# **ISDL LAB PROJECT**

**TESTING DOCUMENT**

***Topic:***

**COURSE REGISTRATION AND TIMETABLE GENERATION**

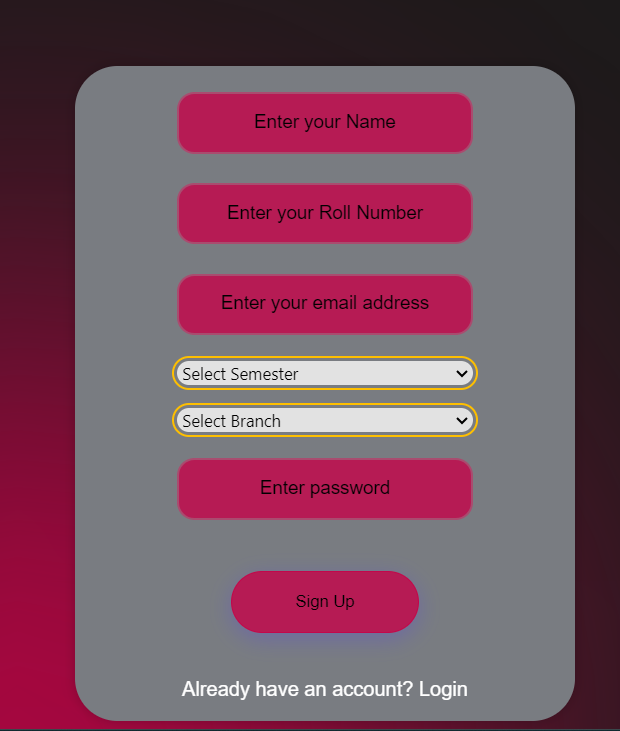
* Pranjal Jainl 21UCS155 *(Product Manager)*
* Pranjal Bansal 21UCS154
* Pranjal Mishra 21UCS156
* Vaibhav Khamesra 21ucs224
* Vaibhav Rai 21ucs225



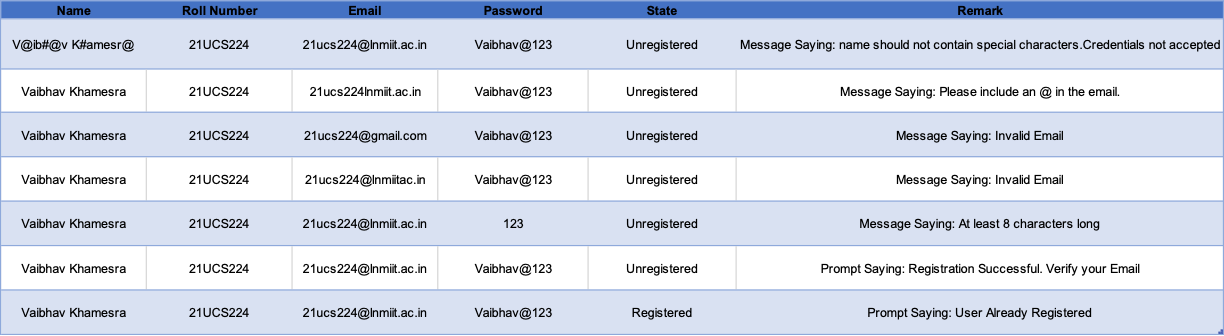
*Department of Computer Science and Engineering  
The LNM Institute of Information Technology*

**BLACK BOX TESTING**

**Account Sign-up page**

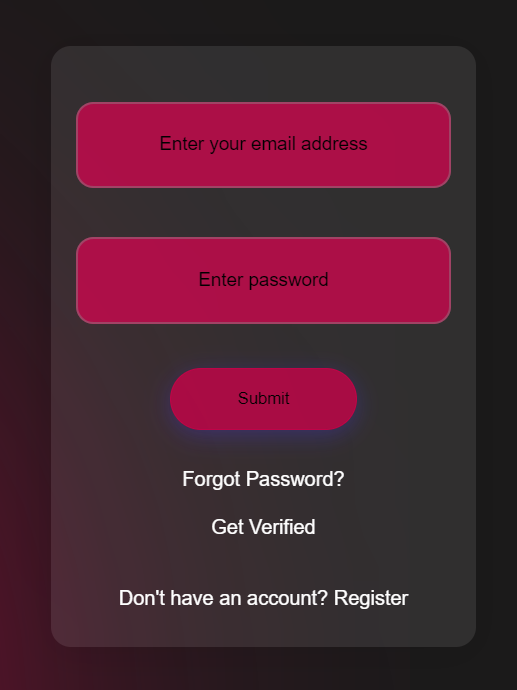


**Test Cases:**

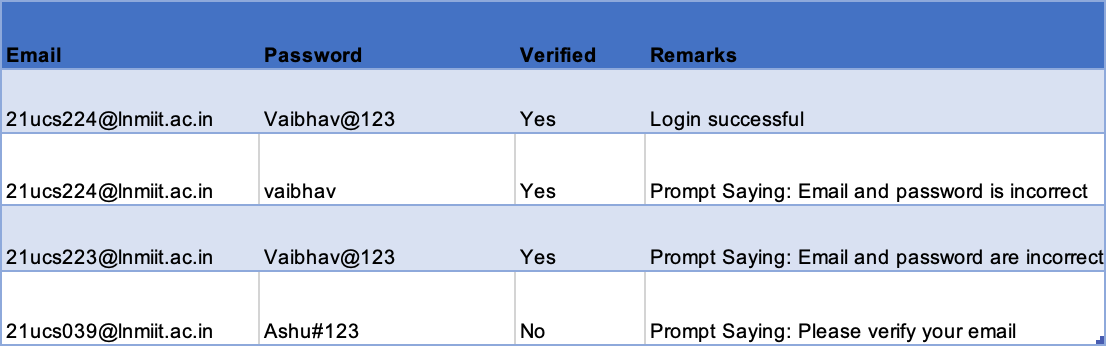


Email Verification Link received after successful validation of detail

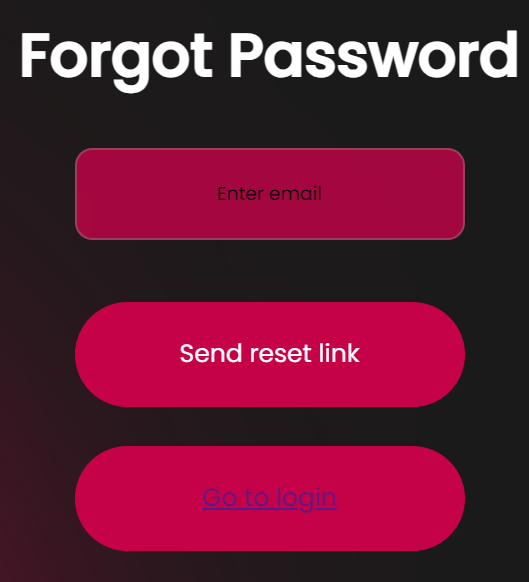
**Account Log-In page**

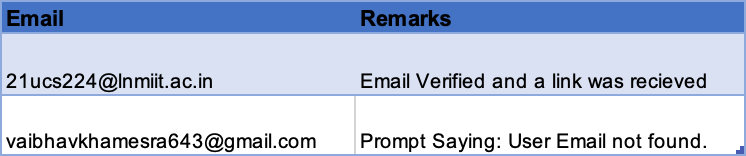
****

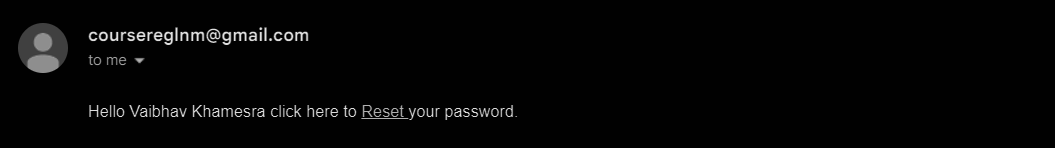
**Test Cases:**

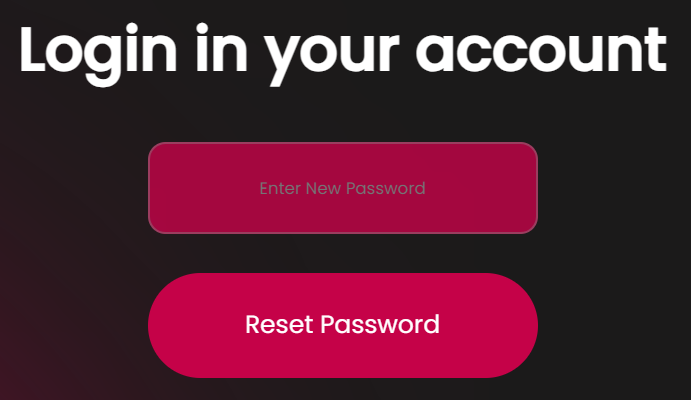
****

**Forgot Password:**

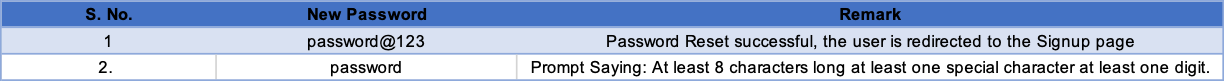
****

****

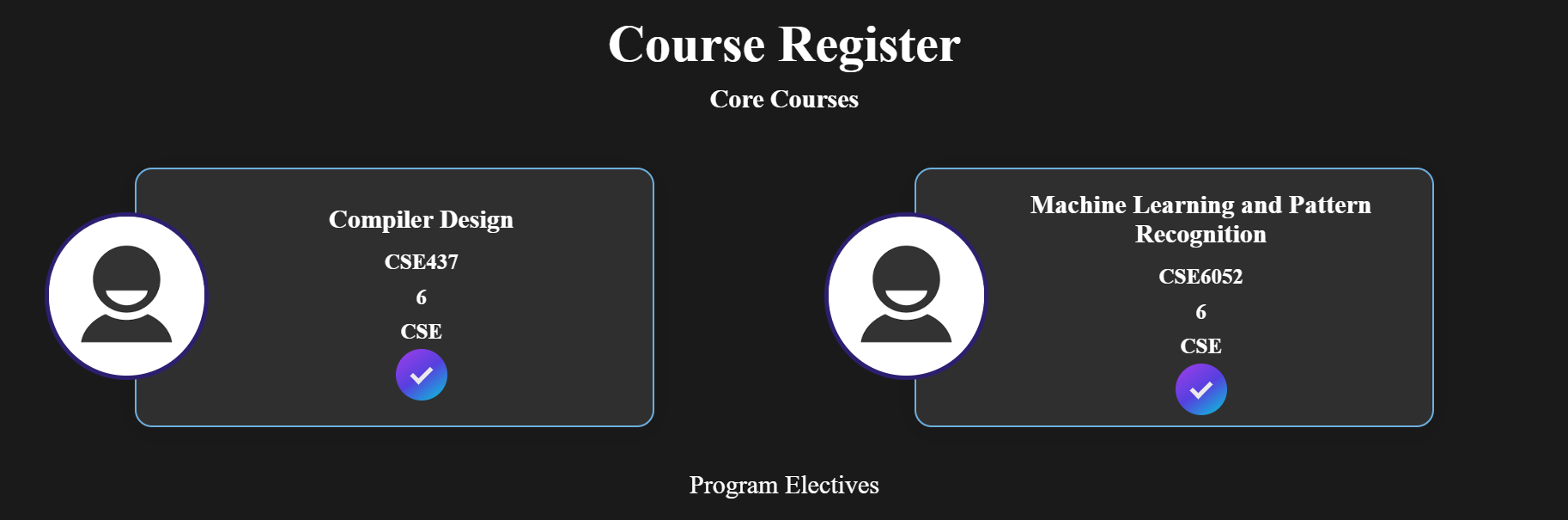


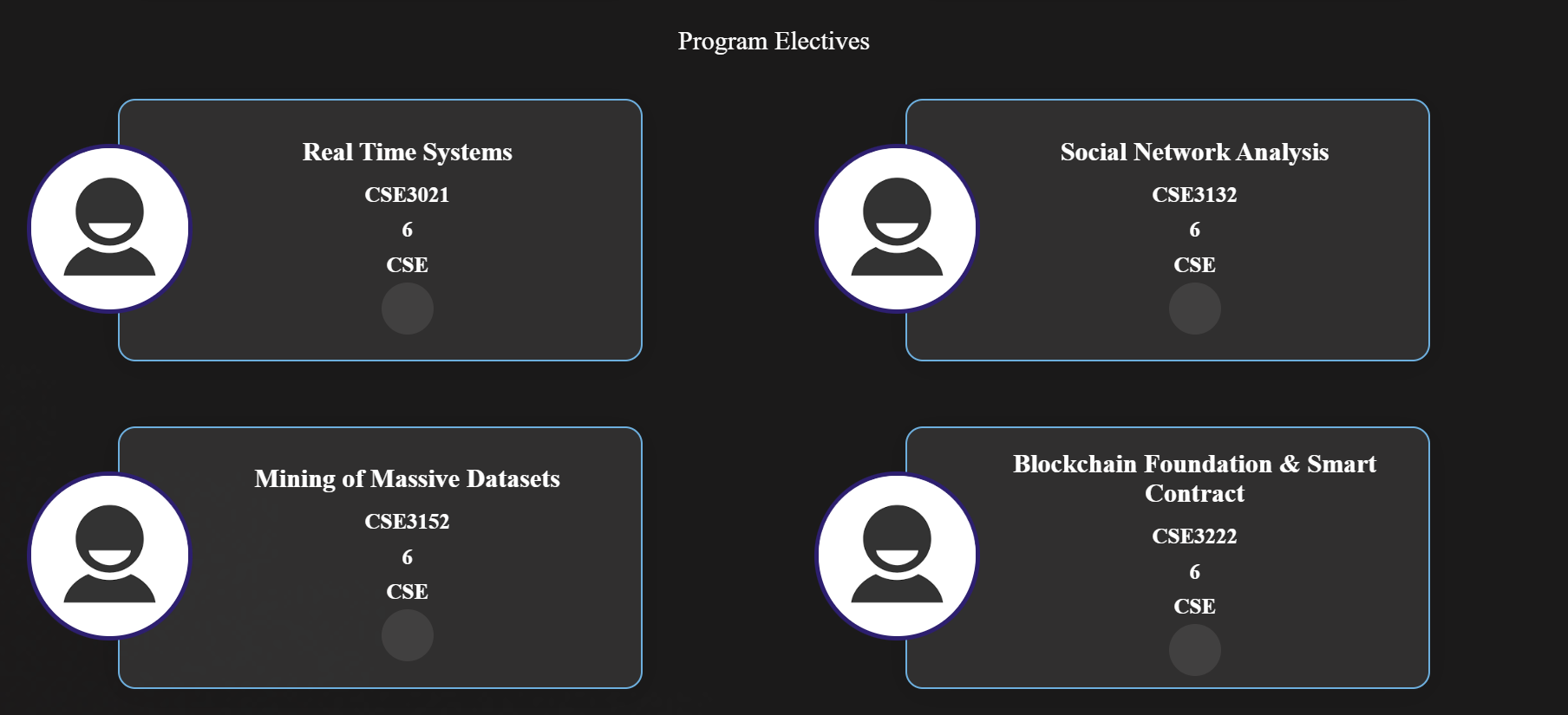


The link received on the email redirects the user to the reset password page as above.

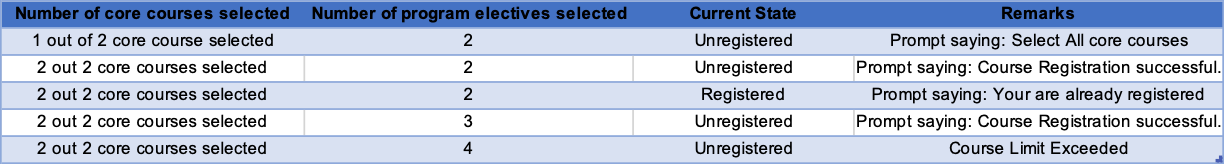


**Course Registration Module:**

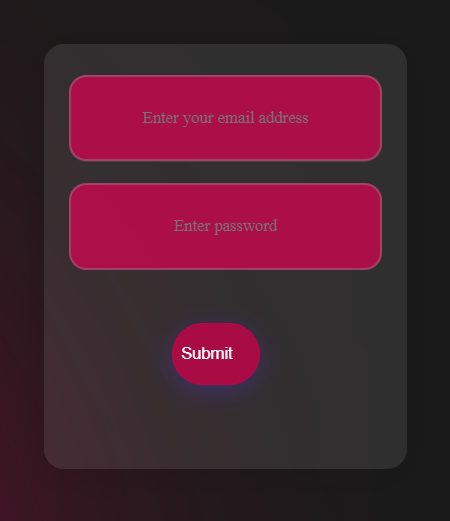
****

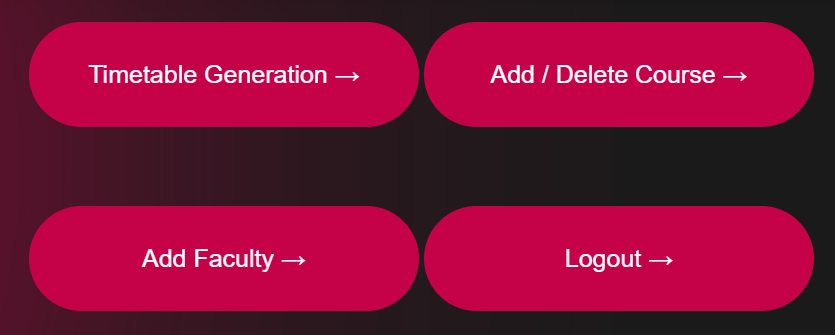
****

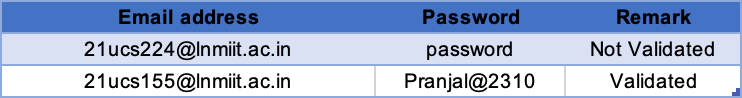
**Test Cases:**

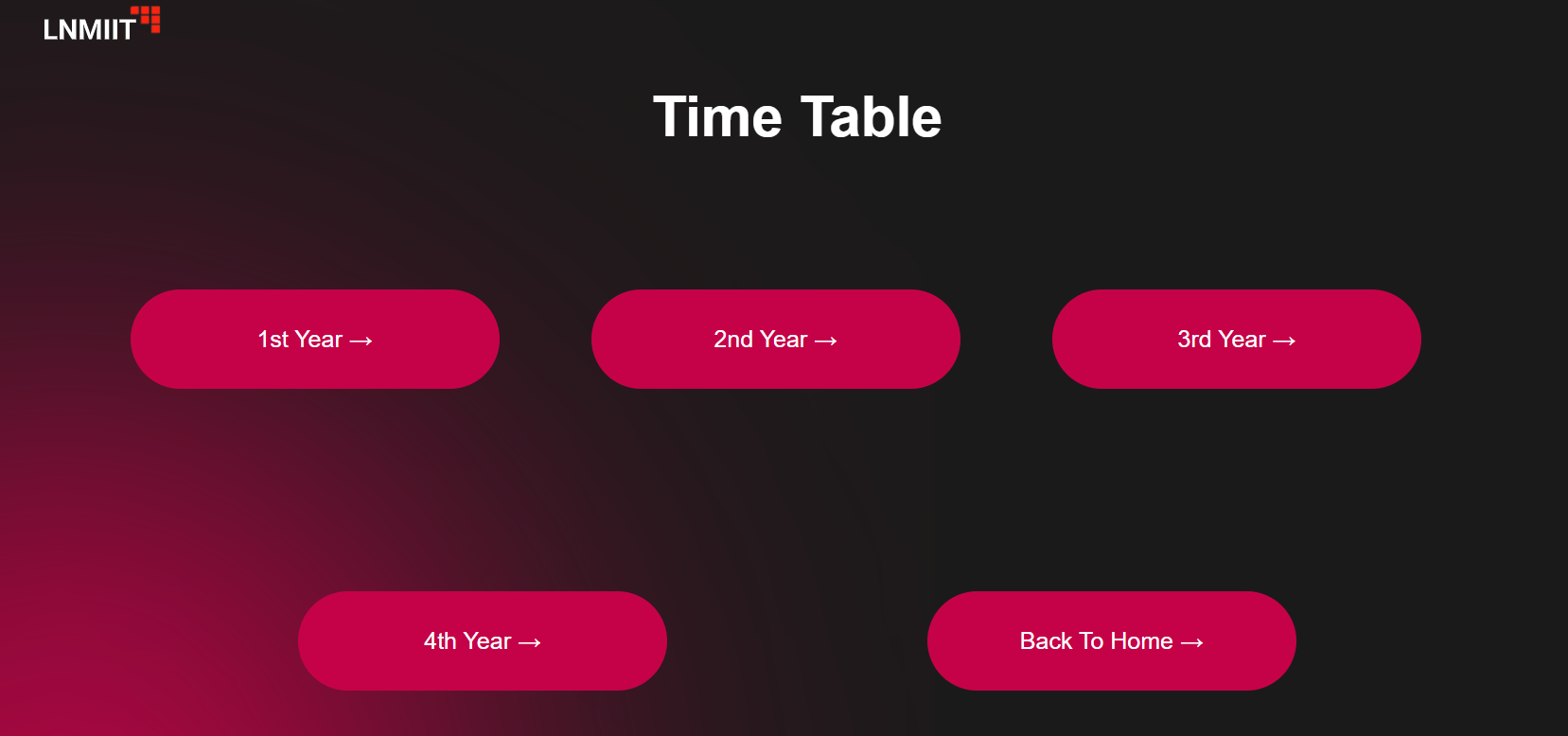


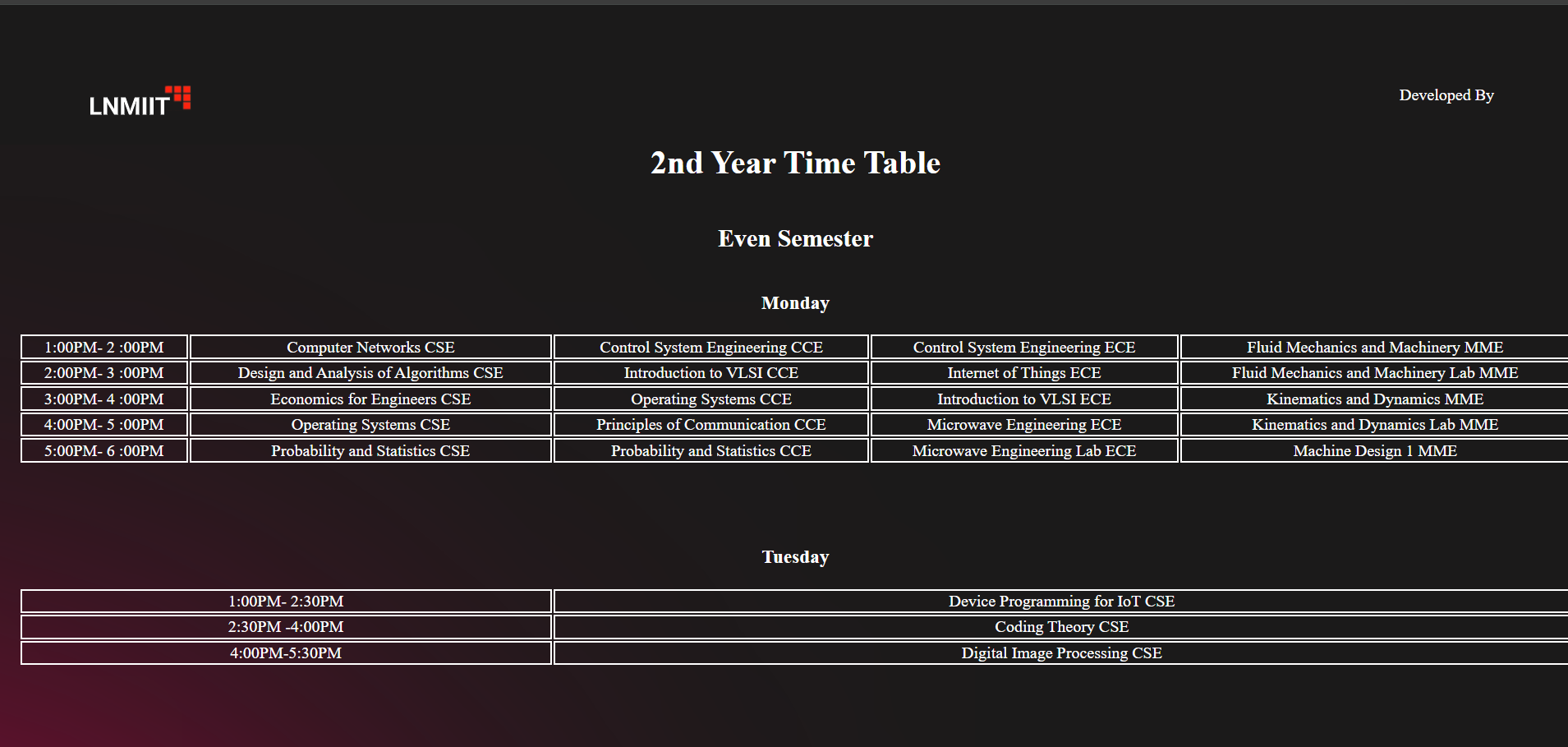
**Admin Login:**





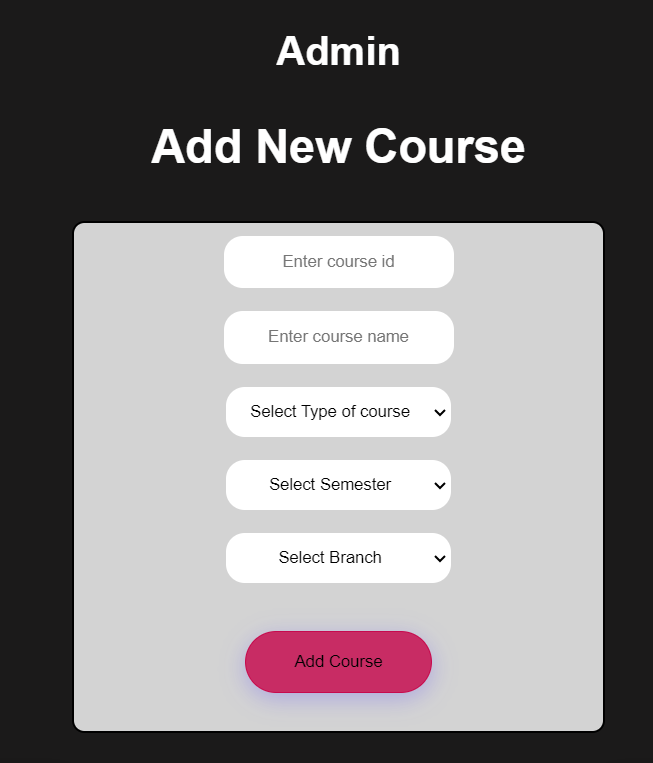
**Test cases:**

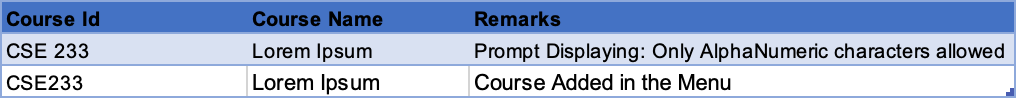
**Time table generation**





**Add Delete Course:**

****

**Test cases:**

**WHITE BOX TESTING:**

**ADMIN-CONTROLLER MODULE**

1. **Function securePassword:**

const securePassword= async(password)=>{

try {

const passwordHash= await bcrypt.hash(password,10);

return passwordHash;

} catch (error) {

console.log(error.message);

}

}

#### Testing secure Password Function:

**Test Case 1**

**Input:**

* password: "test123"

**Expected Output:**

* The function should return a hashed password.

**2.Function addUserMail:**

const addUserMail= async(name,email, password, user\_id)=>{

try {

const transporter= nodemailer.createTransport({

host: 'smtp.gmail.com',

port:587,

secure:false,

requireTLS:true,

auth:{

user:config.emailUser,

pass:config.emailPassword

}

});

const mailOptions={

from:config.emailUser,

to:email,

subject:"Admin add you and verify your mail",

html:'<p>Hello '+name+' click here to <a href="http://127.0.0.1:3000/verify?id='+user\_id+'"> verify</a> your mail. </p> <br><br> <b>Email- </b> '+email+' <br><b>Password -</b>'+password+''

}

transporter.sendMail(mailOptions,function(error,info){

if(error){

console.log(error);

}

else{

console.log("email sent",info.response);

}

})

} catch (error) {

console.log(error.message);

}

}

#### Testing addUserMail Function:

**Test Case 1:**

**Input**

* name: "John Doe"
* email: "john.doe@example.com"
* password: "test123"
* user\_id: "12345"

**Expected Outcome**

* The function should send a verification link to the provided email address.

**3.Function loadLogin:**

const loadLogin= async(req,res)=>{

try {

res.render('login');

} catch (error) {

console.log(error.message);

}

}

#### 

#### Testing loadLogin Function:

**Test Case 1:**

**Input**

* req: Mocked request object
* res: Mocked response object

**Expected Outcome**

* The function should render the 'login' view.

**4.Function verifyLogin:**

const verifyLogin = async(req,res)=>{

try {

const email=req.body.email;

const password= req.body.password;

const userData = await User.findOne({email:email});

if(userData){

const passwordMatch=await bcrypt.compare(password,userData.password);

if(passwordMatch){

if(userData.is\_admin === 0){

res.render('login',{message:"invalid mail or passwrord"});

}

else{

req.session.user\_id= userData.\_id;

res.redirect("/admin/home");

}

}

else{

res.render('login',{message:"invalid mail or password"});

}

}

else{

res.render('login',{message:"invalid mail or password"});

}

} catch (error) {

console.log(error.message);

}

}

#### 

#### 

#### Testing verifyLogin Function:

**Test Case 1:**

**Input**

* Valid email and password for an admin user

**Expected Outcome**

* The function should redirect to "/admin/home".

**Test Case 2:**

**Input**

* Invalid email or password

**Expected Outcome**

* The function should render the 'login' view with an error message.

**5.Function loadDashboard**

const loadDashboard= async(req,res)=>{

try {

const userData = await User.findById({\_id:req.session.user\_id});

if(userData.is\_admin==0){

res.render('404',{message:"you are not admin"});

}

res.render('home',{admin:userData});

} catch (error) {

console.log(error.message);

}

}

#### Testing loadDashboard Function:

**Test Case 1:**

**Input**

* Valid session with an admin user

**Expected Outcome**

* The function should render the 'home' view for the admin.

**6.Function logout:**

const logout=async(req,res)=>{

try {

req.session.destroy();

res.redirect('/admin');

} catch (error) {

console.log(error.message);

}

}

#### Testing logout Function:

**Test Case 1:**

**Input**

* req: Mocked request object
* res: Mocked response object

**Expected Outcome**

* The function should destroy the session and redirect to "/admin".

**7.Function newUserLoad:**

const newUserLoad = async(req,res)=>{

try {

const userData = await User.findById({\_id:req.session.user\_id});

if(userData.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('new-user');

} catch (error) {

console.log(error.message);

}

}

#### 

#### Testing newUserLoad Function:

**Test Case 1:**

**Input**

* Valid session with an admin user

**Expected Outcome**

* The function should render the 'new-user' view.

**8. Function addUser**

const addUser = async(req,res)=>{

try {

const name = req.body.name;

const email = req.body.email;

const rollno=req.body.rollno;

const semester = req.body.semester;

const branch = req.body.branch;

const password = randomstring.generate(8);

const spassword = await securePassword(password);

const user = new User({

name:name,

rollno:rollno,

email:email,

semester:semester,

branch:branch,

password:spassword,

is\_admin:0,

is\_varified:1,

});

const userData = await user.save();

if(userData)

{

addUserMail(name, email, password, userData.\_id);

res.redirect('/admin/home');

}

else

{

res.render('new-user',{message:"Something wrong"});

}

} catch (error) {

console.log(error.message);

}

}

#### Testing addUser Function:

**Test Case 1:**

**Input**

* Valid input data for a new user

**Expected Outcome**

* The function should add a new user, send a verification email, and redirect to "/admin/home".

**Test Case 2:**

**Input**

* Invalid input data

**Expected Outcome**

* The function should render the 'new-user' view with an error message.

**9. Function addcourseLoad**

const addcourseLoad= async(req,res)=>{

try {

const userData = await User.findById({\_id:req.session.user\_id});

const courseData = await Course.find();

if(userData.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('course',{admin:userData, course:courseData});

} catch (error) {

console.log(error.message);

}

}

#### Testing addcourseLoad Function:

**Test Case 1:**

**Input**

* Valid session with an admin user

**Expected Outcome**

* The function should render the 'course' view with course data

**10. Function addcourse:**

const addcourse = async(req,res)=>{

try {

const index = req.body.index;

const name = req.body.name;

const sem = req.body.sem;

const branch = req.body.branch;

const Problem = new Course({

coursename:name,

courseid:index,

semester:sem,

branch:branch

});

const problemData1 = await Problem.save();

const problemData = await Course.find();

res.render('course',{message:"added successfully",problem:problemData})

} catch (error) {

console.log(error.message);

}

}

#### Testing addcourse Function:

**Test Case 1:**

**Input**

* Valid input data for a new course

**Expected Outcome**

* The function should add a new course, and render the 'course' view with a success message and updated course data.

**11. Function deletecourse:**

const deletecourse = async(req,res)=>{

try {

const userData = await User.findById({\_id:req.session.user\_id});

if(userData.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

const id = req.query.id;

await Course.deleteOne({ \_id:id });

res.redirect('/admin/home');

} catch (error) {

console.log(error.message);

}

}

#### Testing deletecourse Function:

**Test Case 1:**

**Input**

* Valid session with an admin user, course ID to delete

**Expected Outcome**

* The function should delete the specified course and redirect to "/admin/home

**12. Function ttgenLoad**

const ttgenLoad= async(req,res)=>{

try {

const userData1 = await User.findById({\_id:req.session.user\_id});

const coursedata1= await Course.find();

const coursedata2= await Programcourse.find();

const coursedata3= await Othercourse.find();

const coursedata4= await Facultycourse.find();

if(userData1.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('ttgen',{admin:userData1, corecourse:coursedata1, programcourse:coursedata2, othercourse:coursedata3, facultycourse:coursedata4});

} catch (error) {

console.log(error.message);

}

}

#### 

#### 

#### Testing ttgenLoad Function:

**Test Case 1:**

**Input**

* Valid session with an admin user

**Expected Outcome**

* The function should render the 'ttgen' view with course data.

**13. Functions ttgenLoady20, ttgenLoady21, ttgenLoady22, ttgenLoady23,**

const ttgenLoady20= async(req,res)=>{

try {

const userData1 = await User.findById({\_id:req.session.user\_id});

const coursedata11= await Course.find({branch:"CSE",semester:8});

const coursedata12= await Course.find({branch:"ECE",semester:8});

const coursedata13= await Course.find({branch:"CCE",semester:8});

const coursedata14= await Course.find({branch:"MME",semester:8});

// console.log(course)

const coursedata21= await Programcourse.find({branch:"CSE",semester:8});

const coursedata22= await Programcourse.find({branch:"ECE",semester:8});

const coursedata23= await Programcourse.find({branch:"CCE",semester:8});

const coursedata24= await Programcourse.find({branch:"MME",semester:8});

// const coursedata2= await Programcourse.find();

const coursedata3= await Othercourse.find();

const coursedata4= await Facultycourse.find();

if(userData1.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('ttgeny20',{admin:userData1, corecoursecse:coursedata11,corecourseece:coursedata12,corecourseme:coursedata14,corecoursecce:coursedata13, programcoursecse:coursedata21, programcoursecce:coursedata22, programcourseece:coursedata23, programcourseme:coursedata24, othercourse:coursedata3, facultycourse:coursedata4});

} catch (error) {

console.log(error.message);

}

}

const ttgenLoady21= async(req,res)=>{

try {

const userData1 = await User.findById({\_id:req.session.user\_id});

const coursedata11= await Course.find({branch:"CSE",semester:6});

const coursedata12= await Course.find({branch:"ECE",semester:6});

const coursedata13= await Course.find({branch:"CCE",semester:6});

const coursedata14= await Course.find({branch:"MME",semester:6});

// console.log(course)

const coursedata21= await Programcourse.find({branch:"CSE",semester:6});

const coursedata22= await Programcourse.find({branch:"ECE",semester:6});

const coursedata23= await Programcourse.find({branch:"CCE",semester:6});

const coursedata24= await Programcourse.find({branch:"MME",semester:6});

// const coursedata2= await Programcourse.find();

const coursedata3= await Othercourse.find();

const coursedata4= await Facultycourse.find();

if(userData1.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('ttgeny21',{admin:userData1, corecoursecse:coursedata11,corecourseece:coursedata12,corecourseme:coursedata14,corecoursecce:coursedata13, programcoursecse:coursedata21, programcoursecce:coursedata22, programcourseece:coursedata23, programcourseme:coursedata24, othercourse:coursedata3, facultycourse:coursedata4});

} catch (error) {

console.log(error.message);

}

}

const ttgenLoady22= async(req,res)=>{

try {

const userData1 = await User.findById({\_id:req.session.user\_id});

const coursedata11= await Course.find({branch:"CSE",semester:4});

const coursedata12= await Course.find({branch:"ECE",semester:4});

const coursedata13= await Course.find({branch:"CCE",semester:4});

const coursedata14= await Course.find({branch:"MME",semester:4});

// console.log(course)

const coursedata21= await Programcourse.find({branch:"CSE",semester:4});

const coursedata22= await Programcourse.find({branch:"ECE",semester:4});

const coursedata23= await Programcourse.find({branch:"CCE",semester:4});

const coursedata24= await Programcourse.find({branch:"MME",semester:4});

// const coursedata2= await Programcourse.find();

const coursedata3= await Othercourse.find();

const coursedata4= await Facultycourse.find();

if(userData1.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('ttgeny22',{admin:userData1, corecoursecse:coursedata11,corecourseece:coursedata12,corecourseme:coursedata14,corecoursecce:coursedata13, programcoursecse:coursedata21, programcoursecce:coursedata22, programcourseece:coursedata23, programcourseme:coursedata24, othercourse:coursedata3, facultycourse:coursedata4});

} catch (error) {

console.log(error.message);

}

}

const ttgenLoady23= async(req,res)=>{

try {

const userData1 = await User.findById({\_id:req.session.user\_id});

const coursedata11= await Course.find({branch:"CSE",semester:2});

const coursedata12= await Course.find({branch:"ECE",semester:2});

const coursedata13= await Course.find({branch:"CCE",semester:2});

const coursedata14= await Course.find({branch:"MME",semester:2});

// console.log(course)

const coursedata21= await Programcourse.find({branch:"CSE",semester:2});

const coursedata22= await Programcourse.find({branch:"ECE",semester:2});

const coursedata23= await Programcourse.find({branch:"CCE",semester:2});

const coursedata24= await Programcourse.find({branch:"MME",semester:2});

// const coursedata2= await Programcourse.find();

const coursedata3= await Othercourse.find();

const coursedata4= await Facultycourse.find();

if(userData1.is\_admin==0){

res.render('404',{message:"you are not an admin"});

}

res.render('ttgeny23',{admin:userData1, corecoursecse:coursedata11,corecourseece:coursedata12,corecourseme:coursedata14,corecoursecce:coursedata13, programcoursecse:coursedata21, programcoursecce:coursedata22, programcourseece:coursedata23, programcourseme:coursedata24, othercourse:coursedata3, facultycourse:coursedata4});

} catch (error) {

console.log(error.message);

}

}

#### Testing ttgenLoady20 function ttgenLoady21, ttgenLoady22, ttgenLoady23 Functions:

**Test Case 1:**

**Input**

* Valid session with an admin user

**Expected Outcome**

* The function should render the respective 'ttgenyXX' view with course data.

**WHITE BOX TESTING:**

**USER-CONTROLLER MODULE**

1. **Function securePassword**

const securePassword = async (password) => {

try {

const passwordHash = await bcrypt.hash(password, 10);

return passwordHash;

} catch (error) {

console.log(error.message);

}

}

#### Testing secure Password Function:

**Test Case 1**

**Input:**

* Password: "test123"

**Expected Output:**

* The function should return a hashed password.

1. **Function senderVerifyMail**

const sendVerifyMail = async (name, email, user\_id) => {

try {

const transporter = nodemailer.createTransport({

host: 'smtp.gmail.com',

port: 587,

secure: false,

requireTLS: true,

auth: {

user: config.emailUser,

pass: config.emailPassword

}

});

const mailOptions = {

from: config.emailUser,

to: email,

subject: "Verification Mail",

html: '<p>Hello ' + name + ' click here to <a href="http://127.0.0.1:3000/verify?id=' + user\_id + '"> verify</a> your mail. </p>'

}

transporter.sendMail(mailOptions, function (error, info) {

if (error) {

console.log(error);

}

else {

console.log("email sent", info.response);

}

})

} catch (error) {

console.log(error.message);

}

}

### Testing sendVerifyMail Function:

**Test Case 1:**

**Input:**

* + 1. Name: "John Doe"
    2. Email: "[john.doe@example.com](mailto:john.doe@example.com)"
    3. User\_id: "12345"

**Expected Outcome:**

The function should send an email with a verification link to the provided

Email adress.

1. **Function sendResetPasswordMail**

const sendResetPasswordMail = async (name, email, token) => {

try {

const transporter = nodemailer.createTransport({

host: 'smtp.gmail.com',

port: 587,

secure: false,

requireTLS: true,

auth: {

user: config.emailUser,

pass: config.emailPassword

}

});

const mailOptions = {

from: config.emailUser,

to: email,

subject: "Reset Password",

html: '<p>Hello ' + name + ' click here to <a href="http://127.0.0.1:3000/forget-password?token=' + token + '"> Reset </a> your password. </p>'

}

transporter.sendMail(mailOptions, function (error, info) {

if (error) {

console.log(error);

}

else {

console.log("email sent", info.response);

}

})

} catch (error) {

console.log(error.message);

}

}

### Testing sendResetPasswordMail Function:

**Test Case 1:**

**Input:**

* 1. Name: "John Doe"
  2. Email: "john.doe@example.com"
  3. Token: "randomToken"

**Expected Outcome:**

a.The function should send an email with a password reset link to the provided email address.

1. **Function loadRegister**

const loadRegister = async (req, res) => {

try {

res.render('registration');

}

catch (error) {

console.log(error.message);

}

}

### Testing loadRegister Function:

Test Case 1:

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

1. The function should render the 'registration' view.
2. **Function insertUser**

const insertUser = async (req, res) => {

try {

const spassword = await securePassword(req.body.password);

const user = new User({

name: req.body.name,

email: req.body.email,

rollno: req.body.rollno,

branch: req.body.branch,

semester: req.body.semester,

password: spassword,

is\_admin: 0,

is\_faculty:0

});

const userData = await user.save();

if (userData) {

sendVerifyMail(req.body.name, req.body.email, userData.\_id);

res.render('registration', { message: "Registration successful. Verify your mail" });

}

else {

res.render('registration', { message: "Rgistration failed." });

}

} catch (error) {

console.log(error.message);

}

}

Testing insertUser Function

**Test Case 1:**

**Input:**

* 1. Mocked request object with user details
  2. Mocked response object

**Expected Outcome:**

a. The function should insert a new user, send a verification email, and render the 'registration' view with a success message.

1. **Function verifyMail**

const verifyMail = async (req, res) => {

try {

const updateInfo = await User.updateOne({ \_id: req.query.id }, { $set: { is\_varified: 1 } });

console.log(updateInfo);

res.render("email-verified");

} catch (error) {

console.log(error.message)

}

}

### Testing verifyMail Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object with user\_id parameter
  2. Mocked response object

**Expected Outcome:**

1. The function should update the user's verification status and render the 'email-verified' view.
2. **Function profileLoad**

const profileLoad = async (req, res) => {

try {

const userData1 = await User.findOne({ \_id: req.session.user\_id });

const userData2 = await Leaderboard.findOne({ email: userData1.email });

res.render('profile', { user: userData1, leaderboard: userData2 });

} catch (error) {

console.log(error.message);

}

}

### Testing profileLoad Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

The function should render the 'profile' view with user and leaderboard data.

1. **Function courseregLoad**

const courseregLoad = async (req, res) => {

try {

const userData1 = await User.findOne({ \_id: req.session.user\_id });

const userData2= await Coursereg.findOne({rollno:userData1.rollno});

if(userData2){

res.render("home",{message:"you have done the course registration", user: userData1,coursereg:userData2.corecourses,coursereg2:userData2.programcourses})

}

const coursedata1 = await Course.find({ semester: parseInt(userData1.semester) ,branch:userData1.branch});

const coursedata2 = await Programcourse.find({ semester: parseInt(userData1.semester) ,branch:userData1.branch});

const coursedata3 = await Othercourse.find({ semester: parseInt(userData1.semester) ,branch:userData1.branch});

res.render('coursereg', { user: userData1, corecourse: coursedata1, programcourse: coursedata2, othercourse: coursedata3 });

} catch (error) {

console.log(error.message);

}

}

### Testing courseregLoad Function:

**Test Case 1:**

**Input:**

* + Mocked request object
  + Mocked response object

**Expected Outcome:**

* + The function should render the 'coursereg' view with user and course data.

1. **Function coursereg:**

const coursereg = async (req, res) => {

try {

const userData1 = await User.findOne({ \_id: req.session.user\_id });

// console.log(userData1.branch);

const coursedata1 = await Course.find({ semester: parseInt(userData1.semester) ,branch:userData1.branch });

const coursedata2 = await Programcourse.find({ semester: parseInt(userData1.semester),branch:userData1.branch });

const coursedata3 = await Othercourse.find({ semester: parseInt(userData1.semester),branch:userData1.branch });

// console.log(coursedata1);

// console.log(req.body.coreCourses);

if (req.body.coreCourses === undefined || req.body.coreCourses.length < coursedata1.length) {

res.render('coursereg', { message: "select all core courses", user: userData1, corecourse: coursedata1, programcourse: coursedata2, othercourse: coursedata3 });

}

else {

const coursereg = new Coursereg({

name: userData1.name,

rollno: userData1.rollno,

corecourses: req.body.coreCourses,

programcourses: req.body.programCourses,

othercourses: req.body.otherCourses,

});

const userData = await coursereg.save();

res.render('coursereg', { message: "reg successful", user: userData1, corecourse: coursedata1, programcourse: coursedata2, othercourse: coursedata3 });

}

} catch (error) {

console.log(error.message);

}

}

### 

### Testing coursereg Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object with selected courses
  2. Mocked response object

**Expected Outcome:**

a. The function should insert course registration data and render the 'coursereg' view with a success message.

1. **Function loginLoad:**

const loginLoad = async (req, res) => {

try {

res.render('login');

} catch (error) {

console.log(error.message);

}

}

### Testing loginLoad Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

1. The function should render the 'login' view.
2. **Function verifyLogin:**

const verifyLogin = async (req, res) => {

try {

const email = req.body.email;

const password = req.body.password;

const userData = await User.findOne({ email: email });

if (userData) {

const passwordMatch = await bcrypt.compare(password, userData.password);

if (passwordMatch) {

if (userData.is\_varified === 0) {

res.render('login', { message: "Please verify your mail." });

}

else {

req.session.user\_id = userData.\_id;

if(userData.is\_faculty==0){

res.redirect('/home');

}

else{

res.redirect('/facultyhome');

}

}

}

else {

res.render('login', { message: "Email and password is incorrect" });

}

}

else {

res.render('login', { message: "Email and password is incorrect" });

}

} catch (error) {

console.log(error.message);

}

}

### Testing verifyLogin Function:

**Test Case 1:**

**Input:**

* + Mocked request object with valid email and password for a non-faculty user
  + Mocked response object

**Expected Outcome:**

* + The function should redirect to "/home".

**Test Case 2:**

**Input:**

* + Mocked request object with invalid email or password
  + Mocked response object

**Expected Outcome:**

* + The function should render the 'login' view with an error message.

1. **Function loadHome:**

const loadHome = async (req, res) => {

try {

const userData = await User.findById({ \_id: req.session.user\_id });

const userData2 = await Coursereg.findOne({ rollno:userData.rollno});

// console.log(userData2);

if(userData.is\_faculty==1){

res.redirect('/facultyhome');

}

console.log(userData2);

if(userData2===null){

res.render('home', { user: userData ,coursereg:{},coursereg2:{}});

}else{

res.render('home', { user: userData ,coursereg:userData2.corecourses,coursereg2:userData2.programcourses});

}

} catch (error) {

console.log(error.message);

}

}

### Testing loadHome Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

1. The function should render the 'home' view with user and course registration data.
2. **Function loadfacultyHome:**

const loadfacultyHome = async (req, res) => {

try {

const userData = await User.findById({ \_id: req.session.user\_id });

const userData2 = await Coursereg.findOne({ rollno:userData.rollno});

// console.log(userData2);

res.render('facultyhome', { user: userData,coursereg:userData2.corecourses,coursereg2:userData2.programcourses });

} catch (error) {

console.log(error.message);

}

}

### Testing loadfacultyHome Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

1. The function should render the 'faculty home' view with user and course registration data.
2. **Function userLogout**

const userLogout = async (req, res) => {

try {

req.session.destroy();

res.redirect('/');

} catch (error) {

console.log(error.message);

}

}

### 

### 

### Testing userLogout Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object
  2. Mocked response object

**Expected Outcome:**

1. The function should destroy the session and redirect to "/".
2. **Function forgetLoad:**

const forgetLoad = async (req, res) => {

try {

res.render('forget');

} catch (error) {

console.log(error.message);

}

}

### Testing forgetLoad Function:

**Test Case 1:**

**Input:**

* + Mocked request object
  + Mocked response object

**Expected Outcome:**

* + The function should render the 'forget' view.

1. **Function forgetVerify:**

const forgetVerify = async (req, res) => {

try {

const email = req.body.email;

const userData = await User.findOne({ email: email });

if (userData) {

const randomString = randomstring.generate();

const updatedData = await User.updateOne({ email: email }, { $set: { token: randomString } });

sendResetPasswordMail(userData.name, userData.email, randomString);

res.render('forget', { message: "Please check your mail to reset your password." });

}

else {

res.render('forget', { message: "User email not found" });

}

} catch (error) {

console.log(error.message);

}

}

### Testing forgetVerify Function:

**Test Case 1:**

**Input:**

* + Mocked request object with a valid email
  + Mocked response object

**Expected Outcome:**

* + The function should send a reset password email and render the 'forget' view with a success message.

**Test Case 2:**

**Input:**

* 1. Mocked request object with an invalid email
  2. Mocked response object

**Expected Outcome:**

a. The function should render the 'forget' view with an error message.

1. **Function forgetPasswordLoad:**

const forgetPasswordLoad = async (req, res) => {

try {

const token = req.query.token;

const tokenData = await User.findOne({ token: token });

if (tokenData) {

res.render('forget-password', { user\_id: tokenData.\_id });

}

else {

res.render('404', { message: "Token is invalid." });

}

} catch (error) {

console.log(error.message);

}

}

### **Testing forgetPasswordLoad Function:**

**Test Case 1:**

**Input:**

* + Mocked request object with a valid token
  + Mocked response object

**Expected Outcome:**

* + The function should render the 'forget-password' view with the user\_id parameter.

**Test Case 2:**

**Input:**

* 1. Mocked request object with an invalid token
  2. Mocked response object

**Expected Outcome:**

a. The function should render the '404' view with an error message.

1. **Function resetPassword:**

const resetPassword = async (req, res) => {

try {

const password = req.body.password;

const user\_id = req.body.user\_id;

const secure\_password = await securePassword(password);

const updatedData = await User.findByIdAndUpdate({ \_id: user\_id }, { $set: { password: secure\_password, token: '' } });

res.redirect("/");

} catch (error) {

console.log(error.message);

}

}

### Testing resetPassword Function:

**Test Case 1:**

**Input:**

* 1. Mocked request object with a valid password and user\_id
  2. Mocked response object

**Expected Outcome:**

a. The function should update the user's password and redirect to "/".

1. **Function verificationLink:**

const verificationLink = async (req, res) => {

try {

res.render('verification')

} catch (error) {

console.log(error.message);

}

}

Testing verificationLink:

**Test Case 1:**

* **Input:**
  + Mocked request object
  + Mocked response object
* **Expected Outcome:**
  + The function should render the 'verification' view.

1. **Function sendVerificationLink**

const sendVerificationLink = async (req, res) => {

try {

const email = req.body.email;

const userData = await User.findOne({ email: email });

if (userData) {

sendVerifyMail(userData.name, userData.email, userData.\_id);

res.render('verification', { message: "Verification link sent on your mail, please check" });

}

else {

res.render('verification', { message: "This email does not exist" });

}

} catch (error) {

console.log(error.message);

}

}

### Testing sendVerificationLink Function:

**Test Case 1:**

**Input:**

* + Mocked request object with a valid email
  + Mocked response object

**Expected Outcome:**

* + The function should send a verification email and render the 'verification' view with a success message.

**Test Case 2:**

**Input:**

* + Mocked request object with an invalid email
  + Mocked response object

**Expected Outcome:**

* + The function should render the 'verification' view with an error message.