this document. It provides an overview of the system's major components and architecture, as well as specifications on the interaction between the system and the user.

The *Detailed description of components* section will be the main focus of *Section 3* of this document. It will describe lower-level classes, components, and functions, as well as the interaction between these internal components.

In *Section 4*, we detail the steps that we are taking to focus on code reuse in this software, and we explain our motives for doing so.

Section 5 lists the major decisions we had to make when designing our system, and why we made the choices we did.

The *Pseudo code* section will provide pseudo code in order to clarify the intended operation of certain components. This is beyond the scope of this version of the SDS.

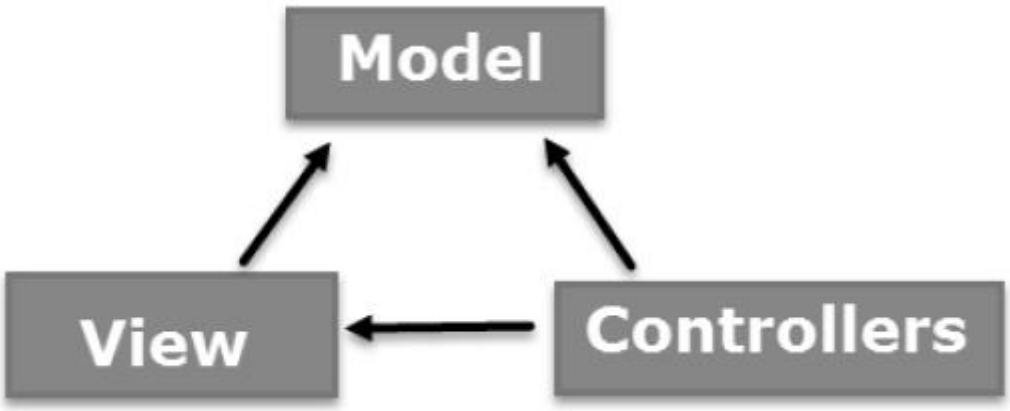
2. System Architecture Description

2.1 Overview of modules / components

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components is built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

2.2 Structure and relationships

Following are the components of MVC:



MODEL

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

VIEW

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

CONTROLLER

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

2.3 User Interface Functionalities

The user interface of the Project Tracker will be divided into four main sections: Search, Insert, Share and Delete. All these functionalities will be accessible to the user depending in the category in which he lies: Guest User or Registered User.
Guest User would be able to access **only** the search functionality of the system's pre-defined database.
Registered User will be able to use **all** the functionalities listed above.

SEARCH
The search boxes would be available along with the filters which would help the user specify which programming entity he/she wants to search. For Example : Packages, Classes, Methods, Data Types, Tasks, Errors etc.
The user would also have the option to specify the scope of the search, public or private. By default, for the *guest user* the search would be made in only the system's database (the option of choosing the scope would not be available to the guest user) and for the *registered user*, same would be made in the public database which would include the system's predefined data as well as other user's public data.

INSERT (only accessible to registered users)
The main aim of building this web application is to allow a programmer to store the gained experiences along the learning journey. The insert functionality will allow any registered user to store the code snippets along with his defined description of any one of the entities mentioned in the drop down menu for filters.

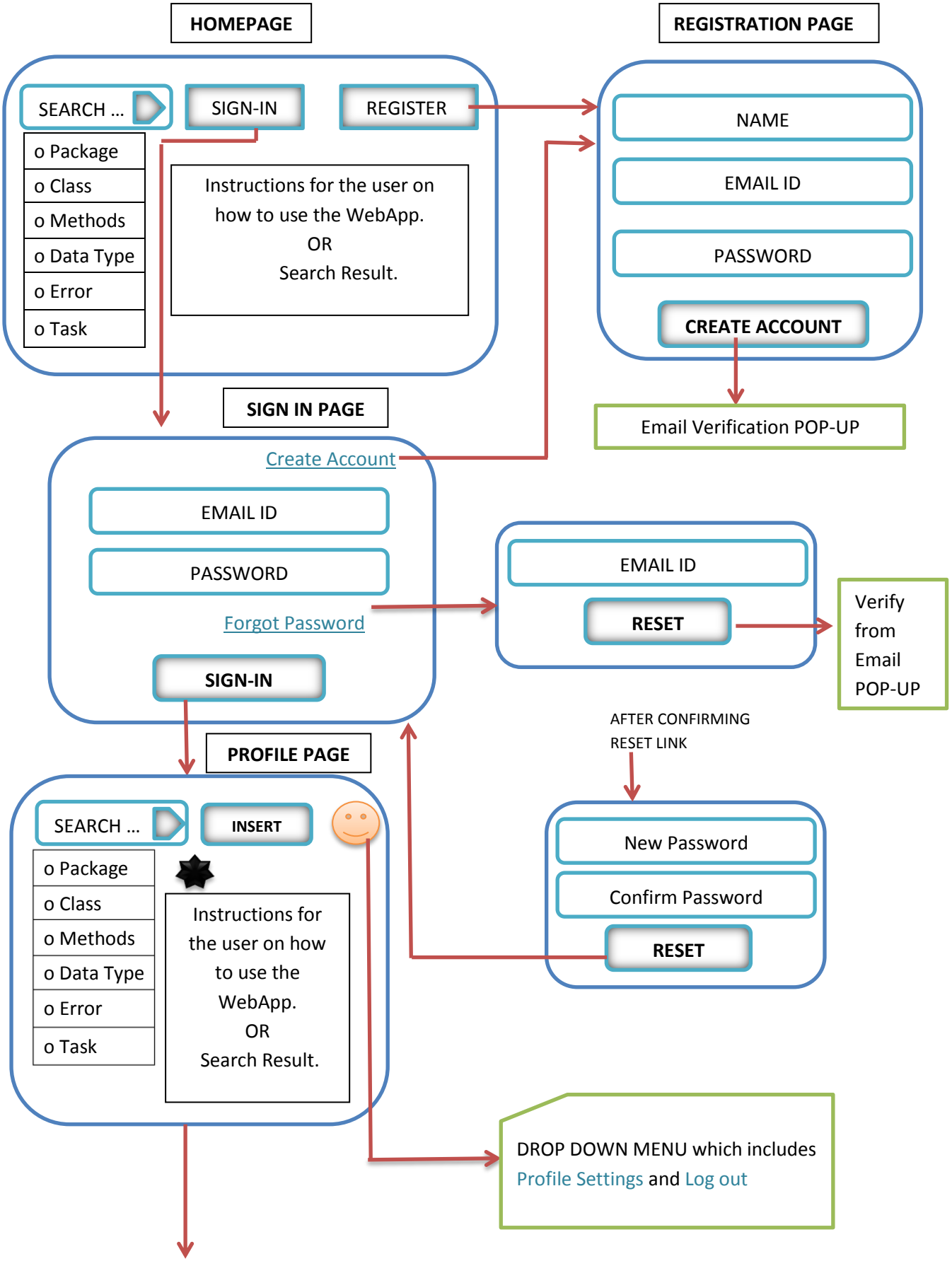
ENTITY (Package/Class/Method/Data Type/ Errors/Tasks etc.)
SCOPE (Private/Public)
TITLE (Tag of the snippet)
CODE SNIPPET (Lines of code)
DESCRIPTION (User's own interpretation of the code)

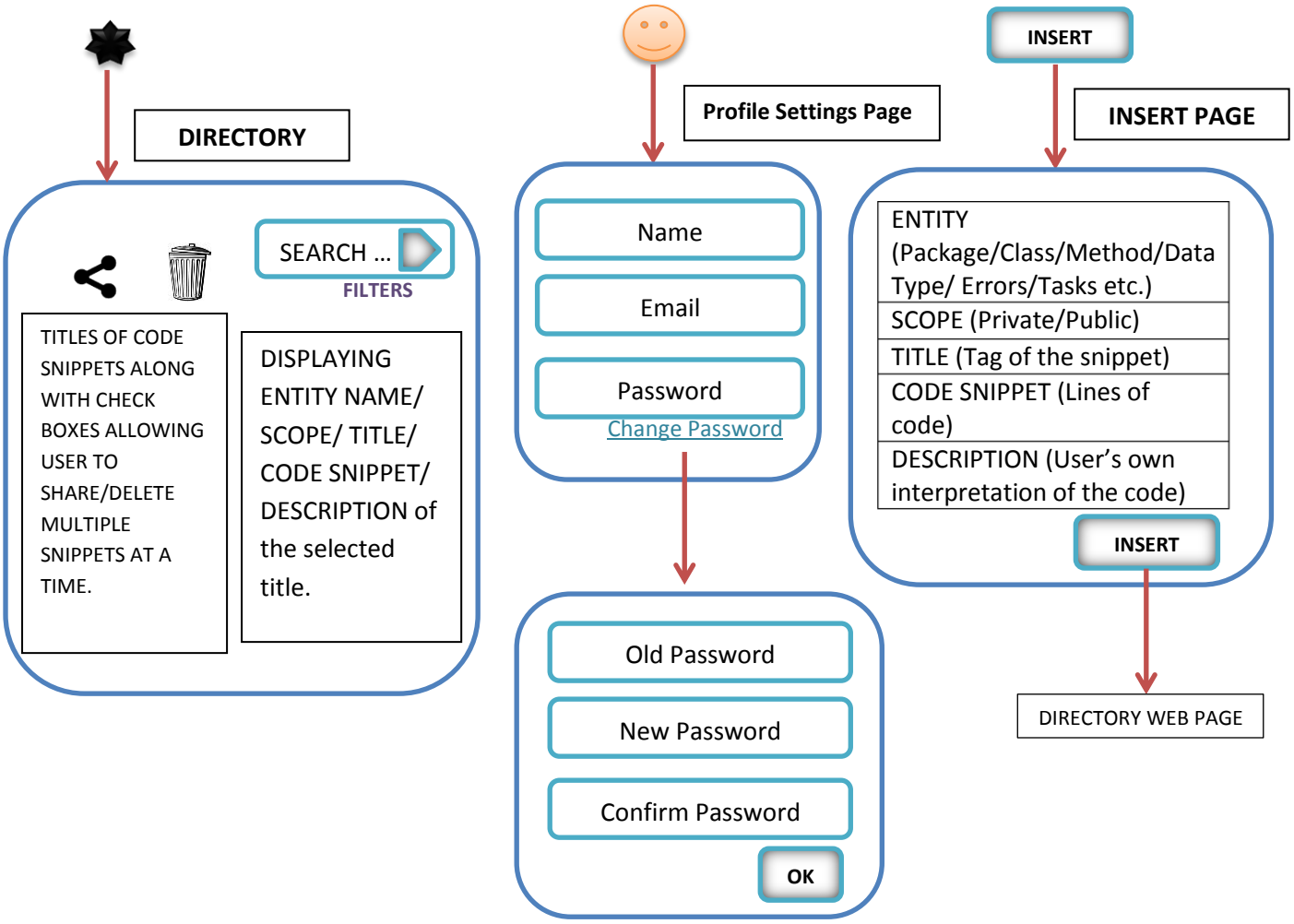
SHARE (only accessible to registered users)
User will also be able to share his code snippets(present in the user's own directory) with any of the other registered users using the share button present in his directory. The user can either choose to make the required snippet public which will thus enable him to share it to all the existing registered user or he/she can even directly share the snippets by specifying the recipient's unique user id.

DELETE (only accessible to registered users)
User will also be able to delete his code snippets(present in the user's own directory) using the delete button present alongside the code snippets.

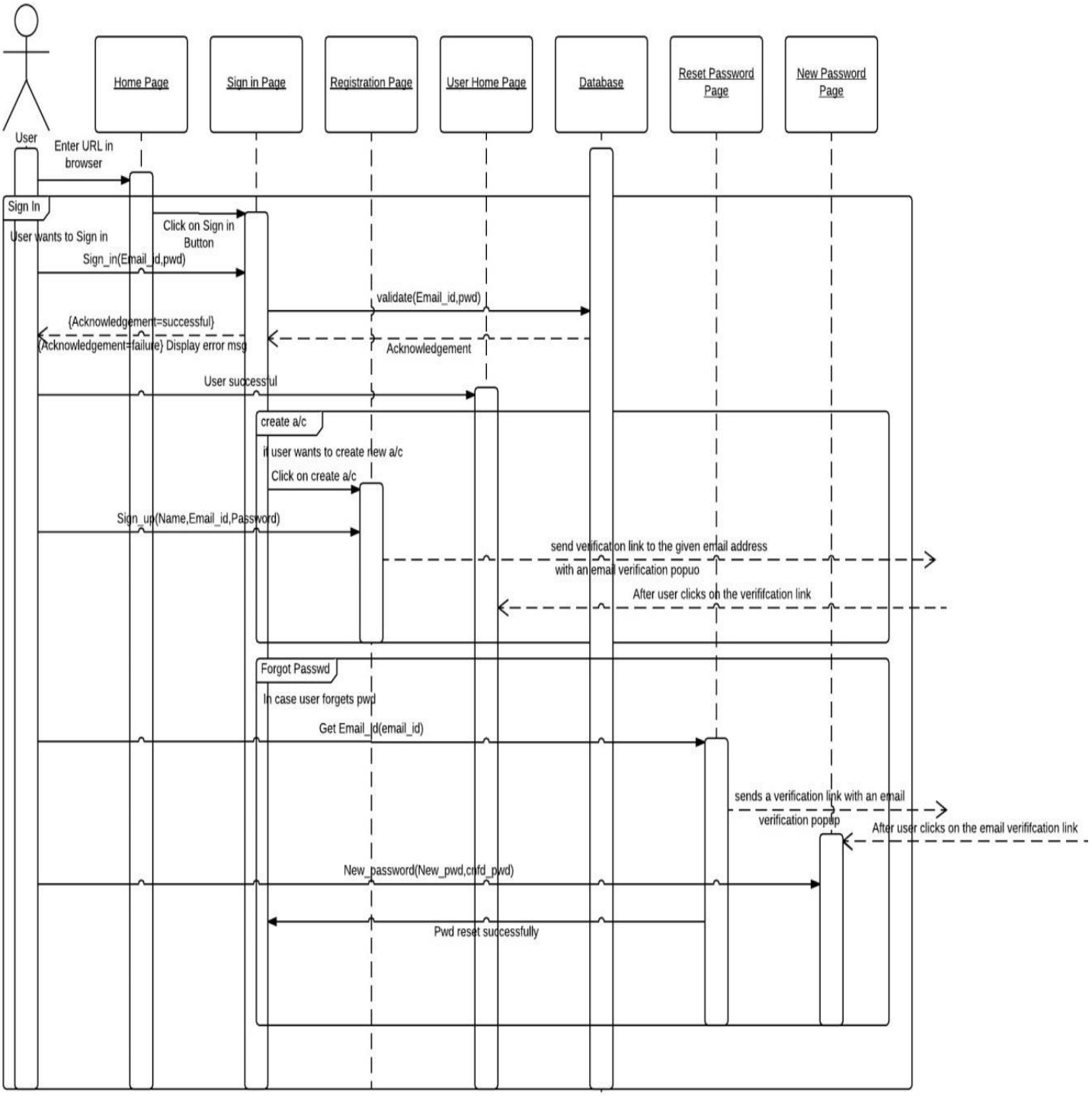
3. Logical Design

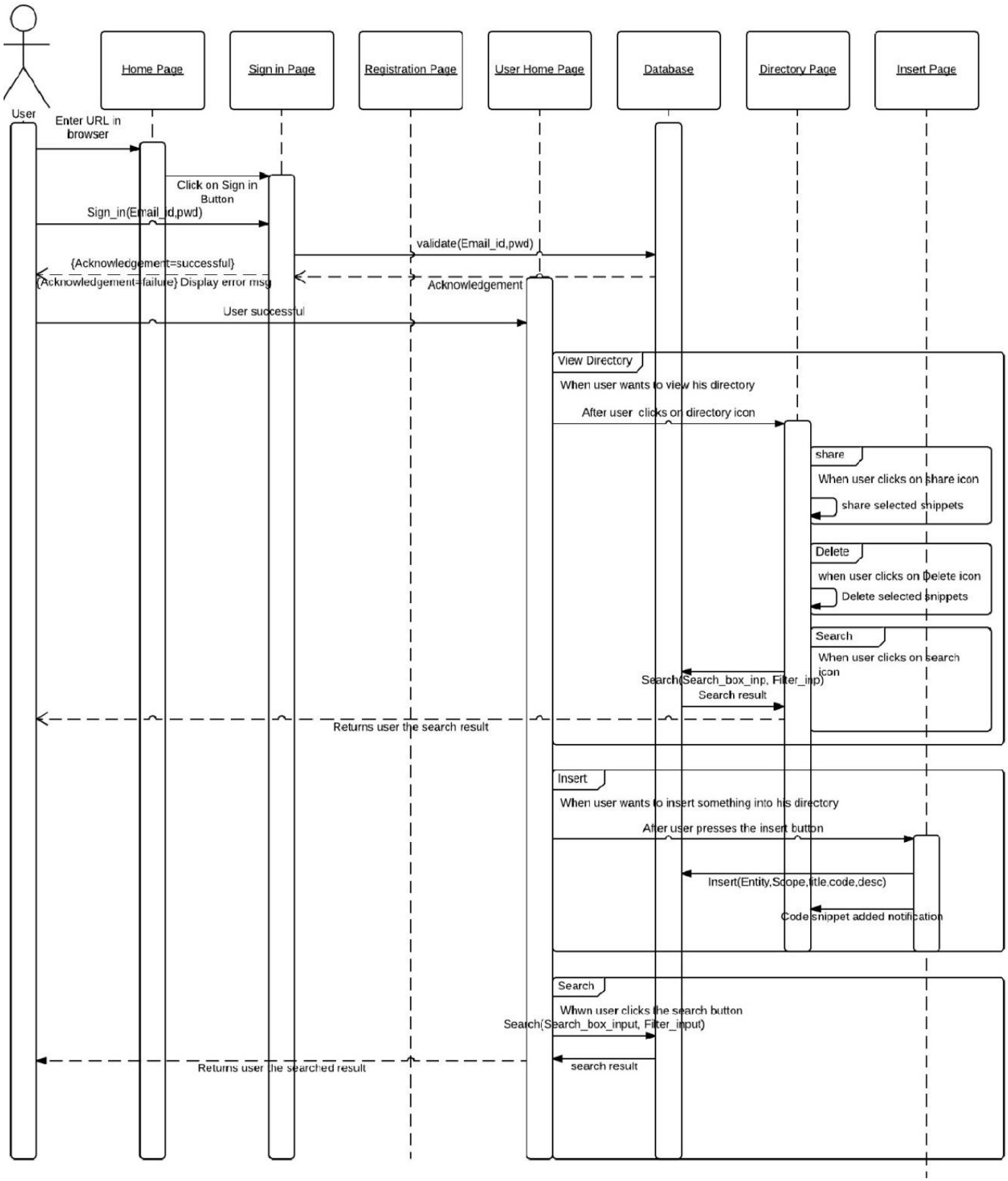
3.1 State Diagram

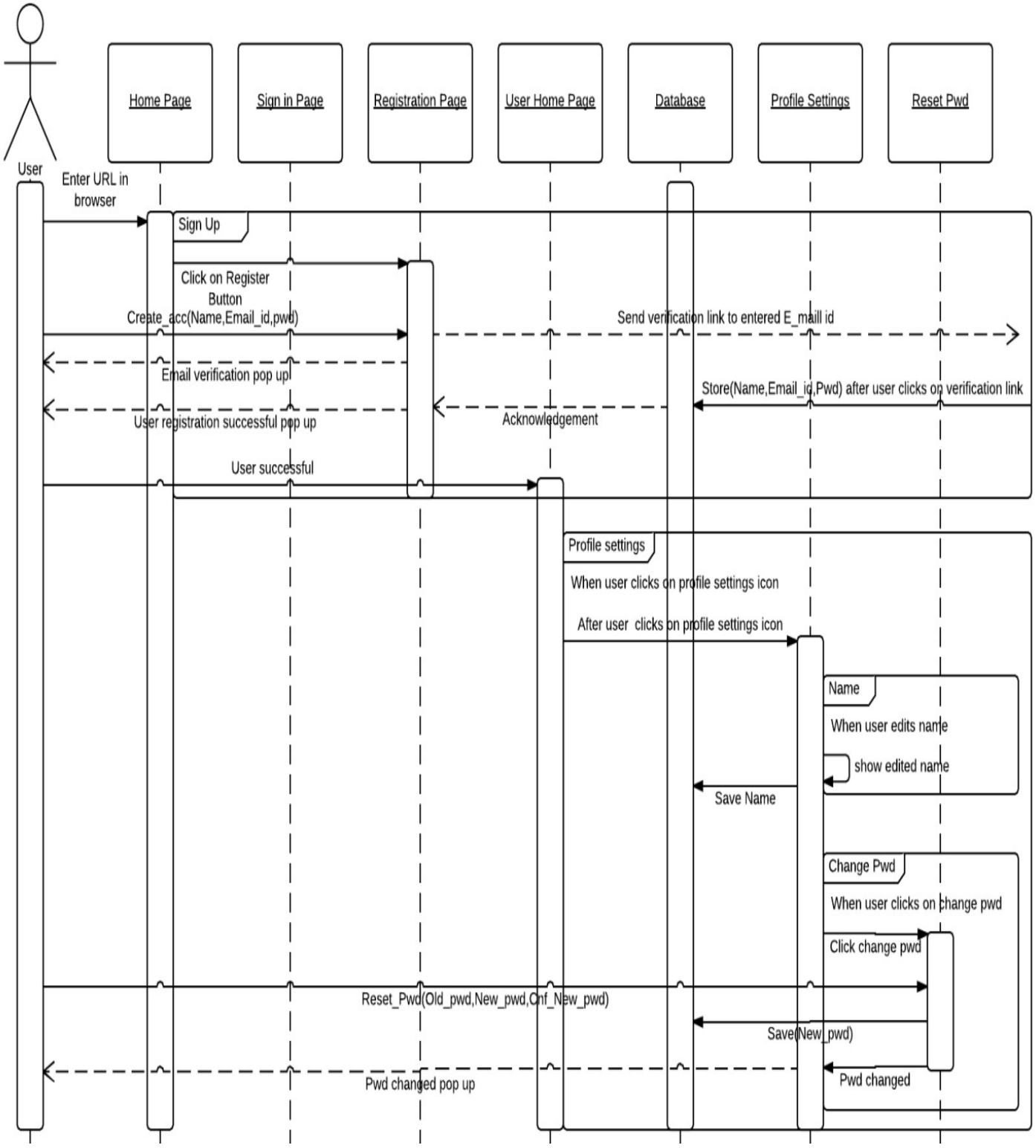



DIRECTORY
SEARCH ...
FILTERS
TITLES OF CODE SNIPPETS ALONG WITH CHECK BOXES ALLOWING USER TO SHARE/DELETE MULTIPLE SNIPPETS AT A TIME.
DISPLAYING ENTITY NAME/ SCOPE/ TITLE/ CODE SNIPPET/ DESCRIPTION of the selected title.
Profile Settings Page
Name
Email
Password
Change Password
Old Password
New Password
Confirm Password
OK
INSERT
INSERT PAGE
ENTITY (Package/Class/Method/Data Type/ Errors/Tasks etc.)
SCOPE (Private/Public)
TITLE (Tag of the snippet)
CODE SNIPPET (Lines of code)
DESCRIPTION (User's own interpretation of the code)
INSERT
DIRECTORY WEB PAGE

3.2 Sequence Diagrams




User
Home Page
Sign in Page
Registration Page
User Home Page
Database
Directory Page
Insert Page
Enter URL in browser
Click on Sign in Button
Sign_in(Email_id,pwd)
validate(Email_id,pwd)
{Acknowledgement=successful}
{Acknowledgement=failure} Display error msg
Acknowledgement
User successful
View Directory
When user wants to view his directory
After user clicks on directory icon
share
When user clicks on share icon
share selected shippets
Delete
when user clicks on Delete icon
Delete selected snippets
Search
When user clicks on search icon
Search(Search_box_inp, Filter_inp)
Search result
Returns user the search result
Insert
When user wants to insert something into his directory
After user presses the insert button
Insert(Entity,Scope,title,code,desc)
Code snippet added notification
Search
Whwn user clicks the search button
Search(Search_box_input, Filter_input)
search result
Returns user the searched result


User
Home Page
Sign in Page
Registration Page
User Home Page
Database
Profile Settings
Reset Pwd
Enter URL in browser
Sign Up
Click on Register Button
Create_acc(Name,Email_id,pwd)
Send verification link to entered E_maill id
Email verification pop up
Store(Name,Email_id,Pwd) after user clicks on verification link
User registration successful pop up
Acknowledgement
User successful
Profile settings
When user clicks on profile settings icon
After user clicks on profile settings icon
Name
When user edits name
show edited name
Save Name
Change Pwd
When user clicks on change pwd
Click change pwd
Reset_Pwd(Old_pwd,New_pwd,Cnf_New_pwd)
Save(New_pwd)
Pwd changed
Pwd changed pop up

3.3 Component Overview

MVC i.e. Model View Controller is one of the most widely used frameworks to design any web app. It separates the whole web app into certain sections which are easy to understand, develop and maintain or in other words it helps structure the code-base.

Let's talk about each section separately with respect to our web app:

3.4 Model

3.5 View

3.6 Controller

3.7 Database

3.4 Model

PURPOSE	Model is basically responsible for managing data of the web application. It replies to the instructions from the controller which is actually requests from the view (user).
SUBORDINATES	Model functions are handled by MySQL
FUNCTIONS	CREATE TABLE INSERT UPDATE SELECT DELETE SET WHERE AND OR

3.5 VIEW

PURPOSE	Presentation of data in a particular format, triggered by a controller's decision to present the data. In other words VIEW is responsible for what is visible to the user on the user interface.		
SUBORDINATES	HTML, CSS, JavaScript handle the VIEW's responsibility		
FUNCTIONS	HTML tags- <p> <a> <form> <input> <textarea> <button> <nav> <header> <footer> <script> <onclick>	CSS functions- Color Background Text Fonts Table Animation Transform Transition Flexible box Borders	JavaScript functions validateForm() toggle() mouseEnter() mouseLeave() scroll()

3.6 CONTROLLER

PURPOSE	**The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the operation that modifies the state of the data model.**
SUBORDINATES	Controller functions are handled by PHP
FUNCTIONS	mysqli_query () md5 () mysqli_connect () session_start () mysqli_real_escape_string () SEARCH () setcookie () mysqli_num_rows () isset () array_key_exists ()

3.7 DATABASE

PURPOSE	The whole MVC framework works upon and relies on data that is stored the database.
SUBORDINATES	The whole database is divided into tables in order to store data.
FUNCTIONS	PROFILE TABLE • ID • Username • Password • Email USER TABLE • ID • Title • Code • Description • Access • TypeOfCode PUBLIC DATA TABLE • ID • Title • Code • Description • TypeOfCode

4. Reuse and relationship to other products

4.1 Execution Architecture

This application will require any appropriate web browser that supports HTML 5, CSS and JavaScript.

4.2 Reusability

We would be using PHP-Swift for verification tasks and certain API's already available in the web development community in order to avoid writing code snippets from scratch.

4.3 Tools

From the beginning we set a goal to make use of any existing code to avoid wasting time duplicating other's work. We also decided to use open source or freeware solutions wherever possible. Since we are creating a web application, it is possible to use all the web development tools available through the development community. It was obvious for us to go with HTML, CSS and Java Script for the front-end as it is ubiquitous in almost all the web applications. As database manipulation is one of the key features of our WebApp, we chose PHP as our back-end language along with Microsoft SQL Server 2014.

5. Design and Decision Tradeoffs

5.1 Front-end and Back-end Decisions

Since we are creating a web application, it is possible to use all the web development tools available through the development community. It was obvious for us to go with HTML, CSS and Java Script for the front-end as it is ubiquitous in almost all the web applications. As database manipulation is one of the key features of our WebApp, we chose PHP as our back-end language along with Microsoft SQL Server 2014 instead of JavaScript because of the lack of experience in the domain of JS across the team.

5.2 Abandoned Ideas

In the early stages of design, we chose to restrict the WebApp to only registered users, but later on extended it by allowing Guest Users to carry out the search in system's database.
Keeping in mind the time span we have to develop the project, we left out the idea of ranking system which would allow other users to rate other user's public data for better references.

6. Pseudo code for components

```
if (New (unregistered) User)
{
        SignUp (email, password)
        {
                if(credentials==appropriate)
                {
                        Connect_database (credentials)
                        {
                                store_info ();
                        }
                        Send_email_verification ();
                }
                else
                        Reenter_credentials ();
        }
        Search (search_box_inp, filter_inp)
        {
                connect _public_database(search_box_inp, filter_inp);
                print (appropriate_result);
```

```
        }

}
else
{
        LogIn (email, password)
        {
                if(credentials==correct)
                {
                        SESSION_start();
                        {
                                Search (search_box_inp, filter_inp)
                                {
                                        connect _user_database(search_box_inp, filter_inp);
                                        connect _public_database(search_box_inp, filter_inp);
                                        print (appropriate_result);
                                }
                                Insert(Title, Entity_type, code, desc, access)
                                {
                                        connect_user_database (Title, Ent_type, code, desc, access);
                                        if (access==public)
                                        {
                                        connect_pub_database (Title, Ent_type, code, desc, access);
                                        }
                                        insert_into_user_table();
                                }
                                Share (receiver_email, code_id)
                                {
                                        connect_user_database (receiver_email, code_id);
                                        insert_into_receiveruser_table (code_info);
                                }
                                Delete (code_id)
                                {
                                        connect_user_database (code_id);
                                        remove_from_user_table (code_info);
                                }
                                if (logout==true)
                                        unset(SESSION);
                        }
                }
                else
                        Reenter_credentials ();
        }
}
```