

# **ISDL LAB PROJECT**

## **TESTING DOCUMENT**

***Topic:***

**COURSE REGISTRATION AND TIMETABLE GENERATION**

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



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

# **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 deletcourse:

```
const deletcourse = 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 deletcourse 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, programcourseeece: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,
programcourseece: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,
corecoursece:coursedata11,corecoursece:coursedata12,corecourseme:coursedata14
,corecoursece:coursedata13, programcoursece:coursedata21,
programcoursece:coursedata22, programcoursece: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.

## 2. 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:

- i. Name: "John Doe"
- ii. Email: "[john.doe@example.com](mailto:john.doe@example.com)"
- iii. User\_id: "12345"

#### Expected Outcome:

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

### 3. 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:**

- a. Name: "John Doe"
- b. Email: "john.doe@example.com"
- c. Token: "randomToken"

### **Expected Outcome:**

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

## **4. Function loadRegister**

```
const loadRegister = async (req, res) => {  
  try {  
    res.render('registration');  
  }  
  catch (error) {  
    console.log(error.message);  
  }  
}
```

### Testing loadRegister Function:

#### Test Case 1:

##### **Input:**

- a. Mocked request object
- b. Mocked response object

### **Expected Outcome:**

- a. The function should render the 'registration' view.

## 5. 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: "Registration failed." });
    }
  } catch (error) {
    console.log(error.message);
  }
}
```

## Testing insertUser Function



## Test Case 1:

### Input:

- a. Mocked request object with user details
- b. 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.

## 6. 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:

- a. Mocked request object with user\_id parameter
- b. Mocked response object

### Expected Outcome:

- a. The function should update the user's verification status and render the 'email-verified' view.

## 7. 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:

- a. Mocked request object
- b. Mocked response object

#### Expected Outcome:

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

## 8. 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.

## 9. 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:

- a. Mocked request object with selected courses
- b. Mocked response object

### Expected Outcome:

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

## 10. Function loginLoad:

```
const loginLoad = async (req, res) => {  
  try {  
    res.render('login');  
  } catch (error) {  
    console.log(error.message);  
  }  
}
```

## Testing loginLoad Function:

## Test Case 1:

### Input:

- a. Mocked request object
- b. Mocked response object

### Expected Outcome:

- a. The function should render the 'login' view.

## 11. 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.

## 12. 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:

- a. Mocked request object
- b. Mocked response object

#### Expected Outcome:

- a. The function should render the 'home' view with user and course registration data.



### 13. 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:

- a. Mocked request object
- b. Mocked response object

##### Expected Outcome:

- a. The function should render the 'faculty home' view with user and course registration data.

## 14. 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:

- a. Mocked request object
- b. Mocked response object

## Expected Outcome:

- a. The function should destroy the session and redirect to "/".

## 15. 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.

## 16. 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:

- a. Mocked request object with an invalid email

- b. Mocked response object

### **Expected Outcome:**

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

## **17. 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:

- a. Mocked request object with an invalid token
- b. Mocked response object

### Expected Outcome:

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

## 18. 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:

- a. Mocked request object with a valid password and user\_id
- b. Mocked response object

**Expected Outcome:**

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

## 19. 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.

## 20. 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:



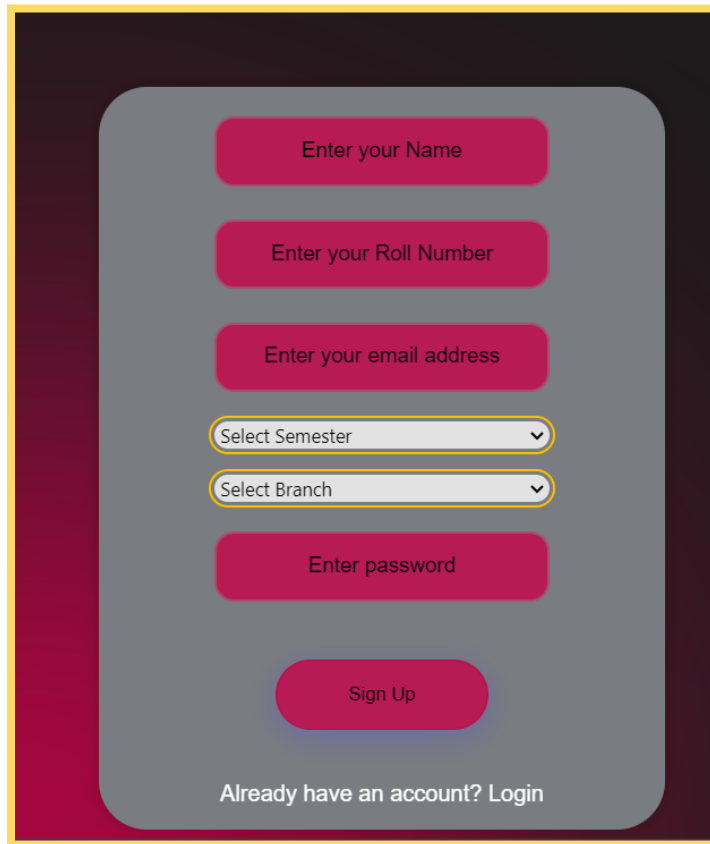
- The function should render the 'verification' view with an error message.

**BLACK**

**BOX**

**TESTING**

# Account Sign-up page

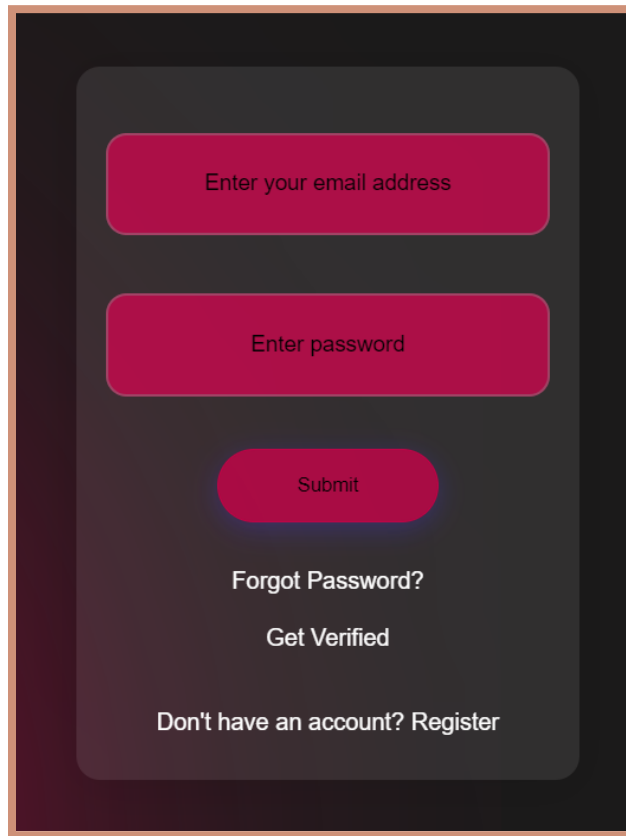
A screenshot of an account sign-up page. The form is centered on a dark background with a yellow border. It contains several input fields: 'Enter your Name', 'Enter your Roll Number', 'Enter your email address', 'Select Semester' (a dropdown menu), 'Select Branch' (a dropdown menu), 'Enter password', and a 'Sign Up' button. At the bottom, there is a link that says 'Already have an account? Login'.

## Test Cases:

Name	Roll Number	Email	Password	State	Remark
V@ib#@v K#amesr@	21UCS224	21ucs224@lnmit.ac.in	Vaibhav@123	Unregistered	Message Saying: name should not contain special characters.Credentials not accepted
Vaibhav Khamesra	21UCS224	21ucs224lnmit.ac.in	Vaibhav@123	Unregistered	Message Saying: Please include an @ in the email.
Vaibhav Khamesra	21UCS224	21ucs224@gmail.com	Vaibhav@123	Unregistered	Message Saying: Invalid Email
Vaibhav Khamesra	21UCS224	21ucs224@lnmit.ac.in	Vaibhav@123	Unregistered	Message Saying: Invalid Email
Vaibhav Khamesra	21UCS224	21ucs224@lnmit.ac.in	123	Unregistered	Message Saying: At least 8 characters long
Vaibhav Khamesra	21UCS224	21ucs224@lnmit.ac.in	Vaibhav@123	Unregistered	Prompt Saying: Registration Successful. Verify your Email
Vaibhav Khamesra	21UCS224	21ucs224@lnmit.ac.in	Vaibhav@123	Registered	Prompt Saying: User Already Registered

Email Verification Link received after successful validation of detail

# Account Log-In page

A mockup of an account login page. It features a dark background with a central light gray rounded rectangle containing the login fields. The fields are: a pink rounded rectangle with the text "Enter your email address", another pink rounded rectangle with the text "Enter password", and a pink rounded rectangle with the text "Submit". Below these fields are the links "Forgot Password?", "Get Verified", and "Don't have an account? Register" in white text.

Enter your email address

Enter password

Submit

[Forgot Password?](#)

[Get Verified](#)

[Don't have an account? Register](#)

## Test Cases:

Email	Password	Verified	Remarks
21ucs224@lnmiit.ac.in	Vaibhav@123	Yes	Login successful
21ucs224@lnmiit.ac.in	vaibhav	Yes	Prompt Saying: Email and password is incorrect
21ucs223@lnmiit.ac.in	Vaibhav@123	Yes	Prompt Saying: Email and password are incorrect
21ucs039@lnmiit.ac.in	Ashu#123	No	Prompt Saying: Please verify your email

## Forgot Password:

# Forgot Password

Enter email

Send reset link

[Go to login](#)

Email	Remarks
21ucs224@lnmiit.ac.in	Email Verified and a link was recieved
vaibhavkhamesra643@gmail.com	Prompt Saying: User Email not found.



coursereglnm@gmail.com

to me ▾

Hello Vaibhav Khamesra click here to [Reset](#) your password.

# Login in your account

Enter New Password

Reset Password

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

S. No.	New Password	Remark
1	password@123	Password Reset successful, the user is redirected to the Signup page
2.	password	Prompt Saying: At least 8 characters long at least one special character at least one digit.

## Course Registration Module:

### Course Register

Core Courses



**Compiler Design**

CSE437

6

CSE

✓



**Machine Learning and Pattern Recognition**

CSE6052


6

CSE

✓

Program Electives

Program Electives




**Real Time Systems**

CSE3021

6

CSE




**Social Network Analysis**

CSE3132

6

CSE




**Mining of Massive Datasets**

CSE3152

6

CSE



**Blockchain Foundation & Smart Contract**

CSE3222

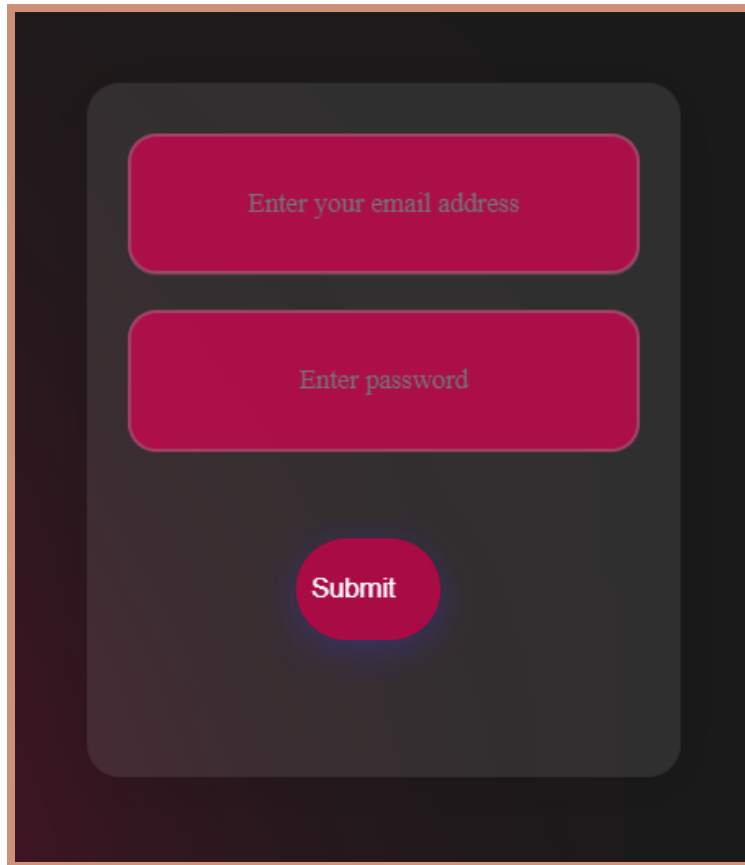
6

CSE

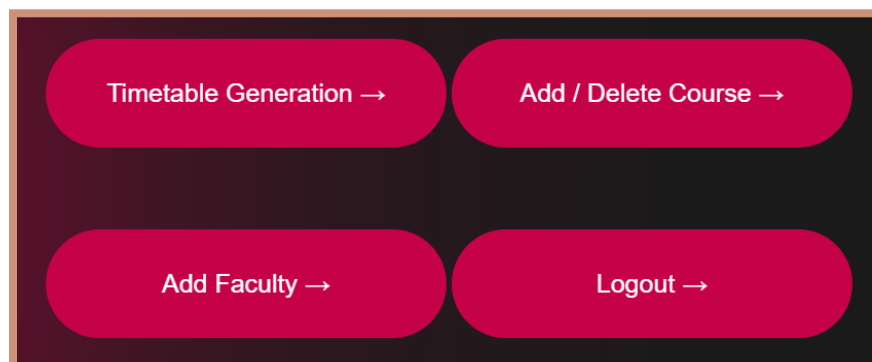
## Test Cases:

Number of core courses selected	Number of program electives selected	Current State	Remarks
1 out of 2 core course selected	2	Unregistered	Prompt saying: Select All core courses
2 out 2 core courses selected	2	Unregistered	Prompt saying: Course Registration successful.
2 out 2 core courses selected	2	Registered	Prompt saying: You are already registered
2 out 2 core courses selected	3	Unregistered	Prompt saying: Course Registration successful.
2 out 2 core courses selected	4	Unregistered	Course Limit Exceeded

## Admin Login:



A screenshot of an admin login interface. It features a dark gray background with a central light gray rounded rectangle. Inside this rectangle, there are two red rounded rectangular input fields. The first field contains the placeholder text "Enter your email address" and the second field contains "Enter password". Below these fields is a red rounded rectangular button with the text "Submit" in white.



A screenshot of an admin dashboard showing four red rounded rectangular buttons arranged in a 2x2 grid. Each button has white text and a right-pointing arrow. The buttons are labeled: "Timetable Generation →", "Add / Delete Course →", "Add Faculty →", and "Logout →".

## Test cases:

Email address	Password	Remark
21ucs224@lnmiit.ac.in	password	Not Validated
21ucs155@lnmiit.ac.in	Pranjal@2310	Validated



# Time Table

[1st Year →](#)[2nd Year →](#)[3rd Year →](#)[4th Year →](#)[Back To Home →](#)

## Time table generation

### 2nd Year Time Table

#### Even Semester

##### Monday

1:00PM- 2 :00PM	Computer Networks CSE	Control System Engineering CCE	Control System Engineering ECE	Fluid Mechanics and Machinery MME
2:00PM- 3 :00PM	Design and Analysis of Algorithms CSE	Introduction to VLSI CCE	Internet of Things ECE	Fluid Mechanics and Machinery Lab MME
3:00PM- 4 :00PM	Economics for Engineers CSE	Operating Systems CCE	Introduction to VLSI ECE	Kinematics and Dynamics MME
4:00PM- 5 :00PM	Operating Systems CSE	Principles of Communication CCE	Microwave Engineering ECE	Kinematics and Dynamics Lab MME
5:00PM- 6 :00PM	Probability and Statistics CSE	Probability and Statistics CCE	Microwave Engineering Lab ECE	Machine Design 1 MME

##### Tuesday

1:00PM- 2:30PM	Device Programming for IoT CSE
2:30PM -4:00PM	Coding Theory CSE
4:00PM-5:30PM	Digital Image Processing CSE

### 3rd Year Time Table

#### Even Semester

##### Monday

1:00PM- 2 :00PM	Compiler Design CSE	Computer Networks CCE	Computer Networks ECE	I C Engines MME
2:00PM- 3 :00PM	Machine Learning and Pattern Recognition CSE	Economics for Engineers CCE	Design Lab. - 2 ECE	Modeling and Design of Robots MME

##### Tuesday

1:00PM- 2:30PM	Compiler Design CSE	Computer Networks CCE	Computer Networks ECE	I C Engines MME
2:30PM -4:00PM	Machine Learning and Pattern Recognition CSE	Economics for Engineers CCE	Design Lab. - 2 ECE	Modeling and Design of Robots MME
4:00PM-5:30PM				

##### Wednesday

1:00PM- 2 :00PM	Compiler Design CSE	Computer Networks CCE	Computer Networks ECE	I C Engines MME
2:00PM- 3 :00PM	Machine Learning and Pattern Recognition CSE	Economics for Engineers CCE	Design Lab. - 2 ECE	Modeling and Design of Robots MME

**Add Delete Course:**

## Admin

### Add New Course

Select Type of course ▾

Select Semester ▾

Select Branch ▾

Add Course

### Test cases:

Course Id	Course Name	Remarks
CSE 233	Lorem Ipsum	Prompt Displaying: Only AlphaNumeric characters allowed
CSE233	Lorem Ipsum	Course Added in the Menu