Chapter 1: Introduction

1.1 PROJECT OVERVIEW

Students in colleges and universities like to be remain informed of what new events, schedules, competitions etc. are happening in their college or university. Students had to physically go to a notice board in their college/university and get their desired information. I, as a student, personally felt this problem. I saw an opportunity to devise a solution to this problem.

Building a complete software system for the CHARUSAT university is the solution. It involves various software components like web app deployed to the local CHARUSAT internet, a REST API deployed on the same local internet, an Android app which displays notices using the REST API and also integrates IBM Watson chatbot where users can push their queries and it will give the desired response back to the user. Moreover, there are Wordpress plugins which will be deployed to the main CHARUSAT website where users can view the notices or live updates provided in the system.

To turn this idea into reality, I really had to learn a lot of new things and had to shift my point of view from a programmer's one to the user's one. I often motivated myself by thinking that my project will be used by thousands of people in my college and thereby had to put a lot of effort. It was quite challenging as I had to learn to make REST API. My efforts and hard work had led me to creating a high-quality software product, which will help people get notified of what's happening around in their college or university.

1.2 SCOPE

It is software system, which provide the functionality of notifying the students about the events, and other information by the faculties without using the mail system in an Android App installed in the students smart phone. Android App also contains the CHATBOT functionality through which users can get the general information about the CHARUSAT University and their repective department. That CHATBOT is also to be deployed on the CHARUSAT currently developing website. All notices which are posted by the faculties that were also displayed on the CHARUSAT website with the help of the plugin

so that users can get notified by visiting that website, in case of they not have Android App. But there is problem for iPhone users because the ios App is not developed for this project and in future we will develop the ios application so that iPhone users also not found any trouble of being notified by the faculties about any events and the other information.

1.3 OBJECTIVE

The main objective of this project is to notify the CHARUSAT students with the latest information and events without any mail system or going to any physical noticeboard.

Chapter 2 System Analysis

2.1 USER CHARACTERISTICS

This e-Noticeboard System is divided in three parts. First, one is the front-end part that is the website in which the notices will displayed. Second, one is the back-end part through which the faculties or any other administrator user can post the notices and that is stored in the database and with the help of plugin, notices will displayed on the website. Third, is the android app through which the students or the users can be able to see the latest information or notices and that notices will fetched with the help of REST API in JSON format.

Android User can only get the notices if there is web service exist through which is data from database is transferred to android phone. The REST API in JSON form transfers that information. The main use of REST API is to connect the web and the android phone. Therefore, the main role of REST API is to offer a programmatic interface to web app. Functionality offered to your users via the UI should be offered to third party apps (who want to integrate with yours) via this kind of services.

2.1.1 Modules

The analysis has been identified to be presented with the following modules and roles.

The modules involved are:

- Admin
- SubAdmin(Teachers)

i) Admin

An admin can have all the rights of the SubAdmin. He/She can **create category of the notice and also can create SubAdmines and can change the password them.**

ii)SubAdmin(Teachers)

A SubAdmin can Create Notices according to their rights also can create notice in the format he/she liked by a TEXT EDITOR, can edit, can delete it also can upload necessary files etc.

Functional Requirements

Table 3-2 Admin side Functional requirements

FR1	**Title:** SubAdmin Sign up **Description:** The SubAdmin will be able to sign up through the email_id. The user should provide Name, Addreass , email-id, City, password, phone number etc. **Process:** Validate and insert data into database.
R2	**Title:** Admin login **Description:** The Admin should be able to login to their account. **Process:** Mapping input data with database entries.

2.2 TOOLS AND TECHNOLOGY

Tools Used in DASHBOARD is AdminLTE 2 and the Technology used are Core PHP, MYSQL, JQUERY, JSON, AJAX, HTML 5, CSS 3, BOOTSTRAP 4.

2.2.1 What is DASHBOARD?

A dashboard is a user interface that, somewhat resembling an automobile's dashboard, organizes and presents information in a way that is easy to read. However, a computer dashboard is more likely to be interactive than an automobile dashboard (unless it is also computer-based). To some extent, most graphical user interfaces (GUIs) resemble a dashboard.

I used **Core PHP** because I am familiar with it more than any Frameworks like Laravel, Codeignitor etc.

Below are the snapshots on which I have worked on.

1)Admin Panel of the Web application

This panel is allows Admin or SubAdmin to login in Web application.

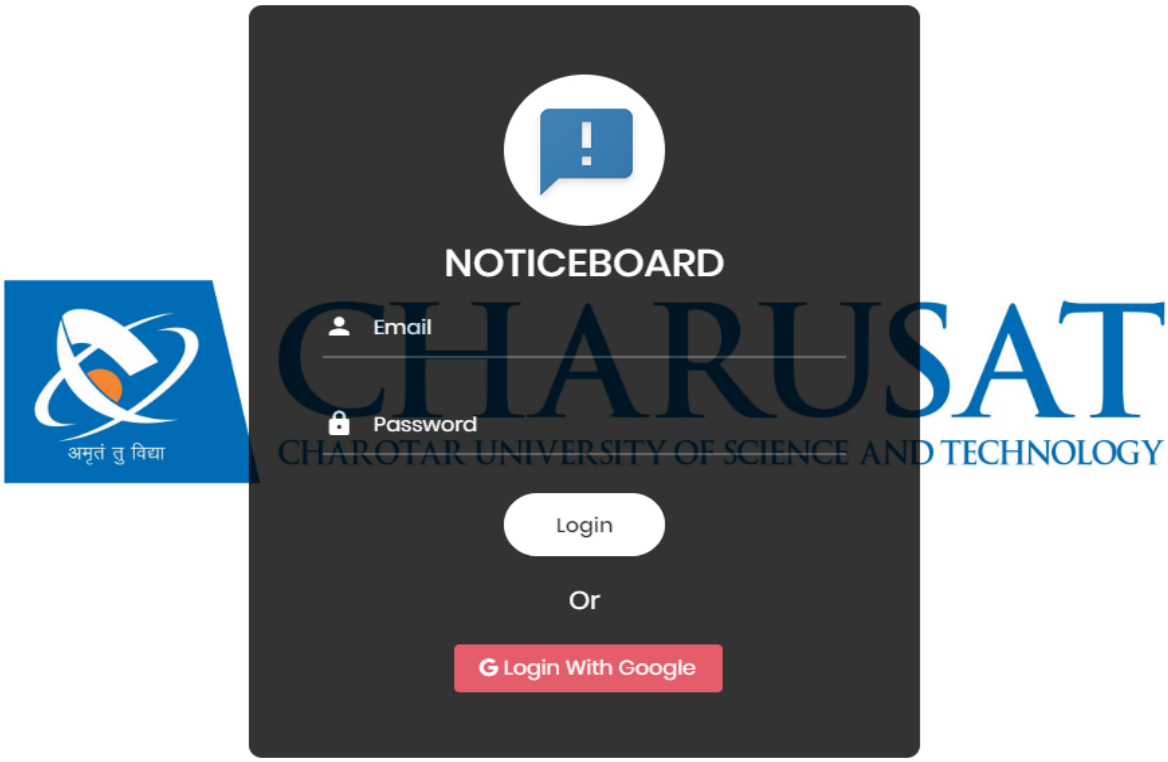


Figure 1.1 Admin Panel

Here, Login with google feature is added but Role management will be done soon.

ADMIN SIDE:

2) Sign up screen of SubAdmin, Create Category and Change Password

SubAdmin can register his/her new Registartion by **Admin** and **Admin** also can create category.

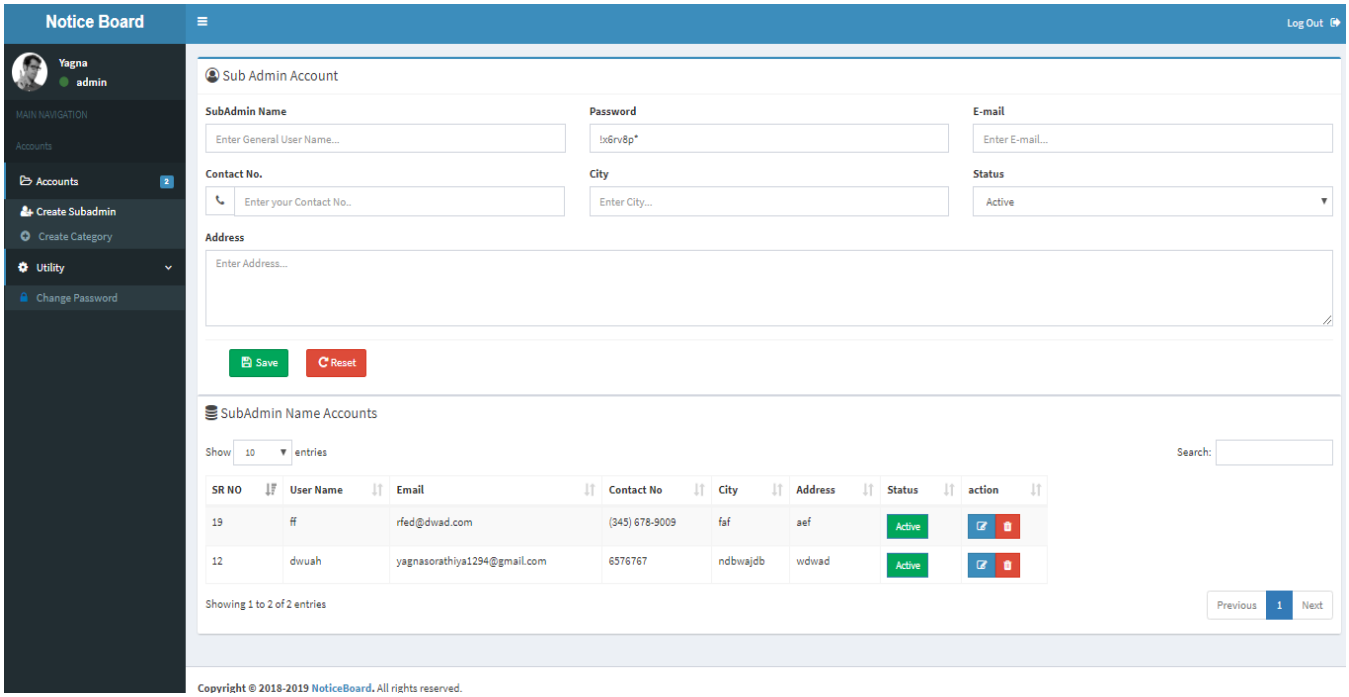


Figure 1.2 SignUp Screen for Management Users

Here, Admin can also **Active** or **Deactive** the User(Teacher) and also **edit** and **delete** them.

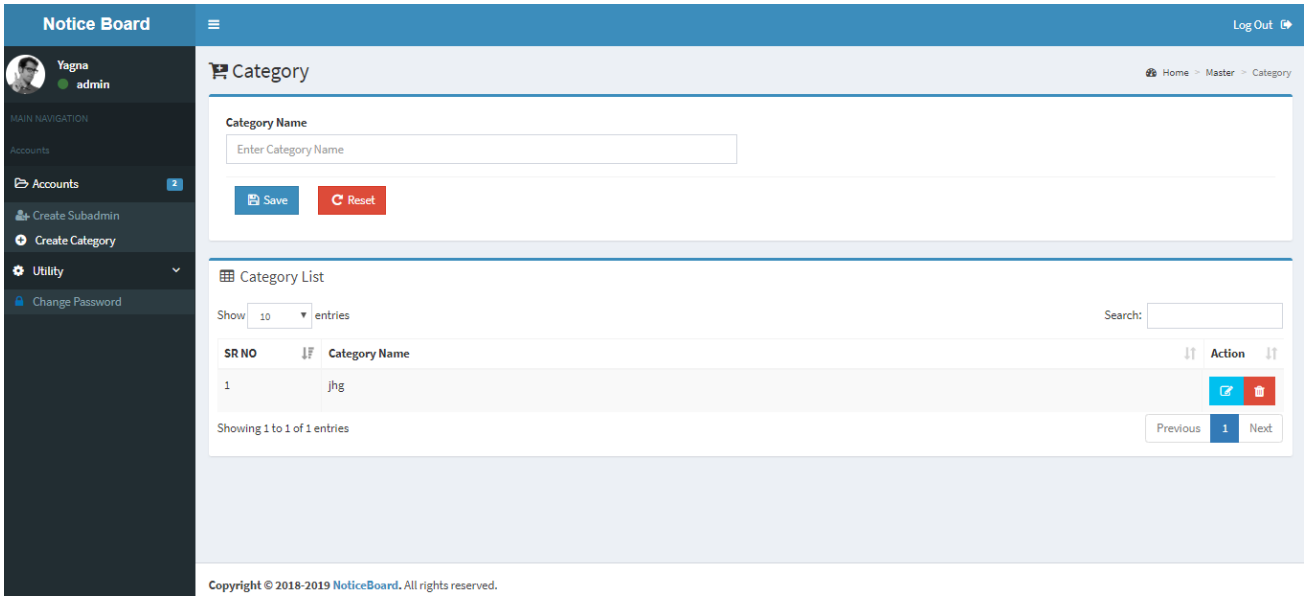


Figure 1.3 Create Category Screen

Also can **create category** of the **notice** so Subadmin later will use this category.

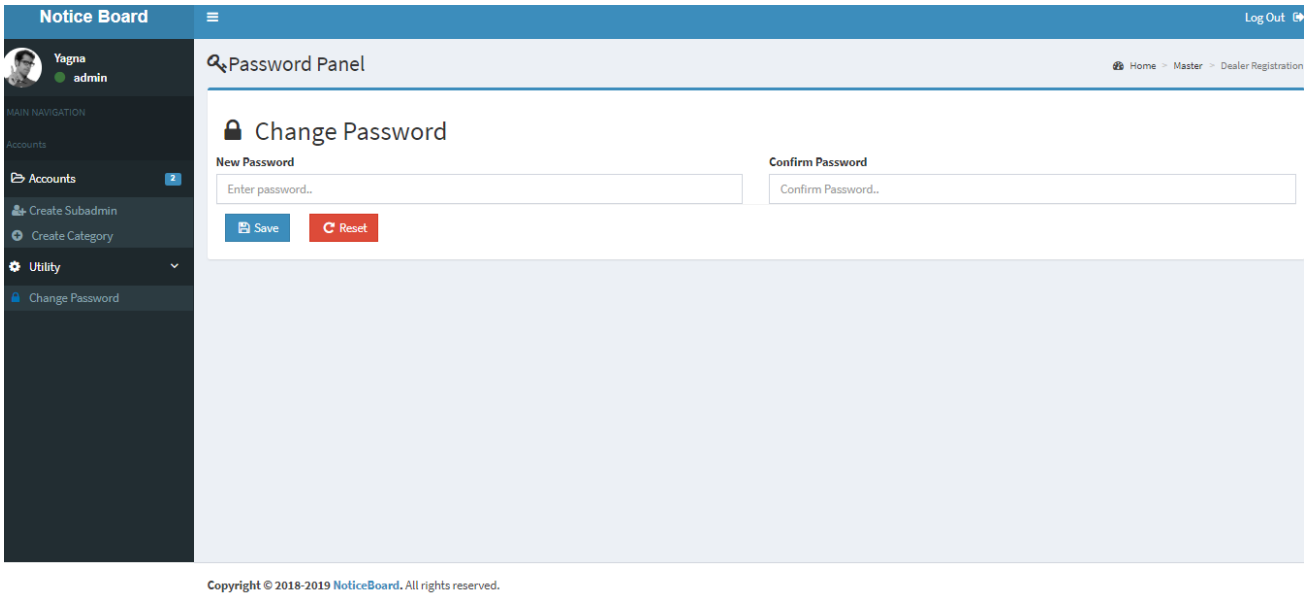


Figure 1.4 Change Password Screen

Admin can **change** the **password** of his/her own by Change Password in Utility section.

SubAdmin Side:

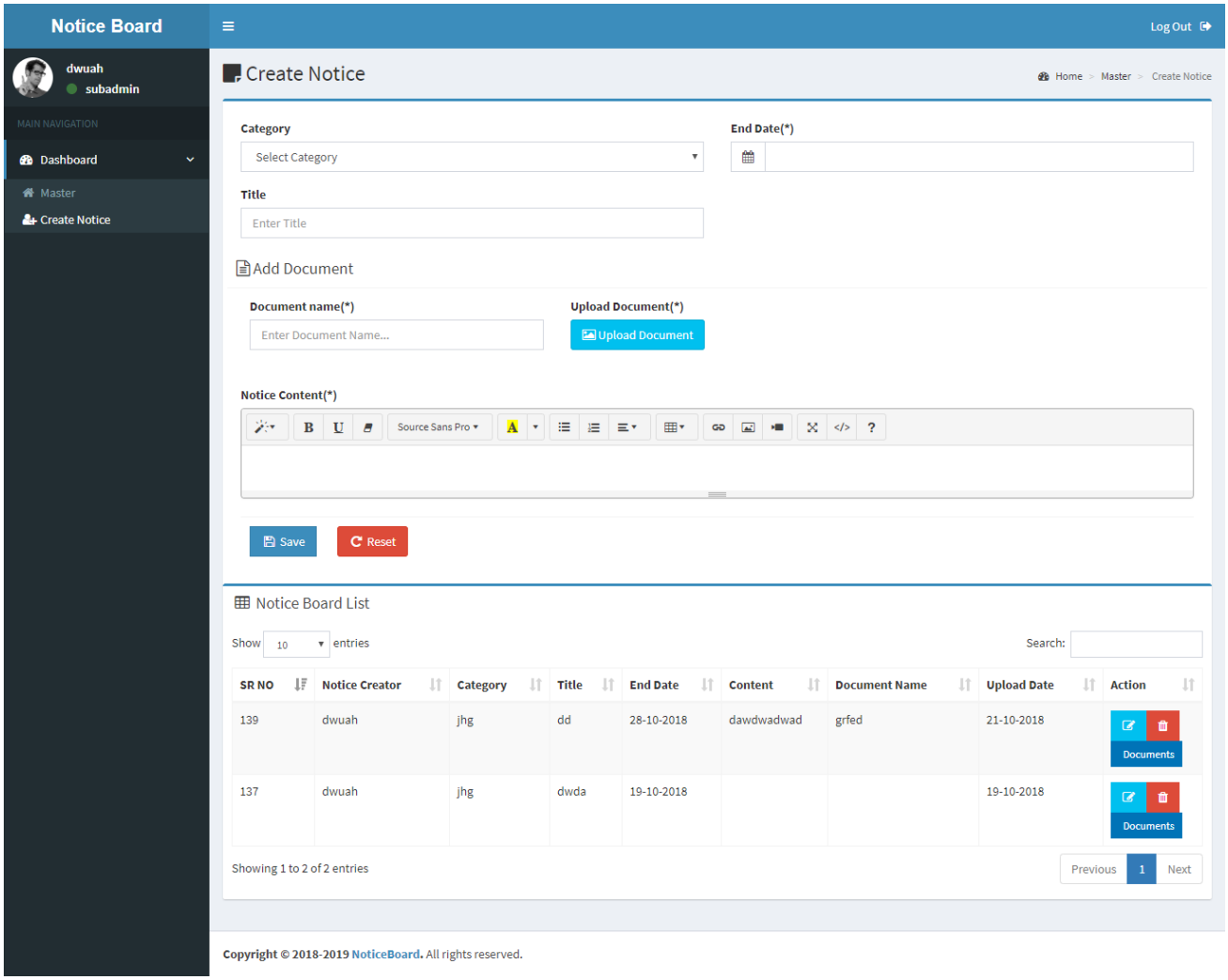


Figure 2.0 SubAdmin Logged Screen

Here, SubAdmin can create notice by selecting the **category** of it which is created by **Admin** and can enter Title of the **Notice** and also can attach the document of any type and also **download** and see it by Clicking **Documents** button in Notice Board List Section.

How Web Pages are Maintained?

There are three files like **noticemaster.php, noticeaction.php and noticedata.php** for each page.

In **noticemaster.php,** there are html,css,bootstrap,ajax,jquery contains and also the rights of the users given by the admin.Insort the file will how the page looks like and what are features of it will display.

In **noticeaction.php,** functions are there will perform the conditions by clicking the button like (insert data,edit data,delete data etc).Insort the functionality of the web page will get run in it and will display the output.

In **noticedata.php,** there are mysql queres which will get executed and fetch the details from the database and display the output according to it.

2.2.2 What is REST API?

Representational State Transfer (REST) web services are one way of providing interoperability between computer systems on the internet. This service allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations. Also in computer programming, an application-programming interface API is a set of subroutine definitions, protocols and tools for building applications. A good API makes it easier to develop a program by providing all the building blocks, which are then put together by the programmer. Therefor a RESTful API is an application program interface that uses HTTP requests to GET, PUT, POST and DELETE data.

REST API could developed in any environment with the help of any programming language. However, in this project, I had created the REST API in the php language with the help of laravel framework that connects the database and the android app. In addition, the android app uses the retrofit library to get the data from the database via REST API.

The whole back-end and the REST API is deployed on the VM system. Therefore, the VM system having Windows Server 2012 operating system and the xampp server software panel is required for this system and the faculties can access the dashboard with address https://172.16.11.70/noticeboard in their system with the help of any web browser so that can upload the notices.

2.2.3 WHY LARAVEL?

Laravel is a free, open-source PHP framework created by Taylor Otwell and intended for development of web applications following the model-view-controller (MVC) architecture.Some of the features of Laravel are a modular packaging system with a dedicated dependency manager, different ways for accessing relational databases, utilities that aid in application deployment and maintenance, and its orientation toward making the syntax more easier to read (syntactic sugar).

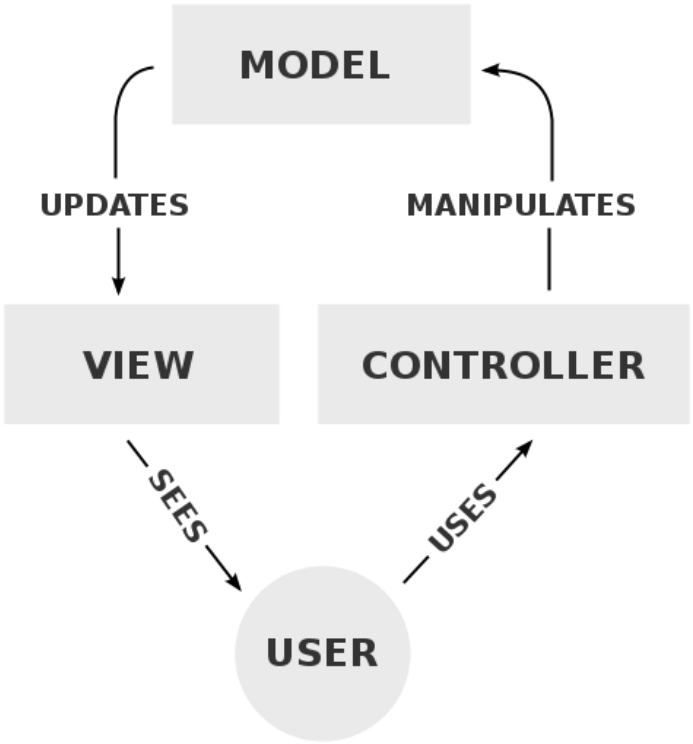
Security is one of the important aspects of managing web applications. Every few days there is a new security threat looming. It assures the users of the website that their data is secured. Laravel is a standout amongst the most well known and advanced PHP frameworks for web applications with its expressive syntax. With eCommerce it ensures an enriching experience for the end users. It helps in creating adaptable and customizable web applications with inbuilt tools Laravel's default authentication provides AES-256 encryption to passwords via a 32-bit key generated during the installation of Laravel.

Laravel provides various mechanisms to secure website. Some of are listed below:

- *Encryption*
- *Storing Password*
- *Authenticating Users*
- *Cross-site request forgery (XSS)*
- *Avoiding SQL injection*
- *Protecting Routes*
- *HTTP Basic Authentication*

The provision of MVC architecture integrated in Laravel, helps making web applications easier, faster and much easy to make changes.

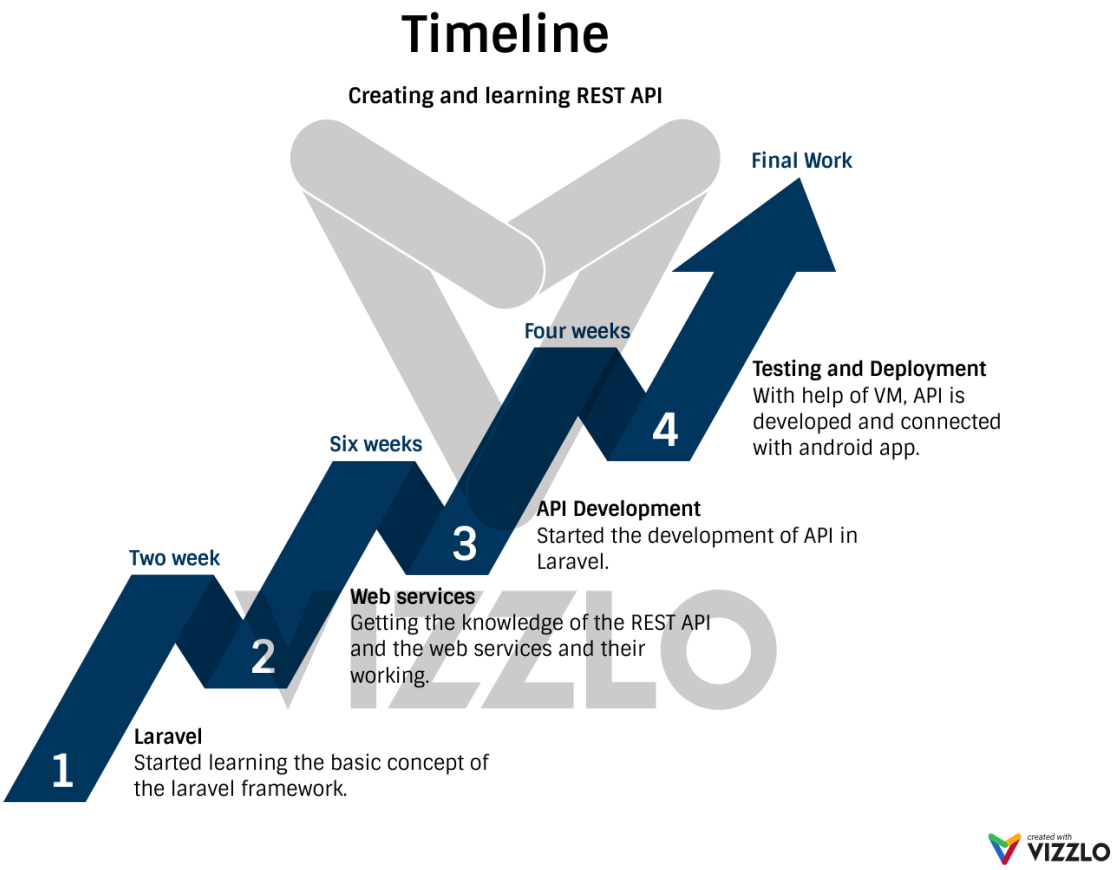
MVC pattern is as follows :-



As per the diagram, view is nothing but what the user sees i.e web page. View consists of HTML and CSS code i.e the visual design part of the web application. When the user clicks a button or submits a form, we require the web application to respond to user inputs. The logic for such actions are written in the Controller part of the architecture. Controller consists of code written in PHP.

Sometimes, a query for database is required to show the user's profile, for instance. The Controller also interacts with the database i.e Model. The code to interact with database is also written in the Controller and data fetched has to be presented to the user via the View.

This is a timeline accounting for every work done beginning from the date of project assigned to its very completion. In each month, we had made a significant amount of progress which led us to creating a good and robust web application.



1) Familiarising with Laravel :

Before even familiarising ourselves with Laravel, we thought of which tools we require for our development of the project. Following are the tools :-

- *PhpStorm IDE*

We searched through the internet for good PHP IDEs. After comparing between 3 IDEs, we chose PhpStorm as our IDE for the project. PhpStorm is a commercial, cross-platform PHP IDE developed by JetBrains. PhpStorm provides an editor for PHP, HTML and JavaScript with on-the-fly code analysis, error prevention and automated refactorings for PHP and JavaScript code. PhpStorm's code completion supports PHP which enhances efficiency to write code. It includes a full-fledged SQL editor with editable query results. At the end of our project, we realised that choosing PhpStorm was the best choice and that has made all the difference.

- *phpMyAdmin*

phpMyAdmin is a free and open source administration tool for MySQL and MariaDB. As a portable web application written primarily in PHP, it has become one of the most popular MySQL administration tools, especially for web hosting services. phpMyAdmin has really helped us manage the database tables easily with so many facilities offered by it. It has avoided us of using the command-line-interface (CLI) for querying and setting up the database for the project.

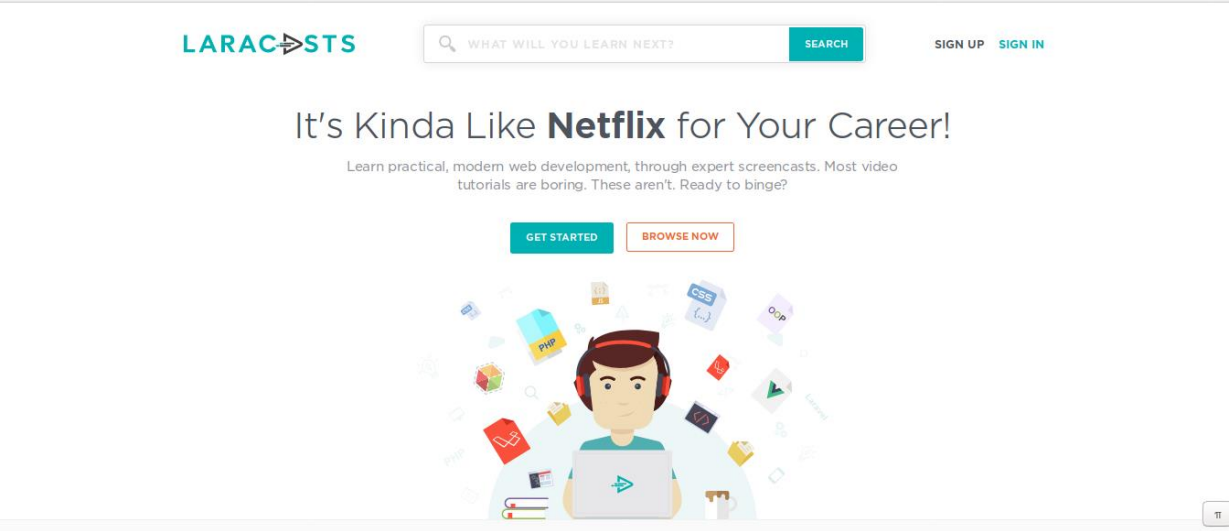
- *XAMPP stack*

XAMPP is a free and open source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P). It is a simple, lightweight Apache distribution that makes it extremely easy for developers to create a local web server for testing and deployment purposes. Everything needed to set up a web server – server application (Apache), database (MariaDB), and scripting language (PHP) – is included in an extractable file. XAMPP is also cross-platform, which means it works equally well on Linux, Mac and Windows. XAMPP clearly avoided all the time required for installation regarding development of a web application. All such things were already done when we installed it on our machines.

As everything was set up on our machines, it was time to get started familiarising with Laravel framework. Installing Laravel with your project was already handled by PhpStorm. You just have to choose Laravel as an additional package while making your project. To begin development, we had to get through a lot of web tutorials offering a course on Laravel development. Most of the courses already assumed that you knew the basic file structure of your Laravel project. One day, we found two sources of tutorials which taught Laravel development for those who are familiar with at least the basics of PHP, CSS and HTML. The sources are as follows :-

- *Laracasts*

Laracasts is the official tutorial for Laravel development. It is different from other web tutorial series. It provides not just theoretical knowledge but even provides you the real-life coding and workflow throughout development of your project. Learning Laravel led us to creating mini projects and teaching us industry-standard practices in web development using Laravel.



www.laracasts.com/

- *Bitfumes tutorials on YouTube :*

- Going through YouTube, we found an excellent YouTube channel called BitFumes Webnologies which teaches web development in Laravel for beginners. Going through his videos, we understood how Laravel actually works, how connections are made to the database, how using the laravel CLI can help your project development. Best part about his tutorials is his teaching methodology. You firstly see what he is doing and then do the coding you wish to do in your sample project. He often assigned the viewers some coding tasks and those actually helped us during the development of this project.

2) Creating mini projects

Mini projects were the projects which we built during our learning period. As we made such projects, we started exploring the official documentation of Laravel (*www.laravel.com//docs/5.5*).

We gained much knowledge and information from the documentation and asking questions on StackOverflow. Mini projects were stepping stones to our project development. These projects include some functionality like eloquent relationships between different tables which we required to be featured in the main projects.

GitHub links to the mini projects are as follows :-

- To-do list (https://github.com/umangjpatel/ToDoList.git)

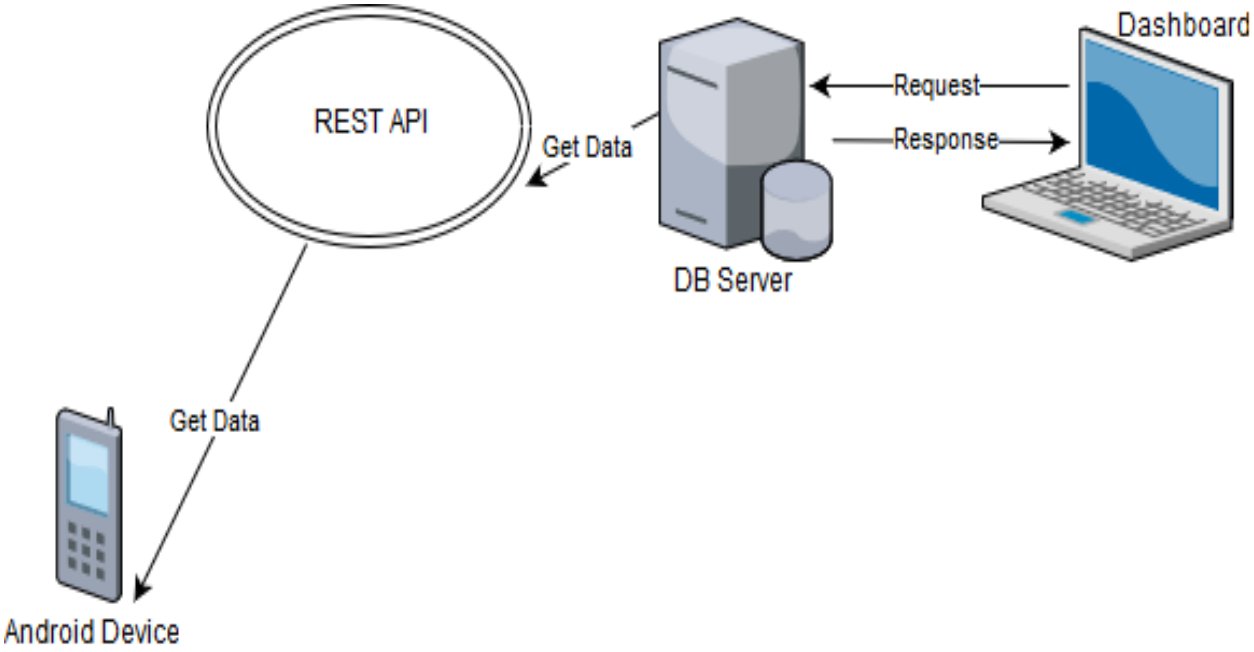
This mini project helped us to familiarise ourselves with the framework. It also taught us how to perform CRUD operations on a database. This project also required us learning to implement Bootstrap CSS components on our visual end so as to create a good user experience for the web application.

- Blog (https://github.com/DozzerCoder/MyBlog.git)

This mini project brushed up our concepts of performing CRUD operations as well as taught us how to make a dashboard displaying all the relevant data. This specific feature is implemented in our project and plays a very crucial part in our web application.

Chapter 3 System Design

3.1 FLOW OF SYSTEM

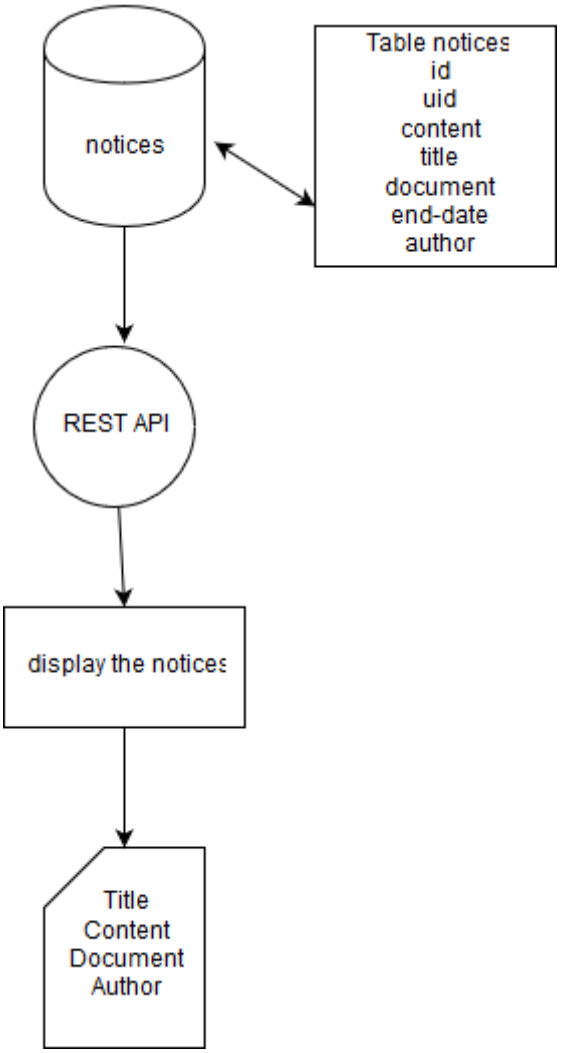


As shown in above figure, dashboard can create the notice, delete the notices, update the notices and read the notice. That notices are stored in the database serve having the table notices.

The REST API is connected to database server with the table notices so that it can get the data and transfer this data to android device.

REST API gets the data from notices table and transfers the data in form of JSON format and after processing that JSON format data. The notices are displayed in the android app with help of retrofit networking library.

3.2 FLOW OF WORKING



In the database notices table, it contains the field like id, uid, content, title, author, enddate. So when a faculty creates a notices with help of dashboard then all the notices and its information are stored in the database server. Now to display all those notices in the android devices. Between the android device and the database table, we are using the web servie REST API which gets the data or notices from the database table in JSON format and after processing it, the data is displayed in the android devices. As shown in above figure.

Chapter 4 Implementation

After learning all the concepts of REST API and learning the laravel framework concepts. It is final task to create the REST API.

4.1 Implementation Environment:

To create the REST API in laravel. First thing I need is the laravel environment in my system and for that I need the composer which is an application-level package manager for the PHP programming language that provides a standard format for managing dependencies of PHP software and required libraries. Phpstorm IDE in which the REST API programming is done and for the testing purpose I had use the postman which send the request and put the data, modify the data and delete the data.

The Postman API allows you to programmatically access data stored in Postman account with ease.

1. You need a valid API Key to send requests to the API endpoints. You can get your key from the integrations dashboard.

2. The API has an access rate limit applied to it.

3. The Postman API will only respond to secured communication done over HTTPS. HTTP requests will be sent a 301 redirect to corresponding HTTPS resources.

4. Response to every request is sent in JSON format. In case the API request results in an error, it is represented by an "error": {} key in the JSON response.

5. The request method (verb) determines the nature of action you intend to perform. A request made using the GET method implies that you want to fetch something from Postman, and POST implies you want to save something new to Postman.

6. The API calls will respond with appropriate HTTP status codes for all requests. Within Postman Client, when a response is received, the status code is highlighted and is accompanied by a help text that indicates the possible meaning of the response code. A 200 OK indicates all went well, while 4XX or 5XX response codes indicate an error from the requesting client or our API servers respectively.

7. Individual resources in your Postman Account is accessible using its unique id (uid). The uid is a simple concatenation of the resource owner's user-id and the resource-id. For example, a collection's uid is {{owner_id}}-{{collection_id}}.

4.1.1 COLLECTION

GET All Collections**:** https://api.getpostman.com/collections

The /collections endpoint returns a list of all collections that are accessible by you. The list includes your own collections and the collections that you have subscribed to. The response contains an array of collection information containing the name, id, owner and uid of each collection.

GET Single Collection: https://api.getpostman.com/collections/{{collection_uid}}

Access the contents of a collection that is accessible to you using its unique id (uid).

POST Create Collection: https://api.getpostman.com/collection

This endpoint allows you to create collections using the Postman Collection v2 format. For more information about the v2 schema, check the format here. On successful creation of the collection, the response returns the collection name, id and the uid.

PUT Update Collection: https://api.getpostman.com/collections/{{collection_uid}}

This endpoint allows you to update an existing collection using the Postman Collection v2 format. For more information about the v2 schema, check the format here.

DELETE Collection: https://api.getpostman.com/collections/{{collection_uid}}

This endpoint allows you to delete an existing collection. On successful deletion of the collection, the response returns the id and uid.

4.2 CODING STANDARDS

api.php

```
<?php



use App\Model\Notice;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Input;

use App\Http\Resources\Notice as NoticeResource;



//

Route::middleware('auth:api')->get('/user', function (Request
$request) {

    return $request->user();

});



//List notices

Route::get('notices', 'NoticeController@index');



//List single notice

Route::get('notices/{id}', 'NoticeController@show');
```

```
//Create a new notice

Route::post('notices', 'NoticeController@store');



//Update a notice

Route::put('notices', 'NoticeController@show');



//Delete a notice

Route::delete('notices', 'NoticeController@destroy');



//Files download

Route::get('files/get', 'FilesController@show');



//Files create and uploaded

Route::post('files/create', 'FilesController@create');



//Search Filter

Route::group(['prefix' => 'v1', 'middleware' => 'auth:api'],
function() {

    Route::get('user', function (Request $request) {

        return $request->user();

    });
```

```
    Route::resource('search',
'NoticeController@getSearchResult');

});
```

As the above code is of the api which is same as routes in normal laravel through which writing this url, we can get the all the notices, single notices by passing id, edit notices, delete the notices. As an example, the index function in NoticeController returns all the notices in JSON format by writitng the url as 127.0.0.1:8000/api/notices

NoticeController.php

```
<?php



namespace App\Http\Controllers;



use App\Model\Notice;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Response;

use Mockery\Matcher\Not;

use App\Http\Resources\Notice as NoticeResource;
```

```
class NoticeController extends Controller

{

    /**

     * Display a listing of the resource.

     *

     * @return
\Illuminate\Http\Resources\Json\AnonymousResourceCollection

     */

    public function index()

    {

        $notices = Notice::paginate(15);



        return response()->json($notices);

    }



    /**

     * Show the form for creating a new resource.

     *

     * @return \Illuminate\Http\Response

     */

    public function create()

    {
```

```
        //
    }


    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request $request
     * @return NoticeResource
     */
    public function store(Request $request)
    {
        $notice = $request->isMethod('put') ?
Notice::findOrFail($request->id) : new Notice();
        $notice->id = $request->input('id');
        $notice->category = $request->input('category');
        $notice->title = $request->input('title');
        $notice->notification = $request->input('notification');


        if ($notice->save()) {
            return new NoticeResource($notice);
        }
    }
```

```
/**
 * Display the specified resource.
 *
 * @param  \App\Model\Notice $notice
 * @return NoticeResource
 */
public function show($id)
{
    //Get notices
    $notices = Notice::findOrFail($id);

    //return one notice
    return new NoticeResource($notices);
}

/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Model\Notice $notice
 * @return \Illuminate\Http\Response
 */
```

```
    public function edit(Notice $notice)
    {
        //
    }


    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request $request
     * @param  \App\Model\Notice $notice
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Notice $notice)
    {


    }


    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Model\Notice $notice
     * @return NoticeResource
```

```
     */

    public function destroy($id)

    {

        $notice = Notice::findOrFail($id);



        if ($notice->delete()) {

            return new NoticeResource($notice);



        }

    }



    public function getSearchResult(Request $request){

        $data = $request->get('title');

        $search_request = Notice::where('title','LIKE',"%{$data}%")-
>get();

        return Response::json(['title'=>$search_request]);

    }

}
```

As above code is of main controller, which is used to get data from database and which format. How the notices are to be displayed i.e. single or all the notices. Main logic of retriving notices and deleting the notices are written here. When the particular url is written the from the notice controller the method is called and the operation is performed.

Noticemigration.php

```
<?php



use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;



class CreateNoticesTable extends Migration

{

    /**

     * Run the migrations.

     *

     * @return void

     */

    public function up()

    {

        Schema::create('notices', function (Blueprint $table) {

            $table->increments('id');

            $table->string('category')->default(0);
```

```
            $table->string('title')->default(0);

            $table->text('content');

            $table->date('end_date');

            $table->string('doc_name');

            $table->string('doc_path');

            $table->timestamps();

        });

    }



    /**

     * Reverse the migrations.

     *

     * @return void

     */

    public function down()

    {

        Schema::dropIfExists('notices');

    }

}
```

Above code explains the creation of table in databse with entity as title, content, uid, id, category, end-date.

NoticeFactory.php

```
<?php



use App\Model\Notice;

use Faker\Generator as Faker;



$factory->define(Notice::class, function (Faker $faker) {

    return [

        'title'=>$faker->text(50),

        'category'=>$faker->text(10),

        'content'=>$faker->text(250),

        'end_date'=>$faker->date(),

        'doc_name'=>$faker->text(10),

        'doc_path'=>$faker->url(),



    ];

});
```

NoticeFactory is used to generate the dummy data by creating the seeder class with command php db:seed after writing this command in the terminal the dummy data of all the notices are created. So that the testing can be easily done without creating form.

4.2.1 Testing and improvements

The last 10% was challenging. Analyzing their opinions and feedback, we made a few minor improvements in the REST API. Prof. Hemant Yadav highlighted a few design flaws in my REST API that could be posed as a serious problem in the future. Considering his helpful review, we began working upon it till those flaws were not corrected completely. Thanks to the MVC architectural pattern embedded in my project, most of the project was kept clean and concise which led to faster completion of my project.

Chapter 5 FEATURES

It is software system, which provide the functionality of notifying the students about the events, and other information by the faculties without using the mail system in an Android App installed in the students smart phone. Android App also contains the CHATBOT functionality through which users can get the general information about the CHARUSAT University and their repective department. That CHATBOT is also to be deployed on the CHARUSAT currently developing website. All notices which are posted by the faculties that were also displayed on the CHARUSAT website with the help of the plugin so that users can get notified by visiting that website, in case of they not have Android App. But there is problem for iPhone users because the ios App is not developed for this project and in future we will develop the ios application so that iPhone users also not found any trouble of being notified by the faculties about any events and the other information.

5.1 Constraints and Future Development

iOS version of this system is not created. So in future, the iOS application of this will created. Hence, the iOS users can also use this system.

Chapter 6 CONCLUSION

This report leads you through the journey of how we **Yagna** and **Darshan** as students learnt so many things and yet developed Dashboard and REST API. Throughout the development of this project, Our perspective had to be changed from a developer's one to the user-oriented one. We had to create the Dashboard so that the management of the notices can easily done through it by the Admin and SubAdmin and the REST API that is used to connect the web with android. A lot of learning took place to reach to the correct destination. We were challenged but We stood up and ended up developing a Dashboard and REST API. Throughout the process, a lot of faculty members and professors enlightened us with key points that were helpful to speed up the development process. Thanks to all the faculty members, Mr. Hemant Yadav and Mr. Ravi Patel and our friends, without them this project could not be much better than it's initial versions.