# Project Report
# COE - 320
## “CollNet”

Akanshi Gupta *(216/CO/12)*
Ayush Sharma *(250/CO/12)*
Chitrasoma Singh *(254/CO/12)*
Divjot Singh *(262/CO/12)*

# CONTENTS

# 1. INTRODUCTION

The most important and influential part for the success of any organisation is the communication channel through which the current students, alumni and new aspirants can connect to each other. In today's world, where social networking sites create separate groups for individual group of students belonging to a particular class, this project comes up with a centralised platform where not only the students of a single organisation can connect to each other at a single platform but students of any organisation across the world can connect and get complete information about any student across the globe. Take all the socializing you typically do as a college student, crank it up a few notches, and you have networking of the sort that can truly help you.

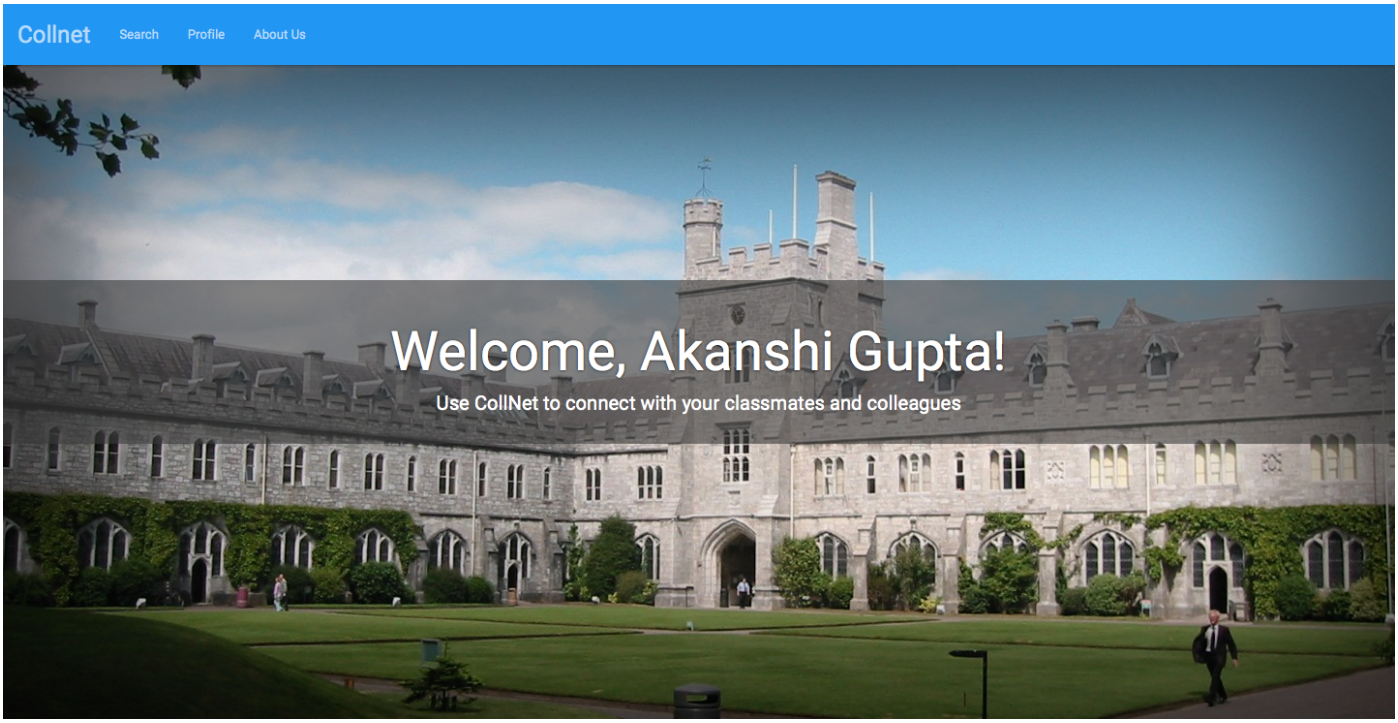
CollNet is all about networking and connections.

- Students engaging with their classmates.
- Alumni engaging with their classmates.
- Students engaging with their alumni.

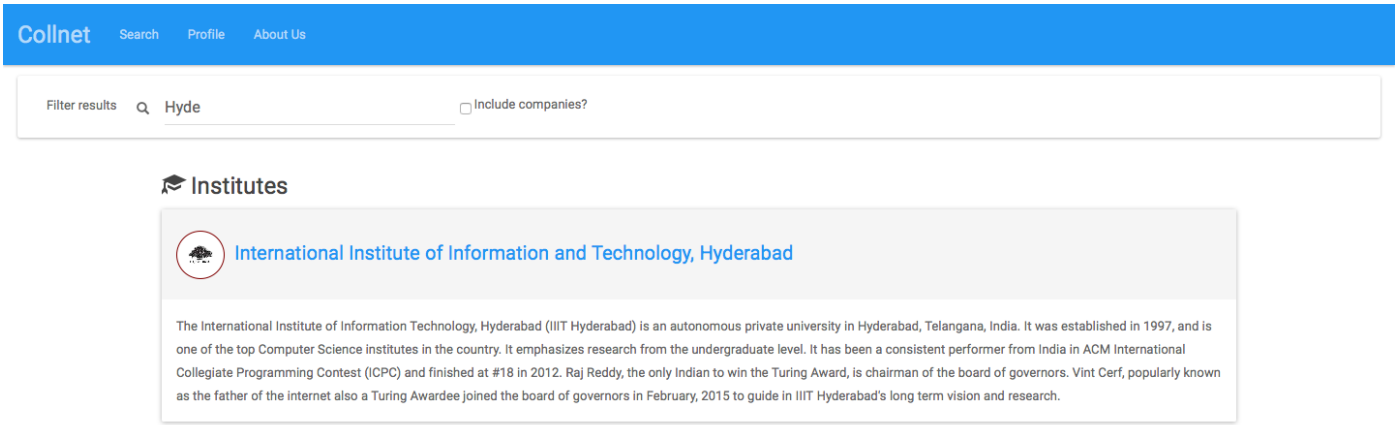
Once registered in a college with our app, the student doesn't have to worry about numerous groups it need to join to get correct information. It becomes easy for him/her to contact other students from his section / department or even college. Further posts at each level allow a better medium of communication than what present social networking sites offer. We can query the data on basis of student's info like college, work profile etc. One can even find colleges which have students working in, say Facebook. This way it helps a fresher to better select a college to join by seeing such statistics.

# 2. SCREENSHOTS

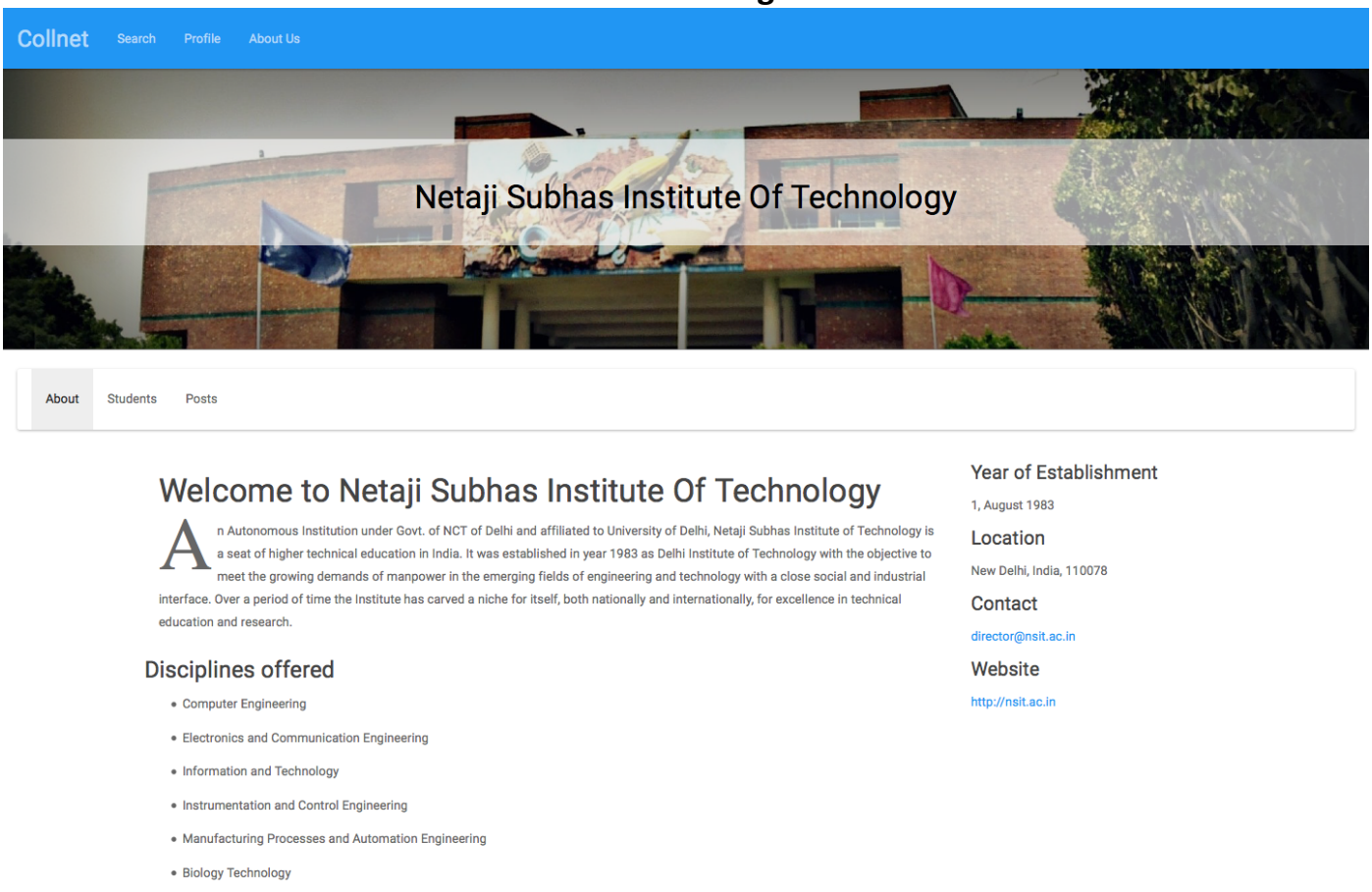
## Main Page



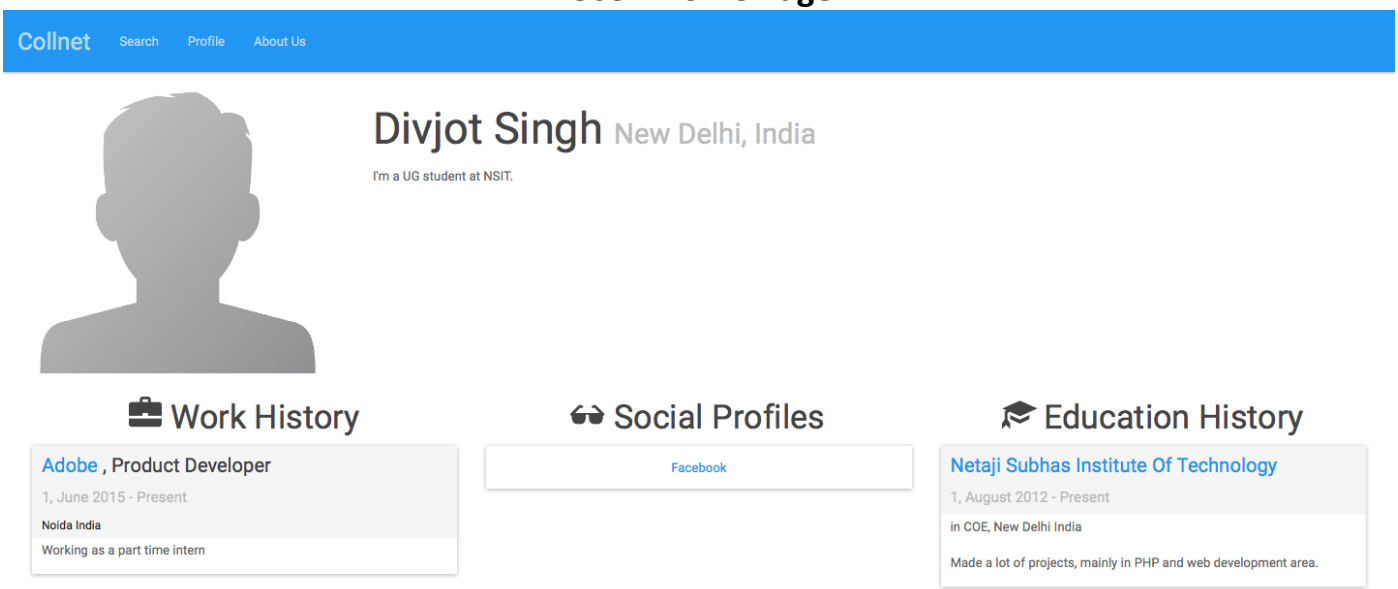

## Search Page



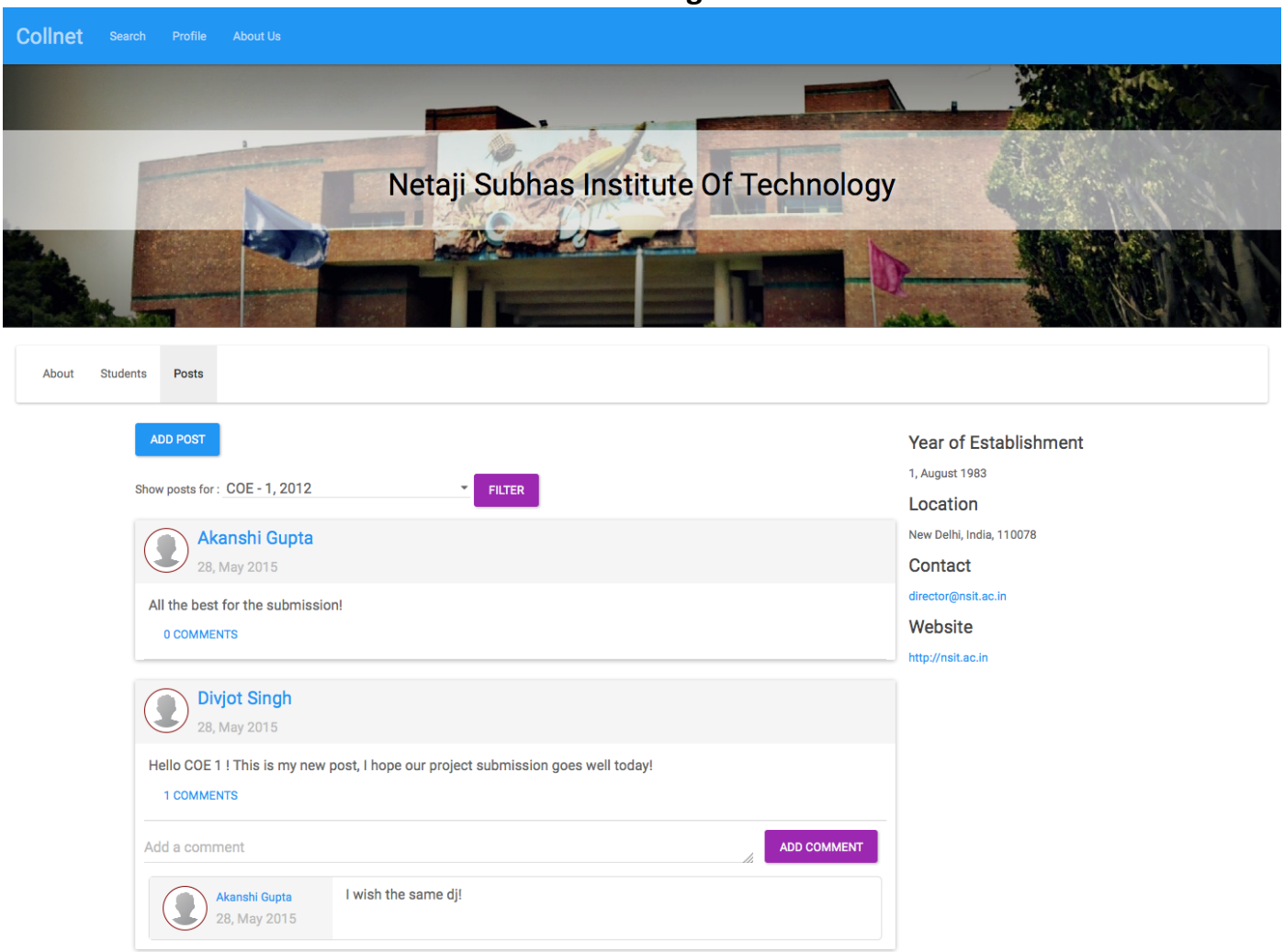
## Institute Page



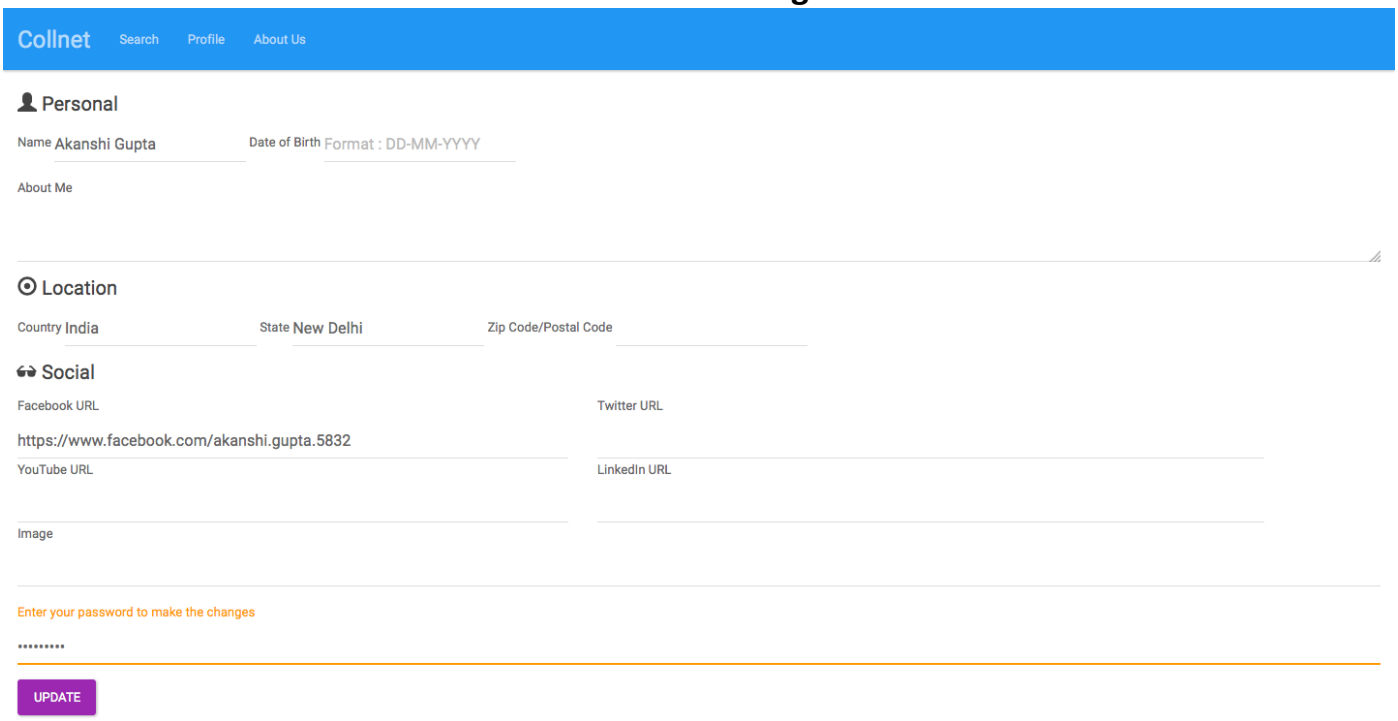

## User Profile Page



## Posts Page




## Edit Profile Page



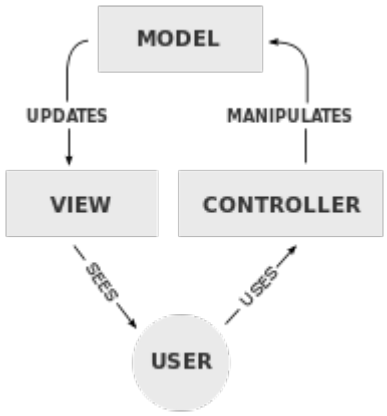
# 3. TECHNICAL DETAILS

CollNet is a single page application. AngularJS is used with JavaScript, CSS and HTML as client side languages with BootStrap layout framework. MongoDB is used for database management with Object Oriented Programming in PHP as for server side scripting. The entire project has been constructed following the MVC architecture for scalability and modularity.

## ➢ 2a. MVC ARCHITECTURE

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

**Components**




The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface. The model directly manages the data, logic and rules of the application. A view can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts input and converts it to commands for the model or view.

**Interactions**

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

- A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).
- A model stores data that is retrieved by the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller.
- A view requests information from the model that it uses to generate an output representation to the user

## ➢ 2b. PHP (Hypertext Preprocessor)

PHP is a general-purpose scripting language that is especially suited to server-side web development, in which case PHP generally runs on a web server. Any PHP code in a requested file is executed by the PHP runtime, usually to create dynamic web page content or dynamic images used on websites or elsewhere. It can also be used for command-line scripting and client-side graphical user interface (GUI) applications. PHP can be deployed on most web servers, many operating systems and platforms, and can be used with many relational database management systems (RDBMS). Most web hosting providers support PHP for use by their clients. It is available free of charge, and the PHP Group provides the complete source code for users to build, customize and extend for their own use.

PHP acts primarily as a filter, taking input from a file or stream containing text and/or PHP instructions and outputting another stream of data. Most commonly the output will be HTML, although it could be JSON, XML or binary data such as image or audio formats.

Originally designed to create dynamic web pages, PHP now focuses mainly on server-side scripting, and it is similar to other server-side scripting languages that provide dynamic content from a web server to a client, such as Microsoft's ASP.NET, Sun Microsystems' JavaServer Pages, and mod_perl. PHP has also attracted the development of many software frameworks that provide building blocks and a design structure to promote rapid application development (RAD).

## ➢ 2c. ANGULAR JS

In software development, **AngularJS** is an open-source web application framework maintained by Google and by a community of individual developers and corporations to address many of the challenges encountered in developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) architecture, along with components commonly used in rich Internet applications.

The AngularJS library works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

AngularJS is built around the belief that declarative programming should be used for building user interfaces and connecting software components, while imperative programming is better suited to defining an application's business logic. The framework adapts and extends traditional HTML to present dynamic content through two-way data-binding that allows for the automatic synchronization of models and views. As a result, AngularJS de-emphasizes DOM manipulation with the goal of improving testability and performance.

AngularJS's design goals include:

- to decouple DOM manipulation from application logic. The difficulty of this is dramatically affected by the way the code is structured.
- to decouple the client side of an application from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
- to provide structure for the journey of building an application: from designing the UI, through writing the business logic, to testing.

Angular implements the MVC pattern to separate presentation, data, and logic components. Using dependency injection, Angular brings traditionally server-side services, such as view-dependent controllers, to client-side web applications. Consequently, much of the burden on the server can be reduced.

The tasks performed by the AngularJS bootstrapper occur in three phases after the DOM has been loaded:

1. Creation of a new Injector
2. Compilation of the directives that decorate the DOM
3. Linking of all directives to scope

AngularJS directives allow the developer to specify custom and reusable HTML-like elements and attributes that define data bindings and the behavior of presentation components. Some of the most commonly used directives are:

*ng-app*

Declares the root element of an AngularJS application, under which directives can be used to declare bindings and define behavior.

ng-controller

Specifies a JavaScript controller class that evaluates HTML expressions.

*ng-repeat*

Instantiate an element once per item from a collection.

*ng-view*

The base directive responsible for handling routes that resolve JSON before rendering templates driven by specified controllers.

*ng-if*

Basic if statement directive that allow to show the following element if the conditions are true. When the condition is false, the element is removed from the DOM. When true, a clone of the compiled element is re-inserted

AngularJS' two-way data binding is its most notable feature, and it reduces the amount of code written by relieving the server backend of templating responsibilities. Instead, templates are rendered in plain HTML according to data contained in a scope defined in the model. The $scope service in Angular detects changes to the model section and modifies HTML expressions in the view via a controller. Likewise, any alterations to the view are reflected in the model. This circumvents the need to actively manipulate the DOM and encourages bootstrapping and rapid prototyping of web applications. AngularJS detects changes in models by comparing the current values with values stored earlier in a process of dirty-checking, unlike Ember.js and Backbone.js that trigger listeners when the model values are changed.

## ➢ 2d. MONGODB

MongoDB is a document database that provides high performance, high availability, and easy scalability.

- Document Database
  - Documents (objects) map nicely to programming language data types.
  - Embedded documents and arrays reduce need for joins.
  - Dynamic schema makes polymorphism easier.
- High Performance
  - Embedding makes reads and writes fast.
  - Indexes can include keys from embedded documents and arrays.
  - Optional streaming writes (no acknowledgments).
- High Availability
  - Replicated servers with automatic master failover.
- Easy Scalability
  - Automatic sharding distributes collection data across machines.
  - Eventually-consistent reads can be distributed over replicated servers.
- Advanced Operations
  - With MongoDB Management Service (MMS) MongoDB supports a complete backup solution and full deployment monitoring

A MongoDB deployment hosts a number of databases. A database holds a set of collections. A collection holds a set of documents. A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

**Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

**Collection**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

**Document**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Below given table shows the relationship of RDBMS terminology with MongoDB

| RDBMS | MongoDB |
|---|---|
| Database | Database |
| Table | Collection |
| Row | Document |
| Column | Field |
| Table Join | Embedded Documents |

To use mongodb with php you need to use mongodb php driver.

**Make a connection and Select a database**

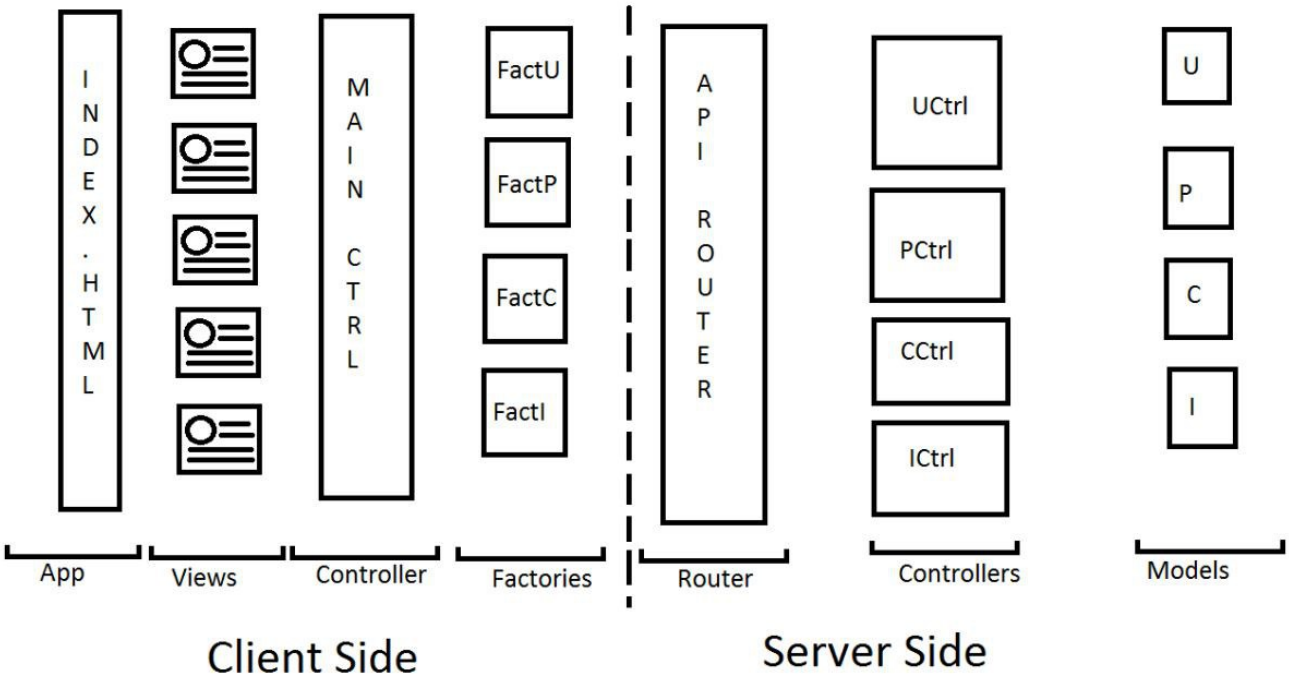
To make a connection, you need to specify database name, if database doesn't exist then mongodb creates it automatically.

```
<?php
    $m = new MongoClient("mongodb://127.0.0.1:27017");
    $db = $m->selectDB('collnet');
?>
```

The functions of MongoDB used are:

- insert()-Inserts a document or documents into a collection.
- find()-Selects documents in a collection and returns a cursor to the selected documents.
- findOne()-Returns one document that satisfies the specified query criteria. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk.
- findAndModify()-Modifies and returns a single document. By default, the returned document does not include the modifications made on the update. To return the document with the modifications made on the update, use the new option.
- remove()-Removes documents from a collection.

# 4. PROJECT DETAILS




**Project Architecture**

# CLIENT SIDE

## INDEX.HTML

The file “index.html” loads the actual page. From there on, the page doesn’t reloads. Only the content of the page updates as per the different views and processing. This is the only file where all the scripts used by the project are included only once.

## VIEWS

The different views incorporated in the app are:

- login
- company-profiles
- company-employees
- contact-us
- edit-profile
- institute-posts
- institute-profile
- institute-students
- main
- profile-self
- search
- user-profile

**MAIN-CONTROLLER**

This is the main frontend controller that uses angularJs. It is responsible for bringing the data requested by client side view from the backend with the help of factories. The $scope service in Angular detects changes to the model section and modifies HTML expressions in the view via a controller. Likewise, any alterations to the view are reflected in the model. This circumvents the need to actively manipulate the DOM and encourages bootstrapping and rapid prototyping of web applications. AngularJS detects changes in models by comparing the current values with values stored earlier in a process of dirty-checking, unlike Ember.js and Backbone.js that trigger listeners when the model values are changed.

// Institute Related

1. **$scope.studiedHere = function(instituteId) {}**
   - Checks whether the logged in student had studied in the institute with given id or not

2. **$scope.addInstituteToProfile = function(details) {}**
   - Adds the input details of the institute to the education history of logged in user

// Company Related

1. **$scope.workedHere = function(companyId) {}**
   - Checks whether the logged in user has worked in the company with the given company id or not

2. **$scope.addCompanyToProfile = function(details) {}**
   - Adds the details of the selected company to the work profile of the logged in user

// User Related

1. **$scope.login = function(details) {}**
   - Logins in a user, invokes a call to the User factory

2. **$scope.logout = function() {}**
   - Logs out a user, invokes a call to the User factory

3. **$scope.register = function(details) {}**
   - Registers a user, invokes a call to the User factory

4. **$scope.redirectNicely = function() {}**
   - Checks login status and redirects to appropriate page

5. **$scope.editProfile = function(details) {}**
   - Edits a profile of user, invokes a call to the User factory

// Post Related

1. **$scope.addPost = function(instituteId, postDetails) {}**
   - Adds the current post details made by user to user model with current time stamp

2. **$scope.addComment = function(postId, commentDetails) {}**
   - Adds the comment details made by the user to the post with given id with current timestamp
3. **$scope.filterPostsBy = function(filterBy) {}**
   - Filter posts by year, discipline, year and discipline or year and discipline and section. This enables a user to check out various groups.

// Misc Functions

1. **$scope.getEducationHistory = function(user, instituteId) {}**
   - Returns education history object of a particular user at an institute.

2. **$scope.getEducationDescription = function(user, instituteId) {}**
   - Returns information of the education history of a particular user at an institute.

3. **$scope.getWorkHistory = function(user, companyId) {}**
   - Returns work history object of a particular user at a company.

4. **$scope.getWorkDescription = function(user, companyId) {}**
   - Returns title and duration of work history of a particular user at a company.

5. **$scope.fillDurations = function(obj) {}**
   - Adds new field 'duration' wherever fromDate and toDate exist

6. **$scope.stringifyDate = function(dateObj) {}**
   - Converts date object into appropriate format string

7. **$scope.toDate = function(dateString) {}**
   - Converts dateString into Date object

8. **$scope.formatDate = function(dateString) {}**
   - Formats a string into a Date object. Assumes format : DD-MM-YYYY

9. **$scope.getSections = function(disciplineCode, disciplines) {}**
   - **L**ooks for the given discipline code in the disciplines list to return the sections available in that discipline.

**FACTORIES**

Factories in angularJs are used to retrieve information or data from the backend and provide it to the front end to display on views.
4 different factories have been used :

1. **factoryUser**
   This factory is used to invoke User related operations and call the server with the request in order to get / send required data from / to the backend.
   For example : signing in, getting profile etc.

2. **factoryPost**
   This factory is used to invoke Post related operations and similarly call the server with the request in order to get / send required data from / to the backend.
   For example : retrieving posts of an institute.

3. **factoryCompany**
   This factory is used to invoke Company related operations and similarly call the server with the request in order to get / send required data from / to the backend.
   For example : retrieving all companies, a company profile etc.

4. **factoryInstitute**
   This factory is used to invoke Institute related operations and similarly call the server with the request in order to get / send required data from / to the backend.
   For example : retrieving all institutes, a institute profile etc.

# SERVER-SIDE

## API ROUTER

All calls from the factories at the client side first hit this page which is located at app/api/index.php. This page simply routes each request to corresponding function/controller by making use of a switch statement. Some high level validations are also made to ensure consistency of data.

## CONTROLLERS

When router invokes a function call of a controller, a particular task is done. The Controllers Folder contains the controller classes responsible for handling client's input and responses coming via router. In all five controllers have been defined in this app as listed below :

1. Controller

   This is the base class for every controller. It basically contains one function called "respond" which helps in communicating with the client side over JSON. **This shows the use of inheritance in our project to reduce code redundancy.**

2. User

   Following are several functionalities User Controller provides :

   - *signup($details)*
     - Creates a new user by taking array details($details)
   - *login($username, $password)*
     - Used to login a user using its username and password
   - *logout()*
     - Used to logout from the session(or collnet)
   - *isLoggedIn()*
     - Used to check if user is logged in or not
   - *getLoginStatus()*
     - Used to display appropriate message if user is logged in or not.
   - *studentsOf($instituteId)*
     - Displays all students of a particular institute.

- *employeesOf($companyId)*
    - Displays all employees of a particular company.

- *getProfile($username)*
    - Used to get the data of a particular username.

- *addCompany($username, $password, $newDetails)*
    - Used to add company to the profile of a particular user.

- *addPost($postDetails)*
    - Used to create and add post with the mentioned post details (using $postdetails)

- *addInstitute($username, $password, $newDetails)*
    - Used to add institute to the profile of a particular user.

- *update($username, $password, $newDetails)*
    - Used to update the details of a particular user (using $newDetails)and since update is being performed password is asked again.

- *changePassword($username, $old_password, $new_password)*
    - Used to change the password of a particular user.

3. Company

Following are several functionalities Company Controller provides :

- *retrieve($_id)*
    - Used to return a particular company (i.e using its particular $_id).

- *retrieveall($filters)*
    - Used to return all companies satisfying the particular filters.

4. Institute

Following are several functionalities Institute Controller provides :

- *retrieve($_id)*
    - Used to return a particular institute (i.e using its particular $_id).

- *retrieveall($filters)*
    - Used to return all institutes satisfying the particular filters.

5. Post

Following are several functionalities Post Controller provides :

- *retrieve($_id)*
    - Used to return a particular post (i.e using its particular $_id).

- *retrieveall($filters)*
    - Used to return all posts satisfying the particular filters.

**MODELS**

Every controller makes use of models in order to do database operations. **This ensures modularity throughout our project.** We have one main model class and four sub model classes. The main model class is used to establish a connection and select a database and each subclass inherits this model class, the model and four sub classes have been explained below:

1. Model

This is the base class for every other model classes used. Here, we are establishing a connection with the MongoDB and our database “collnet”.

2. User Model

This class is used to perform operations on user which involves basic CRUD operations and other helper functions are used such as to authenticate user or to convert password into its hash etc. Following are the constructors and the functions used in this class(with parameters inside the brackets):

- *__construct0()*
  - Used to create an empty object over which we can use create.

- *__construct1($username)*
  - Used to retrieve object with a particular username.

- *__construct2($username, $password)*
  - Used to authenticate a particular user.

- *create($d)*
  - Used to create a user in the database with the details in variable $d.

- *retrieve()*
  - Used to retrieve all of the information present in a profile

- *update()*
  - Used to update the information of a profile.

- *delete()*
  - Used to delete the information of an user from the database(i.e whole of the profile).

- *retrieveById($_id)*
  - Used to retrieve a particular user by his/her id(i.e $_id)

- *studentsOf($instituteId)*
  - Used to get a list of all students of a particular institute(alumni or current student both) using its unique id.

- *employeesOf($companyId)*
  - Used to get a list of all employees(even ex-employees) of a particular company using its unique id.

- *addCompany($details)*
    - Used to add a company in the workhistory section of the user profile with the details specified by the variables $details.

- *addInstitute($details)*
    - Used to add an institute in the educationhistory section of the user profile with the details specified by the variables $details.

- *exists()*
    - Used to check whether a particular username exists or not.

- *authenticate()*
    - Used to authenticate whether the username and the password entered by the user matches to that present in the database.

- *set($d)*
    - Used to set the values of all the variables of a particular object.

- *unsetAll()*
    - Used to unset the values of all the variables of a particular object.

- *to_array2()*
    - Used to make an object an associative array.

- *to_array()*
    - Used to return associative array form of the model.

- *to_json()*
    - Used to convert associative array into json format

- *get_username()*
    - Used to return the username of the current object.

- *get_is_admin()*
    - Used to return true or false depending on whether an user is admin or not.

- *hash_password()*
    - Used to hash the password .

- *verify_password()*
    - Used to verify the password entered by an user with that stored in the database for that particular username.

3. Institute Model

This class is used to perform operations on institute which involves basic CRUD operations and other helper functions are used. Following are the constructors and the functions used in this class(with parameters inside the brackets):

- *__construct0()*
  - Used to create an empty object.

- *__construct1($_id)*
  - Used to retrieve institute's info using _id and by calling the retrieve function in turn.

- *create($d)*
  - Used to create institute in the database with the details in variable $d.

- *retrieveAll($filters)*
  - Used to retrieve information of all the  institutes.

- *retrieve()*
  - Used to retrieve a particular institute's info using its _id.

- *update()*
  - Used to update the information of a institute.

- *delete()*
  - Used to delete the information of an institute from the database.

- *exists()*
  - Used to check whether a particular institute's name exists or not in the database.

- *set($d)*
  - Used to set the values of all the variables of a particular object.

- *unsetAll()*
  - Used to unset the values of all the variables of a particular object.

- *to_array()*
  - Used to return associative array form of the model.

- *to_json()*
  - Used to convert associative array into json format

4. Company Model

This class is used to perform operations on company which involves basic CRUD operations and other helper functions are used. Following are the constructors and the functions used in this class(with parameters inside the brackets):

- *__construct0()*
  - Used to create an empty object

- *__construct1($_id)*
  - Used to retrieve company's info using _id and by calling the retrieve function in turn.

- *create($d)*
  - Used to create company in the database with the details in variable $d.

- *retrieveAll($filters)*
  - Used to retrieve information of all the companies.

- *retrieve()*
  - Used to retrieve a particular company's info using its _id.

- *update()*
  - Used to update the information of a company.

- *delete()*
  - Used to delete the information of a company from the database.

- *exists()*
  - Used to check whether a particular company's name exists or not in the database.

- *set($d)*
  - Used to set the values of all the variables of a particular object.

- *unsetAll()*
  - Used to unset the values of all the variables of a particular object.

- *to_array()*
  - Used to return associative array form of the model.

- *to_json()*
  - Used to convert associative array into json format.

5. Post Model

This class is used to perform operations on company which involves basic CRUD operations and other helper functions are used.Following are the constructors and the functions used in this class(with parameters inside the brackets):

- *__construct0()*
  - Used to create an empty object
- *__construct1($_id)*
  - Used to retrieve a particular post using _id and by calling the retrieve function in turn.
- *create($d)*
  - Used to create a post in the database with the details in variable $d.
- *retrieveAll($filters)*
  - Used to retrieve all the  posts.
- *retrieve()*
  - Used to retrieve a particular post using the  _id.
- *update()*
  - Used to update a post.
- *delete()*
  - Used to delete a post from the database.
- *exists()*
  - Used to check whether a particular post exists or not in the database.
- *set($d)*
  - Used to set the values of all the variables of a particular object.
- *unsetAll()*
  - Used to unset the values of all the variables of a particular object.
- *to_array()*
  - Used to return associative array form of the model.
- *to_json()*
  - Used to convert associative array into json format.