VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April 2019

MYFFCS TIMETABLE MAKER FOR FFCS USING NODE.JS AND MONGODB

A Project Report

Under the Guidance of,
Prof. THENMOZHI. T

By

16BCI0020	ABHINAV NAIN
16BCE0808	SHUBHAM
17BCE2033	AMREETA KONER

ABSTRACT

FFCS is a time where students of VIT have to make their own timetable by selecting suitable subjects and teachers that they want. MYFFCS is a platform where children will be able to select teachers and courses and make their own mock timetables for their aid during course registration. This system has a login both native and using open authentication so that the users will be able to carry the timetable that they made anywhere and across all devices.

INTRODUCTION

OBJECTIVE: Our objective is simple to make the student make feel more comfortable while doing their FFCS and making their mock timetable. This will help them because many students thinks that teachers will be available or not so within this each and every teacher that will be available during the main course registration will be seen there.

MOTIVATION: many students have been seen days before the FFCS course registration. They think that we will not get this teacher and not that so to not create that problem we introduced the mock timetable maker.

BACKGROUND: The process of registration of courses is hectic and is followed by poor server performance, login crashes etc. Due to aforementioned problems, a lot of students cannot plan and have to rush their registration process, often resulting in getting slot or faculties that were not desired by the student.
This problem can be solved by improving and deploying more servers, stabilising the platform etc. But these are changes that can be completed only by VIT administration. As a third party, we can establish a portal where students can create a planned schedule beforehand so as to avoid making mistakes under any anomalies during registration.

Proposed work:

The first task is making an online portal where people can visualize their timetable and choose their courses. For that I will be making a website

The second task is the backend, which will act as an application program or transaction processor to serve and render dynamic HTML.

The third task is making a database. Here I have chosen a NoSQL database to prevent join loss and optimize complex queries

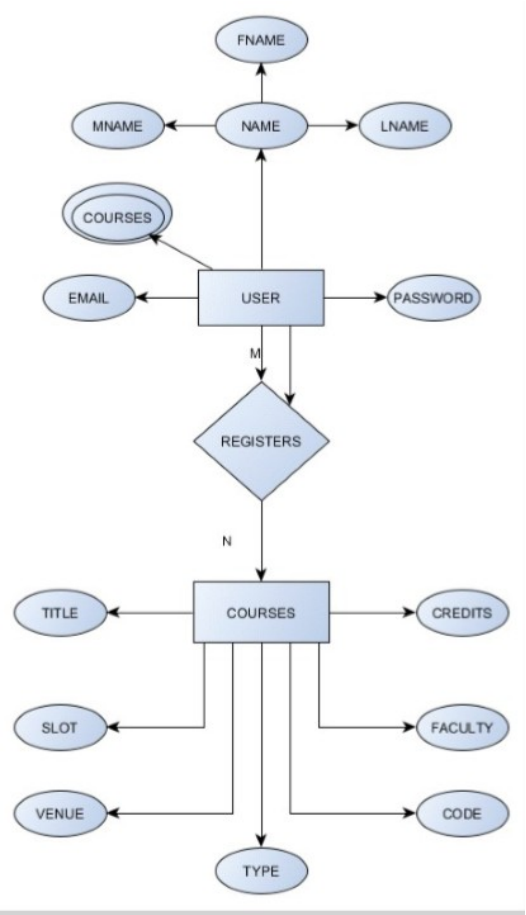
Hardware requirements

An acceptable amount of bandwidth, and RAM.

Software requirements

1. Node JS
2. A text editor (Visual Studio Code in this case)
3. A command line shell (bash)
4. Online hosting service (heroku)
5. MongoDB

ER DIAGRAM OF NoSQL DATABASE



Methodology

Method used

Web interface:

1. The project is rendered onto a server

2. On reaching the route via GET request we render an ejs (embedded javascript) file which asks user login or signup

3. We use Open authentication from google for sign in and also Native nodejs and MongoDB for making and storing transactions

4. The user POSTS his username and password to the backend. Then the node service checks if mongoDB has the same user or not, if not then it creates a user and saves in the database else it throws a message back to the frontend.

5. The POST request is handled for the courses in the backend where we do the following steps->

a) Extract the course ID the user has posted
b) Search the database for the course that has the course ID
c) Send the acquired JSON to the frontend

MongoDB Database functionality:

1. Create: Creating user and course records and saving them in javascript object notation format

2. Read: Reading user and course records and sending them to the application processor

3. Update: Updating the courses array in the user collection so that we can add and change courses that the user has registered for.

4. Delete: Deleting the courses that the user doesn't want from the user course registration list

TEST CASES:

TEST CASE ID	TEST DESCRIPTION	TEST PREREQUISITE	TEST INPUT	TEST RESULT
1	USERNAME	should preceed with a capital letter	Mohit	Accepted
2	USERNAME	special characters are not allowed	$Mohit	fail
3	USERNAME	must not start with numericals	145Mohit	fail
4	PASSWORD	should contain min of 6 character	random124	Accepted
5	PASSWORD	should not exceed the limit of 12	random123457	fail
6	PASSWORD	alphanumeric character with special characterstic are allowed	mohit@123	Accepted
7	LOGIN	click once on login	single click	Accepted
8	LOGIN	click double on login	double click	fail
9	LOGIN	button gets highlighted when mouse is on the button	cursor on the button	Accepted
10	COURSEID	should not start with the number	3020cse	fail
11	COURSEID	should not contain special letters	cse3020@//	fail
12	REQUEST FACULTY	single click to send input	single click	faculty registered
13	REQUEST FACULTY	button highlighted when cursor taken to cell	cursor on button	Accepted
14	COURSELIST	faculty cell gets highlighted when mouse is on the cell	mouse over cell	highlighted
15	COURSELIST	faculty gets booked to slot if clicked	single click on cell	Accepted
16	SLOTCELL	cannot select when the slot is occupied before		fail
17	CREDITS	maximum 27 credits	saving timetable with more than 27 credits	fail

PROJECT DESCRIPTION AND GOALS

Software Requirement Specification

1.1 Document Purpose

Software requirements specification is a document that lays out the description of the software that is to be developed as well as the intention of the software under development. Software requirements specification shows what the software is supposed to do as well as how it is supposed to perform. It is written down before the actual software development work starts.

1.2 Product Scope

FFCS is a time where students of VIT have to make their own timetable by selecting suitable subjects and teachers that they want. MYFFCS is a platform where children will be able to select teachers and courses and make their own mock timetables for their aid during course registration. This system has a login both native and using open authentication so that the users will be able to carry the timetable that they made anywhere and across all devices.

1.3 Intended Audience and Document Overview

Proposed work

The first task is making an online portal where people can visualize their timetable and choose their courses. For that we will be making a website
The second task is the backend, which will act as an application program or transaction processor to serve and render dynamic HTML.
The third task is making a database. Here we have chosen a NoSQL database to prevent join loss and optimize complex queries

Hardware requirements

An acceptable amount of bandwidth, and RAM.

Software requirements

1. Node JS
2. A text editor (Visual Studio Code in this case)
3. A command line shell (bash)
4. Online hosting service (heroku)
5. MongoDB

2.1 Product Perspective

It is a follow on member of the program which helps to operate a software easily with a prior practical layout. It defines a component of a larger system i.e. it helps to work

on FFCS with ease which a part of vtop helping them to select their subjects with prior knowledge and experience.

2.2 Product Functionality

Inputs from the user such as username, password and signup are taken so that the mock FFCS information made by the user is stored in it.
A temporary option for skipping login is there in which the user will be able to see onlythe faculties and their timeslots available and subjects available. Upon successfully performing the login, the user will see the page in which all the subject with lab slots are visible. Search option is there from which you can easily search the subject and will be able to find all the things related the subject in context.
While searching, if one wants to register the subject, it is easily achievable with just a click. An option to modify slots and delete courses is also provided to the user incase there is a need for editing the timetable.

2.3 Users and Characteristics

The main and the important user for this project will be the student specially who are pursuing some degree in vellore institute of technology as it is in vit that the ffcs system is prevalent for choosing their course. It is mainly for the students of vit that this system is designed as they are huge in number.

2.4 Operating Environment

Using the software developed the students can easily design their timetable prior to FFCS. Using MongoDB, which is a noSQL database, we will be able to store data in the form of JSON objects and will be able to form multiple layers of data in one collection.
MongoDB is very efficient, as it prevents the need of using Joins and multivalued dependencies as well as data redundancy.

2.5 Design and Implementation Constraints

A few disadvantages come with this system. This entire process is done online using VIT's portal, but over the years, there have repeatedly been issue with servers, web application glitches and a combination of other factors. These pose a problem, giving a rise to cases where students face problem in setting up the timetable such as one may not get desired faculty, course and slots. To mitigate this issue, one must prepare a lookalike (mock) of the desired timetable beforehand.

2.6 User Documentation

A user help and policy area will be added where the users get to know about the policy
and in help various features and functions of the project will be illustrated for smooth functioning of the system.

2.7 Assumptions and Dependencies

Heroku is a cloud platform as a service, which will be used in the project for hosting backend systems for our web application. This is one of the dependencies of the

project. The other, very important dependency of the project is course database provided by VIT administration. The availability and correctness of this database will determine the usefulness of help provided by the project to students.

3.1 External Interface Requirements

3.1.1 User Interfaces

Web application interface displayed using HTML and Javascript rendered using Node.js.
A login page will greet the user upon accessing home page from the server. Username and Password form included, other login options will also be available to access the database.
After login, user will receive a page displaying courses available in the database, from where the user can begin creating a timetable. Here, slots will be displayed along with issued courses and faculties undertaking them. A button to logout will be available on all pages post login. An option to trigger delete function for removing already selected courses will be provided to the user. A box alongside the timetable will display amount of credits registered.

3.1.2 Hardware and Software Interfaces

Heroku's hosting service will be used for server hardware to store application back-end systems. NodeJS will be used to communicate between server and client to render dynamic HTML. MongoDB will provide No-SQL database interaction on the server. This application is platform independent.

3.1.3 Communications Interfaces

A web browser will be required to interact with the web application. No specific operating system is required. Since the application does not store any data on client, an Internet connection is required to contact and interact with the server.
HTTP protocols will be used. Communication will be protected using algorithms provided by SSL/TLS.

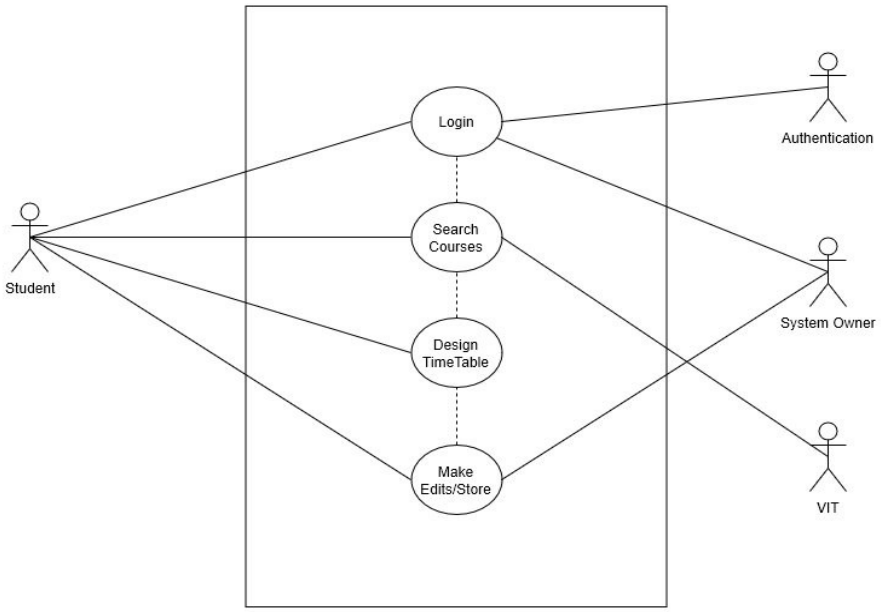
3.2 Functional Requirements

User account facilitates saving timetable configuration for later use. This function is made possible using the backend database, but a Google account sign-in function is also provided.
Timetable maker extracts the course provided by the user using an identifier (course code), searches the database for the course and sends the acquired JSON to the frontend.

3.3 Behaviour Requirements

3.3.1 Use Case View

Authentication provided by Database or Google Sign-in.
VIT databases provide available courses. User acts on designing timetable using available courses. Timetable saved for future use in application database.



4.1 Performance Requirements

- Login authentication should correctly identify credentials.
- Application should have minimum delay (excluding network lag on client side) for displaying data.
- Timetable should be correctly saved associated to the user account.
- CourseID and slot information should be valid.

4.2 Safety and Security Requirements

- Login facility must be safe from attacks like SQL injection and Cross-site Scripting.
- Only concerned user must have access to their timetable configuration.

4.3 Software Quality Attributes

- Ease of use must be a priority while maintaining user interface design aesthetics.
- Must be accessible as intended on all browsers.
- Servers must be available at all times to remove cases of denial of service.

5.1 Data Objects, Attributes and Relationships

There are two primary data objects/entities, User and Courses. User has Name, Password and Email as attributes, as well as Courses which is a derived attribute. The Name attribute is a composite attribute with Fname and Lname. There is a M:N relationship between User and Courses where User registers a Course. Course enitity has multiple attributes like Credits, Slot, Code and Faculty.

Software Design Document

1. Introduction

1.1 Purpose

A written description of design and detail of implementation of the requirements as defined in the Software Requirements Specification. Software design description provides valuable information to be given to software development team for an overall guidance to architecture of the software project.

1.2 System Overview

This project extends the functionality of the Binder Request process that is currently active in PCMS processes. Additional fields and features will be added to the binder request form, new workflow sub-processes will be added to the binder request process, and a process report will be developed that is unique to the binder request process. Metics and TaskViews reports will be available for binder requests, but these will be implemented as the workflow reporting project and will not be included in this SDS.
FFCS is a time where students of VIT have to make their own timetable by selecting suitable subjects and teachers that they want. This project aims to provide a platform where students will be able to select teachers and courses and make their own mock timetables for their aid during course registration. This system has a login both native and using open authentication so that the users will be able to carry the timetable that they made anywhere and across all devices.

1.3 Design Map

The document describes various considerations, design diagrams regarding the project. A brief description of architecture behind the application and high level and low level design elements. Directions regarding user interface are also included.
Use-case, sequence, interaction and collaboration diagrams have been included with this version of the software development specifications.

2. Design Considerations

2.1 Assumptions

Availability of database regarding courses from VIT Software Development Cell. Approval of server deployment on Heroku.

2.2 Constraints

Heroku's platform dependent performance issues like network traffic, server stability and availability. Relevancy of database provided by VIT SDC.

2.3 System Environment

The web application developed using Node JS will reside on Heroku web server. It'll run on client's independent of OS but will require a web browser.

2.4 Design Methodology

The first task is making an online portal where people can visualize their timetable and choose
their courses. For that we'll create a web application.
The second task is the backend, which will act as an application program or transaction

processor to serve and render dynamic HTML.
The third task is making a database. Here we have chosen a NoSQL database to prevent join
loss and optimize complex queries.

2.5 Risks and Volatile Areas

There may be discrepancies between course information provided initially and courses offered during official FFCS registration.

3. Architecture

3.1 Overview

There are two underlying databases for the web application. One holds the credentials required to save a time table for any particular user. The other is a static database that contains the information regarding courses available to register during FFCS. These databases have been composed using NoSQL.
User can create a new account or login using an existing one from the landing page of the web application. The user is then redirected to a personalized page where information regarding courses can be obtained, as well as creation and saving of timetable.

3.2 Strategy 1…N

NoSQL was favored over SQL and other traditional relational databases prevent join loss and optimize complex queries.
Heroku was chosen as it provided more facilities as a freemium service than other competitors.
1. The project is rendered onto a server
2. On reaching the route via GET request we render an ejs (embedded javascript) file which asks user login or signup
3. I use Open authentication from google for sign in and also Native nodejs and mongoDB for
making and storing transactions
4. The user POSTS his username and password to the backend. Then the node service checks
if mongoDB has the same user or not, if not then it creates a user and saves in the database else it throws a message back to the frontend.
5. I handle the POST request for the courses in the backend where we do the following steps->
a) Extract the course ID the user has posted
b) Search the database for the course that has the course ID
c) Send the acquired JSON to the frontend
MongoDB Database functionality:
1. Create: Creating user and course records and saving them in javascript object notation format
2. Read: Reading user and course records and sending them to the application processor
3. Update: Updating the courses array in the user collection so that we can add and change courses that the user has registered for.
4. Delete: Deleting the courses that the user doesn’t want from the user course registration List

4. Database Schema

4.1 Tables, Fields and Relationships

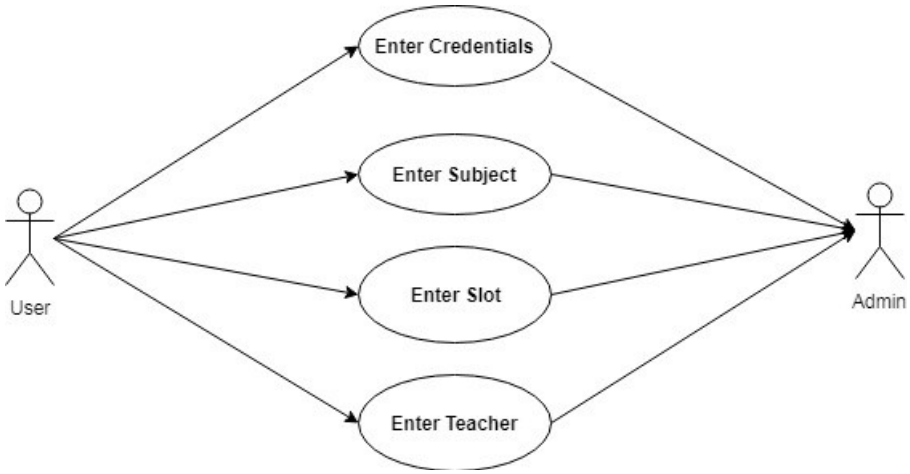
4.1.1 Databases
UserInfo, CourseData

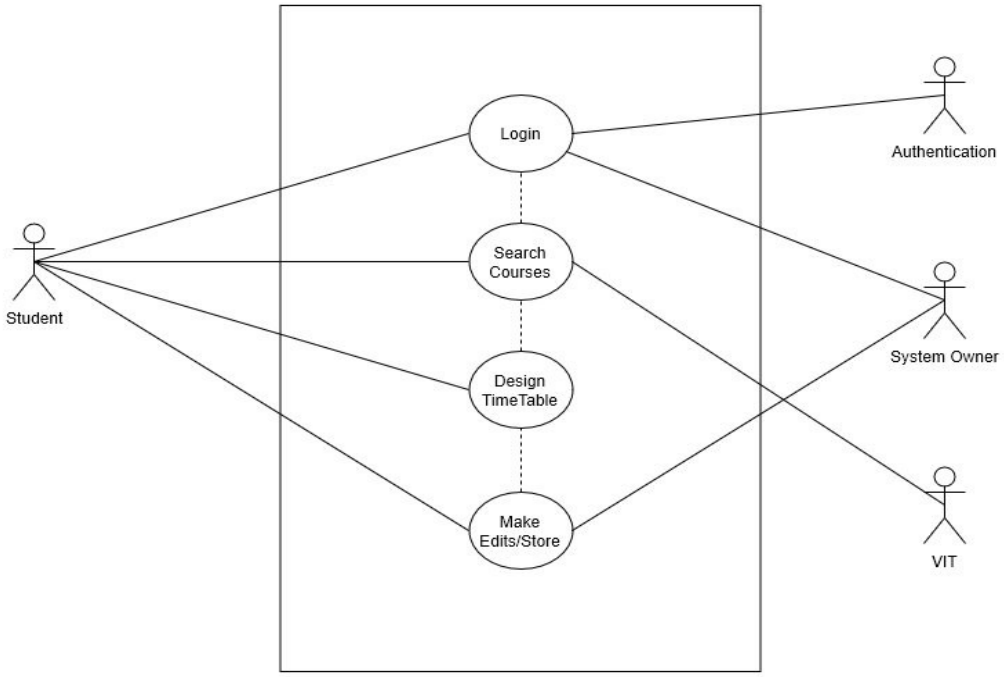
4.1.2 New Fields(s)

Table Name	Field Name	Data Type	Allow Nulls	Field Description
UserInfo	ID	Varchar(50)	NO	Get this field from User when login is initiated.
	Name	Varchar(50)	No	Display this field along with ID when user has logged in
	JSON data	Varchar(100)	No	Store information regarding courses registered with the help of JSON
CourseData	CourseID	Varchar(50)	No	
	Name	Varchar(100)	No	
	Faculty	Varchar(50)	No	
	Slot	Varchar(10)	No	

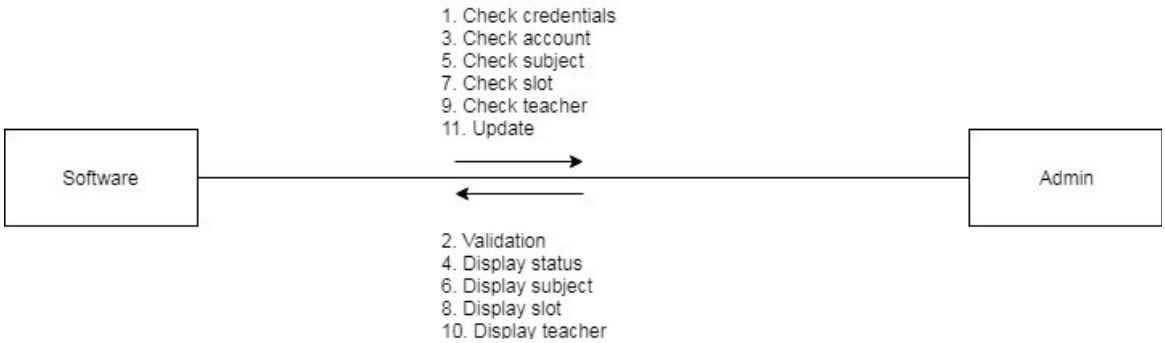
4.1.3 All Other Changes
Relationship between faculty, slot and CourseID is very crucial.

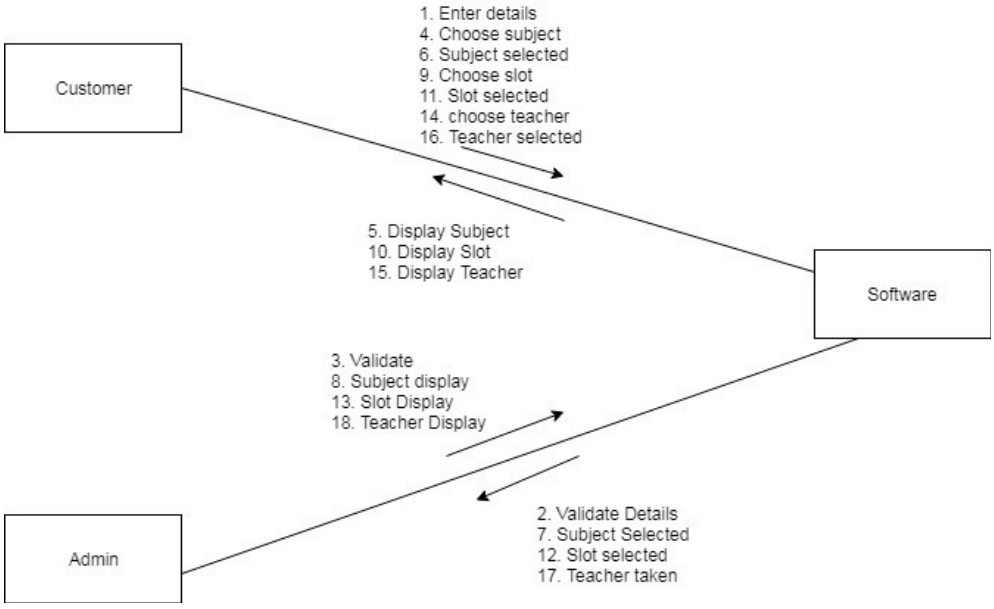
5. Design Diagrams
5.1 Use-case Scenario



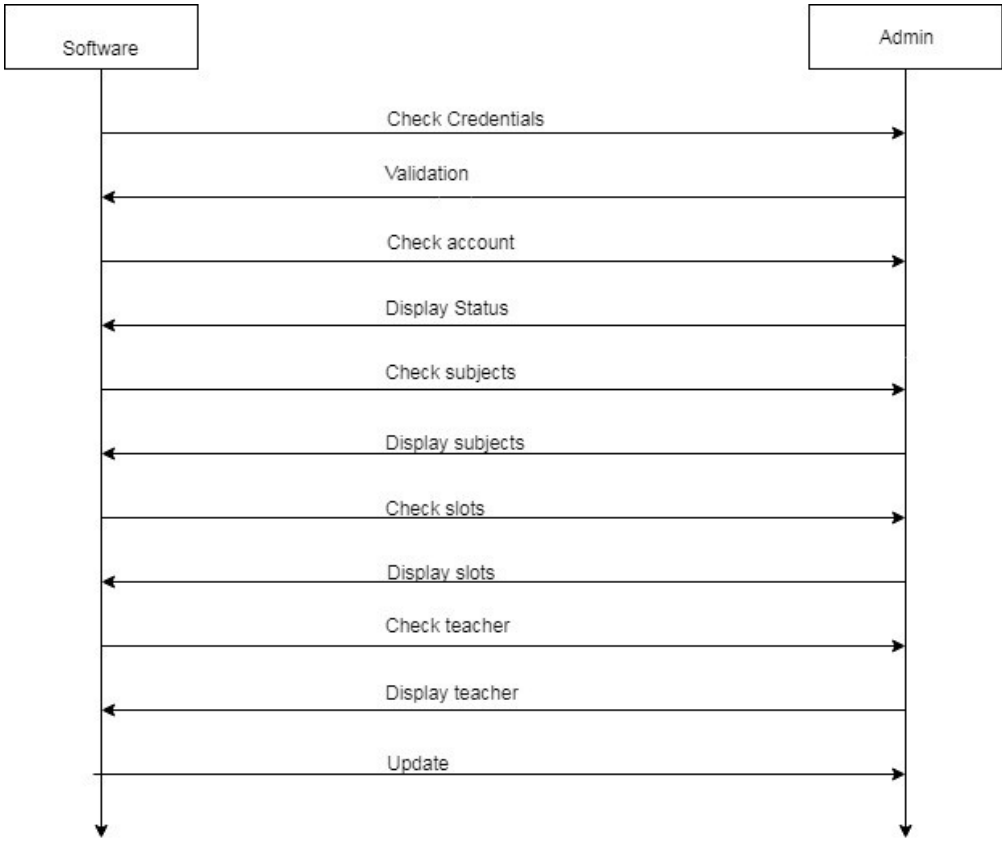


5.2 Collaboration Diagrams

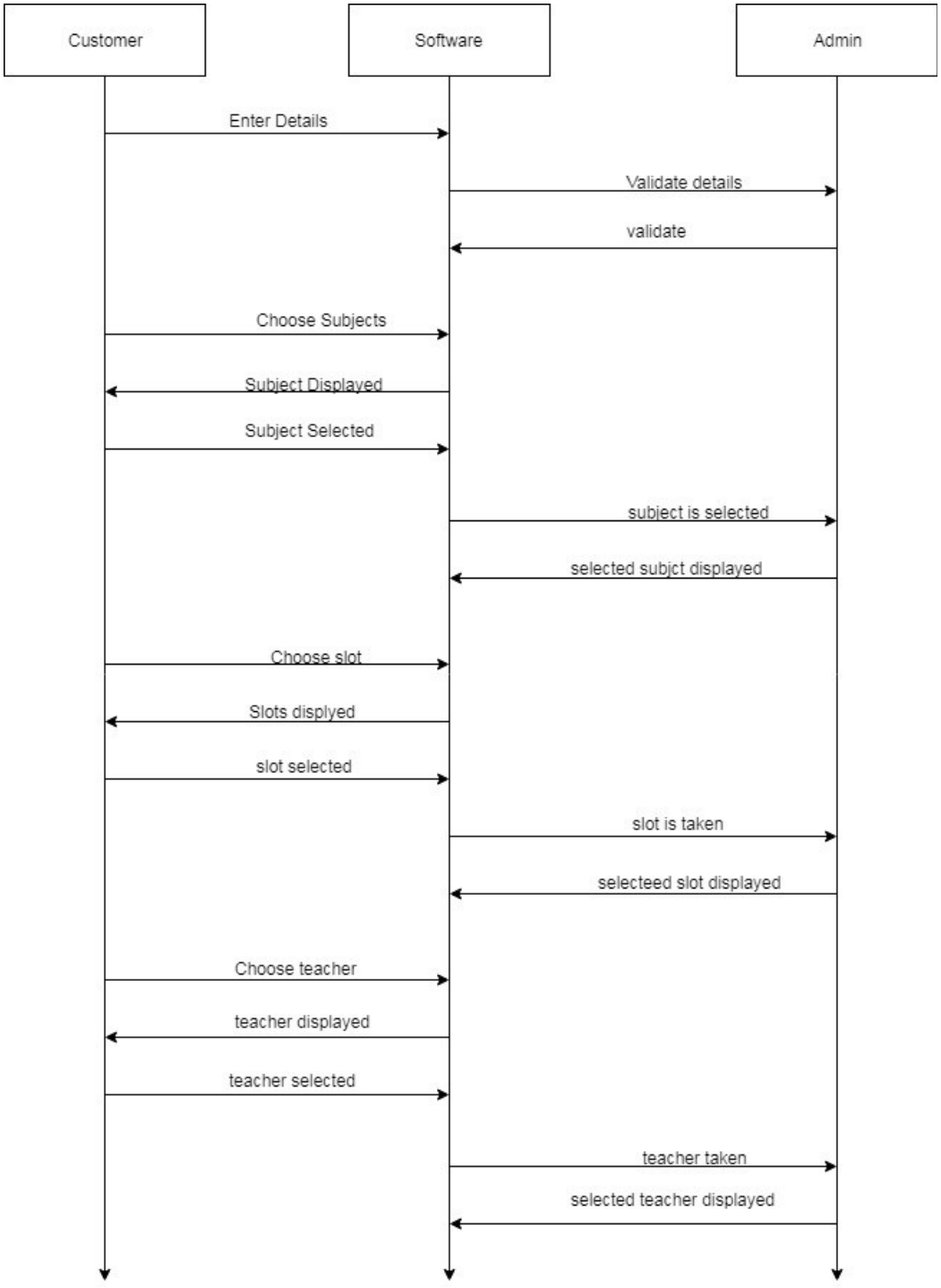




5.3 Sequence Diagram



5.4 Interaction Diagram



USER INTERFACE DESIGN

6.1 Screen 1… N

Landing page:-
User is greeted with a login page. There are multiple ways to login.

Post Login:-
Once the user has logged in, information regarding courses available for registration is displayed.

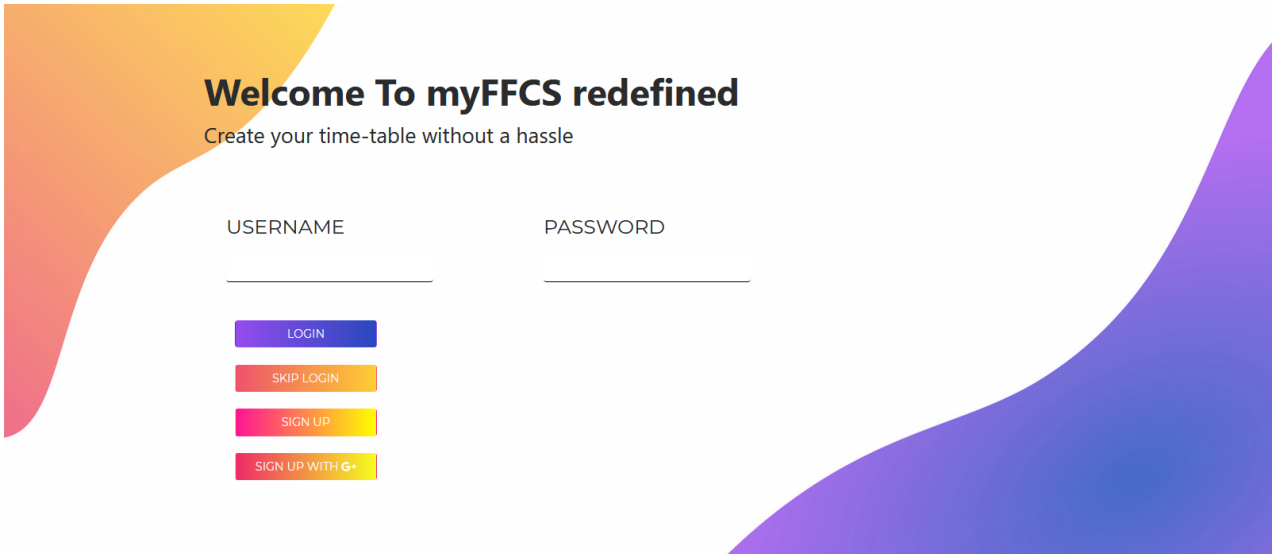
Timetable maker dynamically places courses in slots as selected by user.

CONCLUSION:

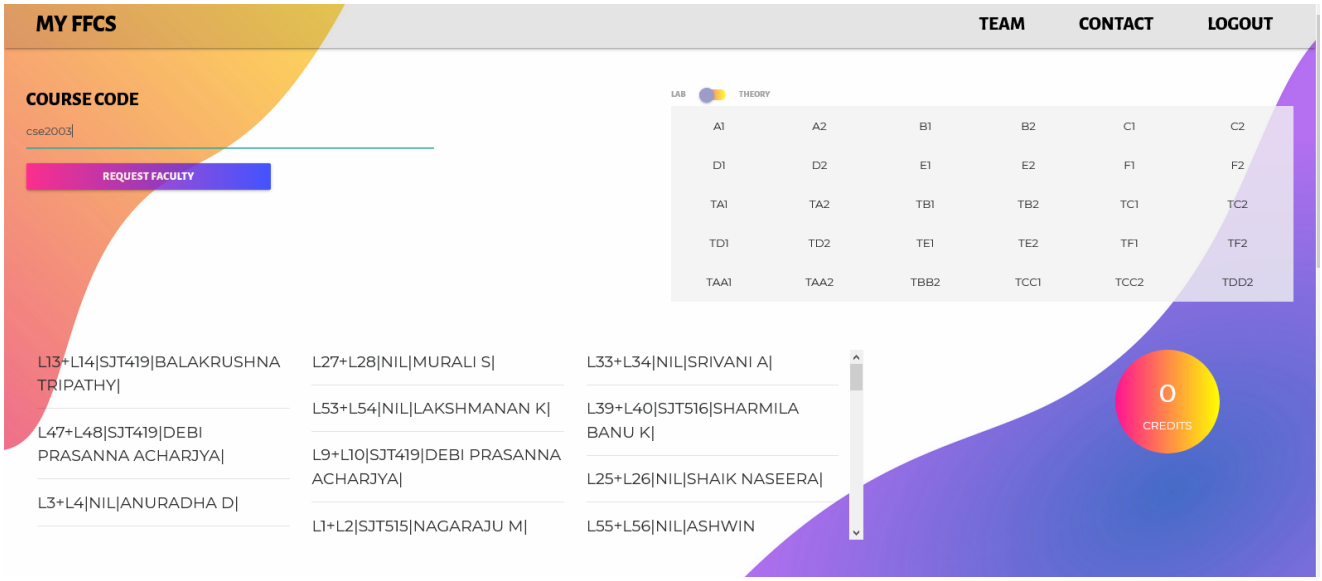
Using MongoDB, which is a noSQL database, we were able to store data in the form of JSON objects and to form multiple layers of data in once collection.
Using mongoDB was very efficient in this case as it prevented having to deal with joins and multivalued dependencies as well as data redundancy.
This project is now functional and very useful for people who want to make their mock timetable before FFCS registration.

SYSTEM IMPLEMENTATION(APPENDIX)

SCREENSHOTS:



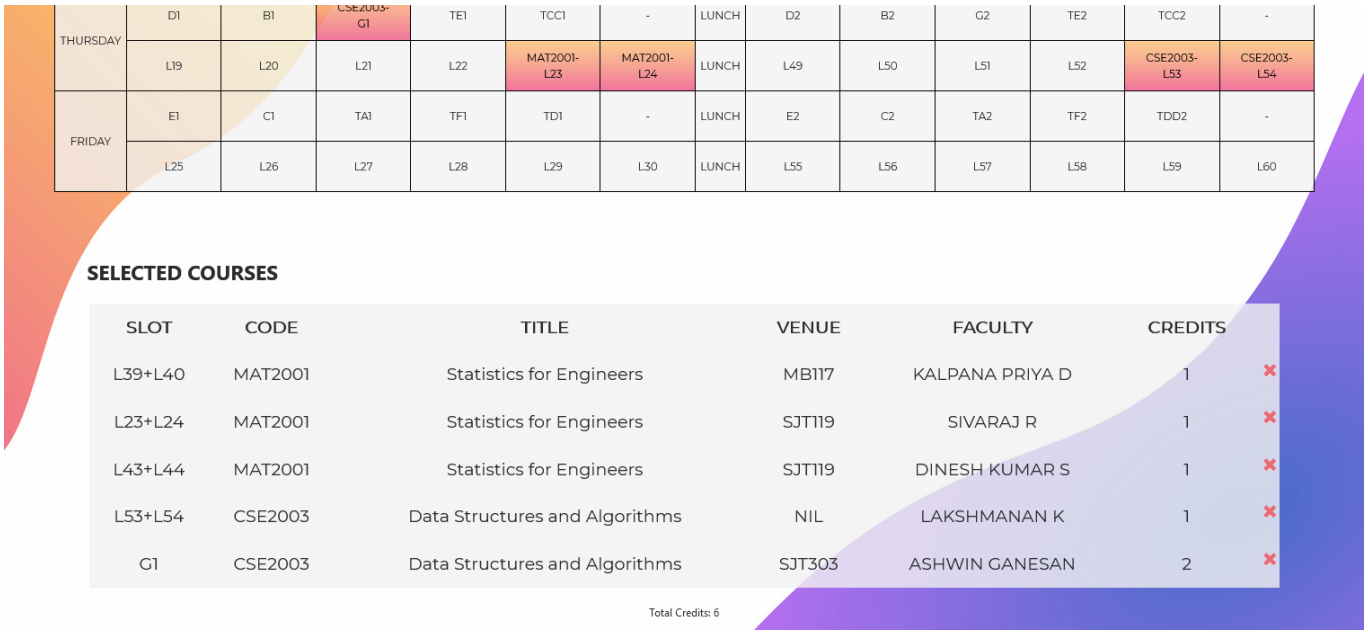
Landing page



Fetching courses from the database and displaying

Weekdays	8:00 to 8:50	9:00 to 9:50	10:00 to 10:50	11:00 to 11:50	12:00 to 12:50	-	1:00 to 1:50	2:00 to 2:50	3:00 to 3:50	4:00 to 4:50	5:00 to 5:50	6:00 to 6:50	-
	8:00 to 8:40	8:45 to 9:30	10:00 to 10:45	10:45 to 11:30	11:30 to 12:15	12:15 to 1:00	1:00 to 1:50	2:00 to 2:45	2:45 to 3:30	4:00 to 4:45	4:45 to 5:30	5:30 to 6:15	6:15 to 7:00
MONDAY	A1	F1	D1	TB1	TG1	-	LUNCH	A2	F2	D2	TB2	TG2	-
	L1	L2	L3	L4	L5	L6	LUNCH	L31	L32	L33	L34	L35	L36
TUESDAY	B1	CSE2003-G1	E1	TC1	TAA1	-	LUNCH	B2	G2	E2	TC2	TAA2	-
	L7	L8	L9	L10	L11	L12	LUNCH	L37	L38	L39	L40	L41	L42
WEDNESDAY	C1	A1	F1	V1	V2	-	LUNCH	C2	A2	F2	TD2	TBB2	-
	L13	L14	L15	L16	-	-	LUNCH	L43	L44	L45	L46	L47	L48
THURSDAY	D1	B1	CSE2003-G1	TE1	TCC1	-	LUNCH	D2	B2	G2	TE2	TCC2	-
	L19	L20	L21	L22	L23	L24	LUNCH	L49	L50	L51	L52	L53	L54
FRIDAY	E1	C1	TA1	TF1	TD1	-	LUNCH	E2	C2	TA2	TF2	TDD2	-
	L25	L26	L27	L28	L29	L30	LUNCH	L55	L56	L57	L58	L59	L60

Timetable maker



THURSDAY	D1	B1	CSE2003-G1	TE1	TCC1	-	LUNCH	D2	B2	G2	TE2	TCC2	-
	L19	L20	L21	L22	MAT2001-L23	MAT2001-L24	LUNCH	L49	L50	L51	L52	CSE2003-L53	CSE2003-L54
FRIDAY	E1	C1	TA1	TF1	TD1	-	LUNCH	E2	C2	TA2	TF2	TDD2	-
	L25	L26	L27	L28	L29	L30	LUNCH	L55	L56	L57	L58	L59	L60

SELECTED COURSES

SLOT	CODE	TITLE	VENUE	FACULTY	CREDITS
L39+L40	MAT2001	Statistics for Engineers	MB117	KALPANA PRIYA D	1
L23+L24	MAT2001	Statistics for Engineers	SJT119	SIVARAJ R	1
L43+L44	MAT2001	Statistics for Engineers	SJT119	DINESH KUMAR S	1
L53+L54	CSE2003	Data Structures and Algorithms	NIL	LAKSHMANAN K	1
G1	CSE2003	Data Structures and Algorithms	SJT303	ASHWIN GANESAN	2

Total Credits: 6

SAMPLE CODE

app.js nodejs backend

```
//https://stackoverflow.com/questions/16827987/expressjs-throw-er-unhandled-error-event
const express = require("express");
const path = require("path")
//for Oauth-2.0 login
const passport = require("passport");

require("dotenv").config()
require('./db/passport_setup.js');

const morgan = require("morgan");

//GraphQL handler
const graphqlhttp = require("express-graphql");
const schema = require("./graphql/schema");
```

```
//session handler
var session = require("express-session");

//handler for post requests
const bodyparser = require('body-parser');

//for main routing functions
const main_router = require("./routes/route");

//slot saving router function
const saveslot_router = require("./routes/slot_saving");

//no-sign-up router
const nosignup = require("./routes/nosignup");

//google+ Oauth-2.0 router
const auth_router = require("./routes/auth_routes");

//load database and save JSON data of courses in it
require("./db/load_in_db");

//set up a server
var app = express();
//for using urlencodedParser in post requests
app.use(bodyparser.json());
app.use(bodyparser.urlencoded({extended:false}));

app.use(morgan('dev'));


//set up a view engine
app.set("view engine","ejs");

//set up middleware for static files
app.use(express.static(path.join(__dirname, 'public')));

//set up a session (It uses cookies)
app.use(session({secret:process.env.COOKIE_SECRET_KEY,
                        saveUninitialized:false,
                        resave:false
                    }));

//use passport js
app.use(passport.initialize());
app.use(passport.session());

//call the main router
app.use(main_router);
```

```
//call slot router function
app.use("/timetable",saveslot_router);

//call no-sign-up router
app.use("/nosignup",nosignup);

//call Oauth-2.0 router for Google+ login
app.use("/auth",auth_router);

//graphql API
app.use("/graphql",graphqlhttp({
    schema,
    graphiql:true
}));

//listen on specified port

app.listen(process.env.PORT || 3000,function(){
    console.log("Listening on localhost: " + process.env.PORT);
});
```

Connect.js used for connecting to DB

```
const mongoose = require('mongoose');
console.log(process.env.MONGO_URL);
mongoose.connect(process.env.MONGO_URL);

mongoose.connection.once("open",()=>{
    console.log("connected to database");
}).on("error",(error)=>{
    console.log("error");
});
```

Model.js schema of mongoDB

```
const mongoose = require("mongoose");

//create a schema for courses
var schema = new mongoose.Schema({
    FACULTY:String,
    TYPE:String,
    CREDITS:String,
    SLOT:String,
    CODE:String,
```

```
    TITLE:String,
    VENUE:String
});


//create a schema for storing user credentials and courses selected
var profileschema = new mongoose.Schema({
    email:String,
    passwd:String,
    courses:[]
});


//create a model based on the schema
var model = mongoose.model('course',schema);


//create a model for the profile collection
var profilemodel = mongoose.model('profile',profileschema);


//export the models
module.exports.courseModel = model;
module.exports.profileModel = profilemodel;
```

Route.js main router

```
const model = require("../db/model").courseModel;
const profileModel = require("../db/model").profileModel;

//to segreagate data into morning and evening theory and lab slots
const segregateData = require("../db/helpers").segregateData;

//hash function for storing passwords
const hashAndSave = require("../db/helpers").hashAndSave;

//verification middleware
const verifyRoute = require("../db/helpers").verifyRoute;

//to check hashed password
const bcrypt = require("bcrypt");

//set up router
const router = require("express").Router();
```

```
router.get('/',(req,res)=>{
    res.render('index');
});



router.post('/',(req,res)=>{

    //if posted registration form
    if(req.body.confirm !== undefined){

        var obj =  { email:req.body.email,passwd:req.body.passwd };

        profileModel.findOne( {email:req.body.email} ).then(function(data){

            if(data === null  ){

                obj = new profileModel( {
email:req.body.email,passwd:req.body.passwd,courses:[] } );

                hashAndSave(obj).then(()=>{
                    //save user email in cookies
                    req.session.email = req.body.email;
                    res.redirect('/timetable');
                }).catch(err=>console.log(err));

            }

            else{
                res.send("Someone with this email already exists");
            }

        });

    }

    //if trying to log in
    else{

        profileModel.findOne( {email:req.body.email} ).then((data)=>{

            if(data === null)
                res.send("User not found");

            else{


                //check hash against password
```

```
                bcrypt.compare(req.body.passwd,data.passwd,(err,result)=>{


                        if(result){
                            //save user email in cookies
                            req.session.email = req.body.email;
                            res.redirect('/timetable');
                        }


                        else
                          res.send("incorrect password");
                    });

            }


    }).catch(err=>console.log(err));

    }


});



router.get("/timetable",verifyRoute,(req,res)=>{

    res.render("timetable");

});



router.get("/timetable/fetch",verifyRoute,(req,res)=>{
    //add timetable looking up TODO

    profileModel.findOne( {email:req.session.email} ).then( (data)=>{


        //to calculate total number of credits
        var credits = (elements)=>{

            var summ=0;

            for(element of elements){
                summ += parseInt(element.CREDITS);
            }
```

```
            return summ;
        }

        res.json({data:data.courses,credits:credits(data.courses)});
    }).catch(err=>console.log(err));

});




router.post("/timetable",verifyRoute,(req,res)=>{


    model.find({CODE:req.body.CODE}).then((data)=>{

        //let segregated_data = segregateData(data);
        res.send( JSON.stringify(data) );

    });


});



module.exports = router;
```

slot_saving.js **main router 2**

```
const profileModel = require("../db/model").profileModel;
const model = require("../db/model").courseModel;
const router = require("express").Router();
var checkClash = require("../db/helpers").checkClash;
var verifyRoute = require("../db/helpers").verifyRoute;



router.get("/logout",verifyRoute,(req,res)=>{
    req.session.email='';
    if(req.user) req.logout();
    res.redirect('/');
```

```
});



/* makes an array of clicked course slots and matches them against all alots
in the database,
if match found then slot clashed */
//There is a better method to do this

router.post("/save",verifyRoute,async (req,res)=>{
    console.log(req.session.email+" Logged in");

    // make an array of slots of the courses being registered
    let slots = req.body.SLOT.split("+"),str = '',cr = 0;

    //string waits for the promise to be completed and then gets assigned the
value of all the slots
    let variable = await
profileModel.findOne( {email:req.session.email} ).then( (data)=>{

        data.courses.forEach( (element)=>{
            //make a string by concatenating all slots in courses
            str += (element.SLOT + '+');
            cr += parseInt(element.CREDITS);
        });
        return {slots:str,credits:cr};

    }).catch( (err)=>{
        console.log(err);
    });

    console.log("total credits = "+(parseInt(req.body.CREDITS)
+variable.credits) );

    //TODO check if applying for same course again, TODO theory and lab
clashes

    // if requested for a course which makes more than 27 total credits then
throw alert
    if(variable.credits + parseInt(req.body.CREDITS) > 27)
        res.send( {status:"limit",info:27-variable.credits} );

    //checkClash returns a clashed slot if slots are clashed
    else if( checkClash(slots,variable.slots) )
        res.send( {status:"clashed",info:slot} );

    //if no clash then update the database
    else profileModel.update( {email:req.session.email}, {$push:
{courses:req.body} } ).then( ()=>{
```

```
        console.log("updated");
        res.send( { status:"updated",info:variable.credits +
parseInt(req.body.CREDITS),course:req.body } );
    });


});



//deletes an element after clicking on a specific button
router.delete('/del',verifyRoute,(req,res)=>{

  console.log(req.body);

  if(req.body._id === undefined){
    profileModel.update({email:req.session.email}, { $set : {courses: [] }} ,
{multi:true} ).then(()=>{
      return res.send("Deleted courses array");
    }).catch(console.log);
  }

    profileModel.update( {email:req.session.email},{$pull: {courses:
{_id:req.body._id}} }  ).then( ()=>{

        console.log("removed course!");

        //TODO res.redirect('/timetable') but it doesnt work
        res.send("Deleted course")//res.redirect("/timetable");

    }).catch( (err)=>{
        console.log(err);
    });


});




//predictive text, receives AJAX requests after typing every character in
input box
router.post('/predict',verifyRoute,(req,res)=>{


    var array=[],unique=[];

    //matches regex for text
```

```
    model.find( {CODE:  {$regex: req.body.code } } ).then((data)=>{

        if(data===undefined) res.send('');

        //to send only titles and course codes to the front end
        for(element of data){
            var obj = {code:element.CODE,title:element.TITLE} ;

            //TODO need to remove duplicate results, but not working
            if(array.indexOf(obj)===-1)
                array.push(obj);

            //to send just 5 results
            if(array.length === 5)
                break;
        };
        //console.log(array);
        res.send(array);
    });
});



module.exports = router;
```