Project Report for Milestone 1

Project: Course Advising Portal

Student Name: Quhura Fathima

Student UIN: 01275914

1. Overview (10 points)

Project Description:

The **Course Advising Portal** is a comprehensive platform designed to streamline and secure the academic advising process for students at Old Dominion University. The portal provides a seamless experience for both students and administrators, ensuring that advising tasks are managed efficiently. Key features include user registration, two-factor authentication, password management, and an admin-specific interface for overseeing student submissions. The system is designed with security, user-friendliness, and responsiveness in mind, leveraging modern web technologies to deliver an optimal experience.

Key Features:

1. **User Registration and Authentication**:
    - **Student Registration**: Students can sign up for the portal by providing personal details such as their first name, last name, email, Password, Department, Degree and UIN (University Identification Number).
    - **Email Verification with OTP**: To ensure authenticity, an OTP (One-Time Password) is sent to the student's email address during registration. Only verified users are granted access to the portal.
    - **Two-Factor Authentication (2FA)**: For added security, users are required to enter an OTP sent to their email after logging in with their credentials.
2. **Password Management**:
    - **Forgot Password**: Students who forget their passwords can request an OTP to reset their password, ensuring that the process is secure.
    - **Change Password**: Logged-in users can also update their passwords from within their profile.
3. **User Dashboard and Profile Management**:

   - **Student Dashboard**: Once logged in, students are directed to their dashboard, where they can access their profile, update personal information, and manage their account details.
   - **Profile Update**: Users can edit personal details such as their first name, last name, and department. Email changes are restricted for security reasons.
4. **Admin Interface**:
   - **Admin Dashboard**: Administrators have a dedicated dashboard with a distinct interface. Admins can view and approve submitted advising sheets from students, managing the course advising process efficiently.
   - **Admin Privileges**: Admins are differentiated from regular users and can only be created from the backend by a system administrator.

Technologies Used:

1. **Frontend**:
   - **React.js**: The portal's frontend is built using **React.js**, providing a dynamic, responsive, and seamless user interface.
   - **Material UI**: **Material UI** components ensure that the portal maintains a clean, modern design while adhering to accessibility and usability standards.
2. **Backend**:
   - **Node.js** and **Express.js**: The backend server is developed using **Node.js** and **Express.js**, providing a fast, scalable, and flexible architecture for handling user requests, authentication, and database operations.
   - **Session Management**: The application uses session-based authentication, maintaining user sessions securely and ensuring that only authorized users can access specific routes.
3. **Database**:
   - **MySQL**: Data persistence is handled by **MySQL**, a reliable and scalable relational database. User information, OTPs, and advising records are securely stored and managed.
   - **XAMPP**: The development environment uses **XAMPP** to run MySQL locally, offering an easy-to-manage solution for database operations during development.
4. **Email Services**:

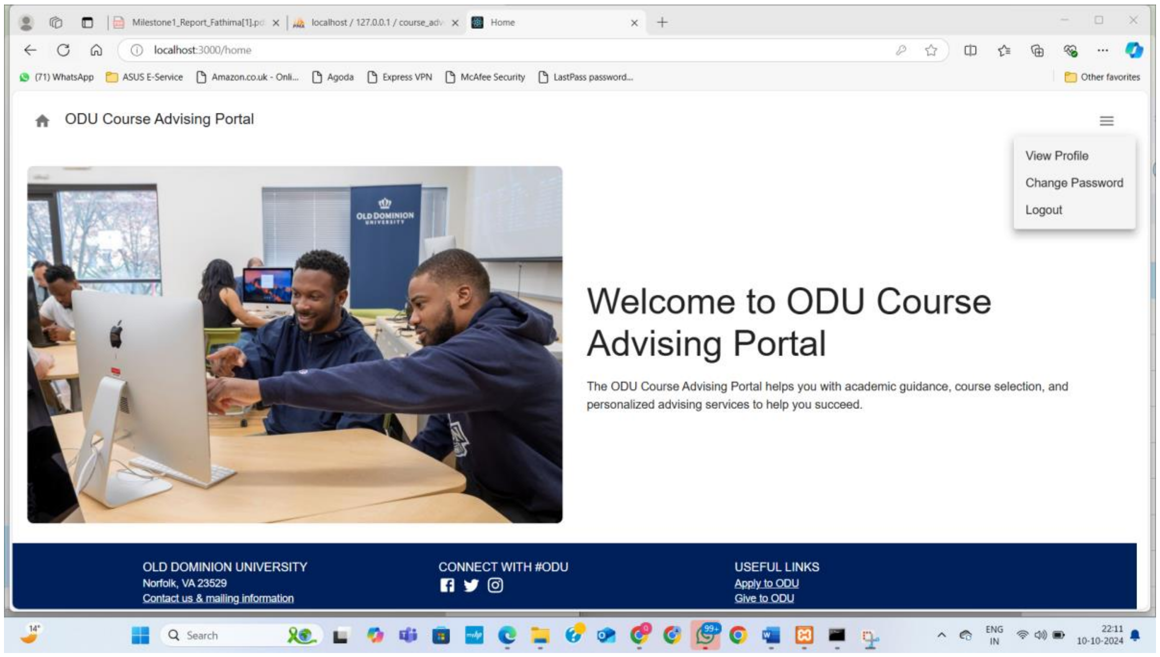
   - **Nodemailer**: OTP verification is implemented via email using **Nodemailer**. Users receive OTPs for both registration and two-factor authentication via their registered email addresses.

Security Features:

- **Password Encryption**: All user passwords are hashed using **bcrypt** before being stored in the database, ensuring that sensitive information is protected.
- **Two-Factor Authentication (2FA)**: By requiring OTP verification during login, the portal significantly reduces the risk of unauthorized access.
- **Session-Based Authentication**: The system uses secure sessions to manage user authentication, preventing unauthorized access to protected routes.

Below are the screenshots of all the working pages

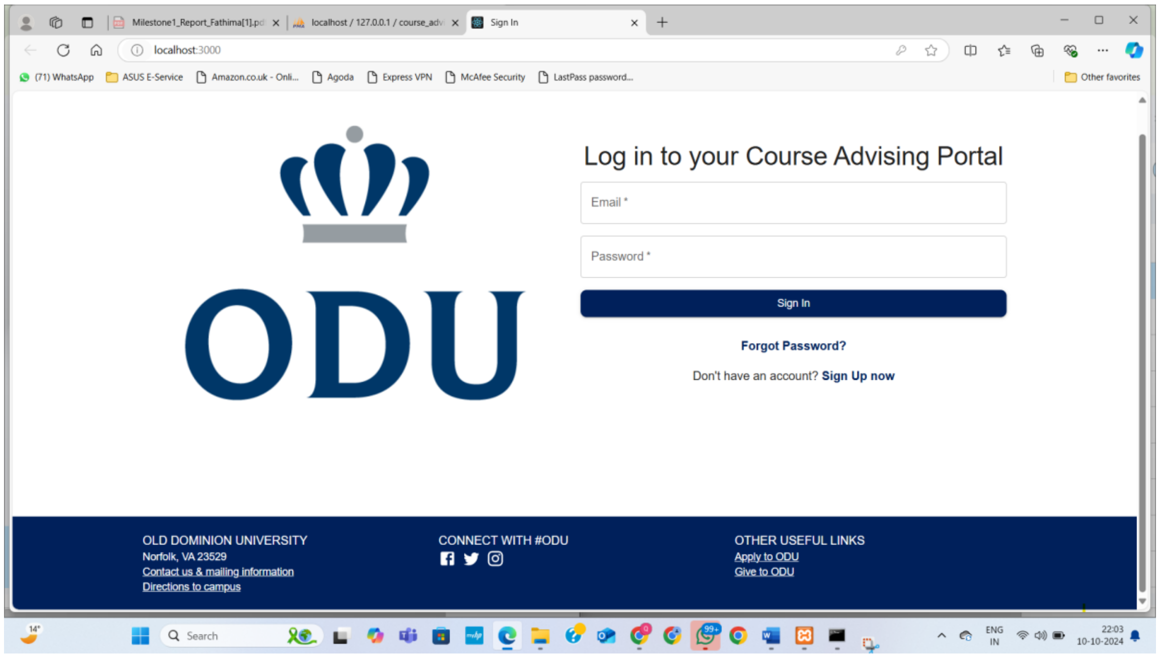
SignUp Page



Welcome to ODU Course Advising Portal

The ODU Course Advising Portal helps you with academic guidance, course selection, and personalized advising services to help you succeed.

SignIn Page



Log in to your Course Advising Portal

Email *

Password *

Sign In

Forgot Password?

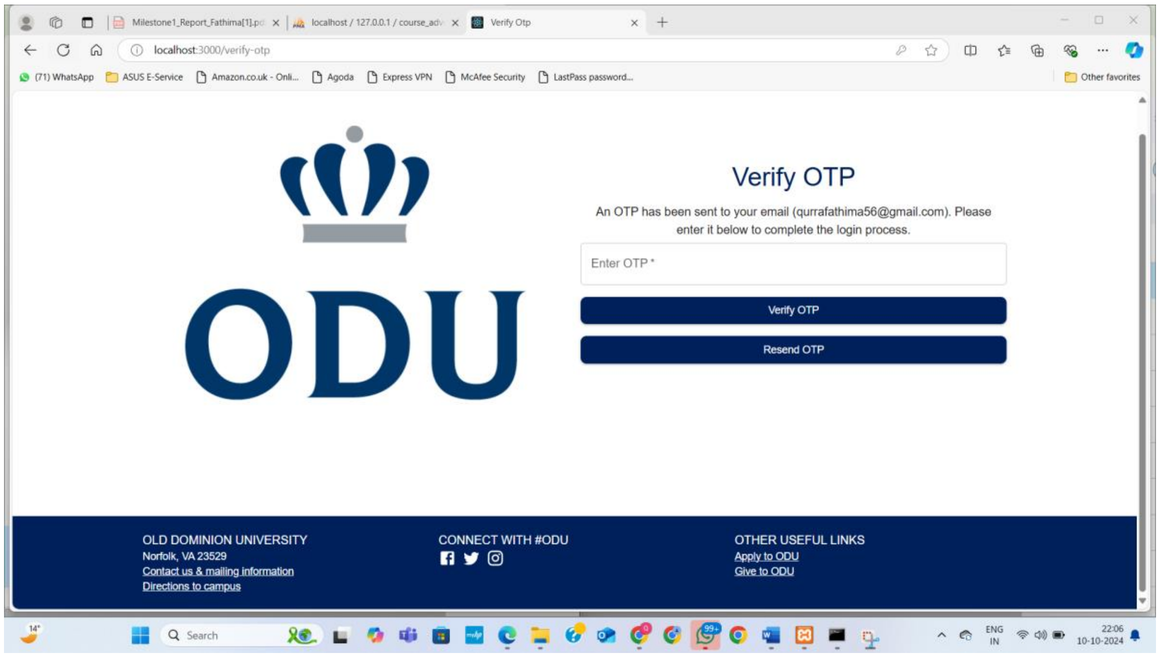
Don't have an account? Sign Up now

OLD DOMINION UNIVERSITY
Norfolk, VA 23529
Contact us & mailing information
Directions to campus

CONNECT WITH #ODU

OTHER USEFUL LINKS
Apply to ODU
Give to ODU

VerifyOtp Page



Verify OTP

An OTP has been sent to your email (qurrafathima56@gmail.com). Please enter it below to complete the login process.

Enter OTP *

Verify OTP

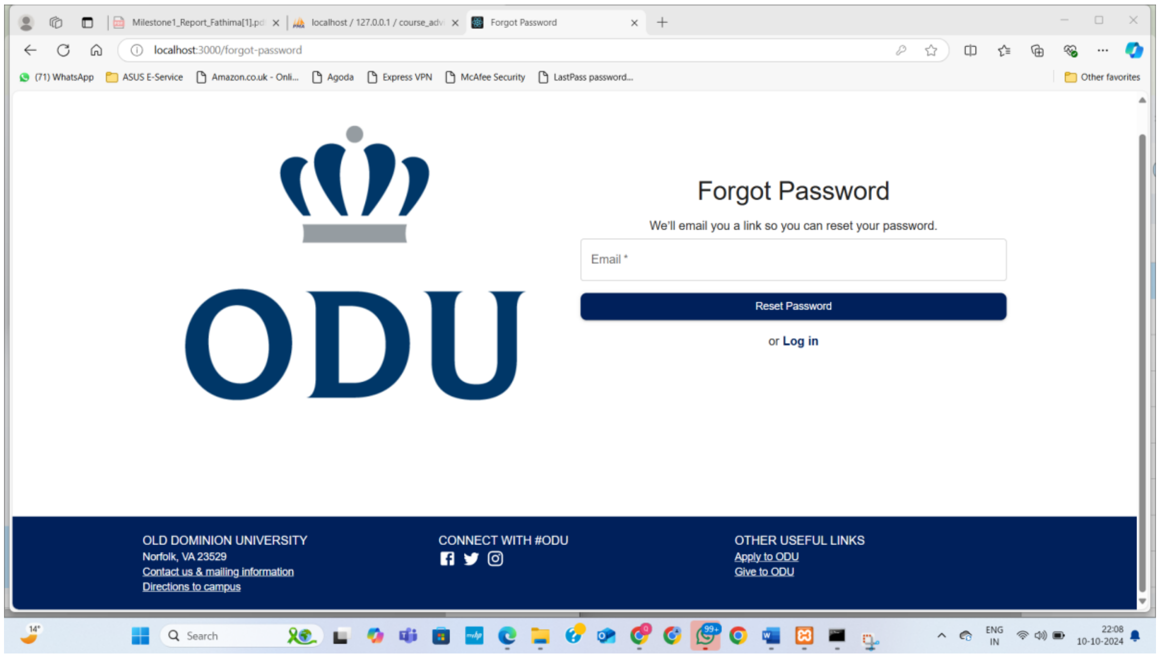
Resend OTP

OLD DOMINION UNIVERSITY
Norfolk, VA 23529
Contact us & mailing information
Directions to campus

CONNECT WITH #ODU

OTHER USEFUL LINKS
Apply to ODU
Give to ODU

Forgot Password Page



Forgot Password

We'll email you a link so you can reset your password.

Email *

Reset Password

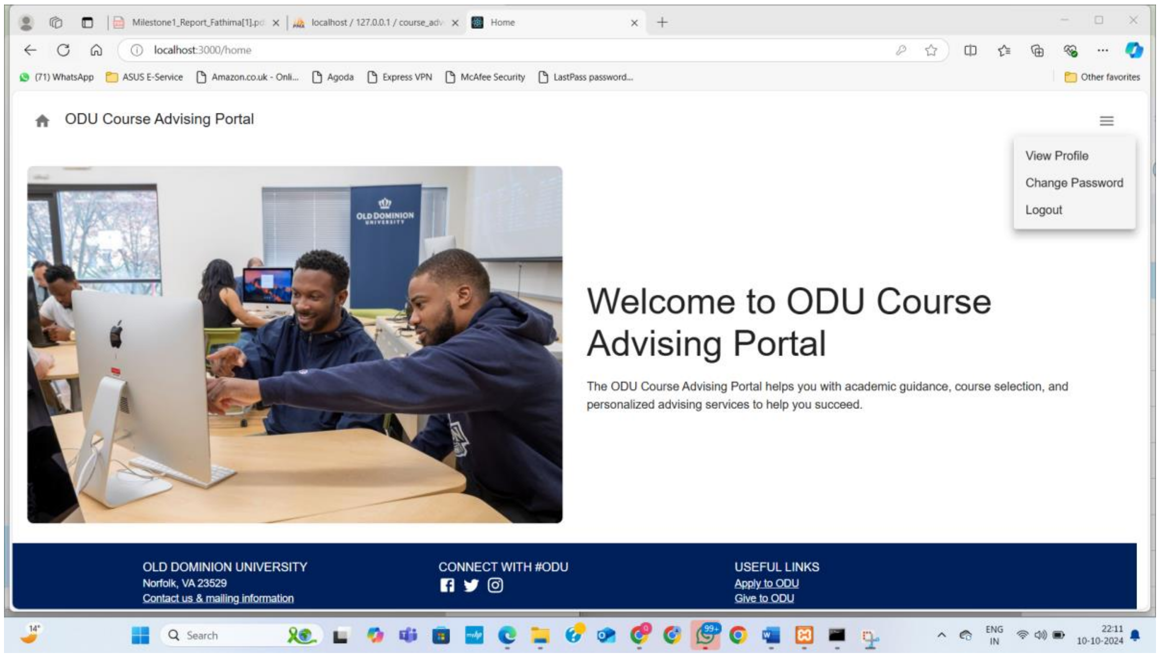
or Log in

OLD DOMINION UNIVERSITY
Norfolk, VA 23529
Contact us & mailing information
Directions to campus

CONNECT WITH #ODU

OTHER USEFUL LINKS
Apply to ODU
Give to ODU

User Home Page



ODU Course Advising Portal

View Profile
Change Password
Logout

Welcome to ODU Course Advising Portal

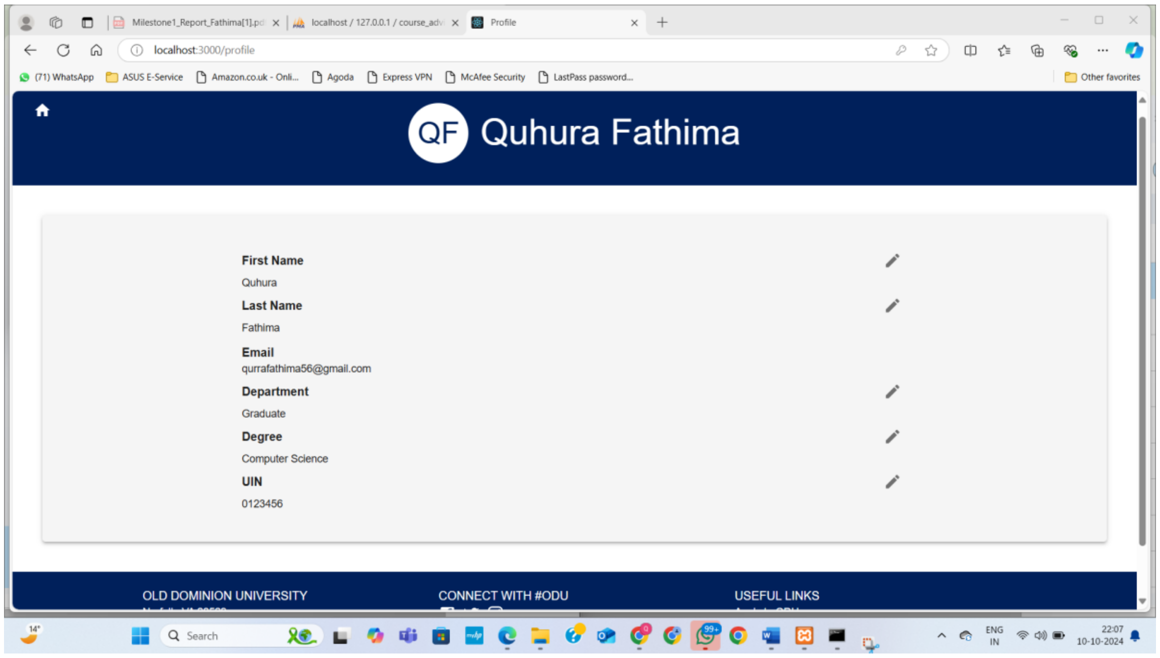
The ODU Course Advising Portal helps you with academic guidance, course selection, and personalized advising services to help you succeed.

OLD DOMINION UNIVERSITY
Norfolk, VA 23529
Contact us & mailing information

CONNECT WITH #ODU

USEFUL LINKS
Apply to ODU
Give to ODU

Profile Page



localhost:3000/profile

QF Quhura Fathima

First Name
Quhura

Last Name
Fathima

Email
qurrafathima56@gmail.com

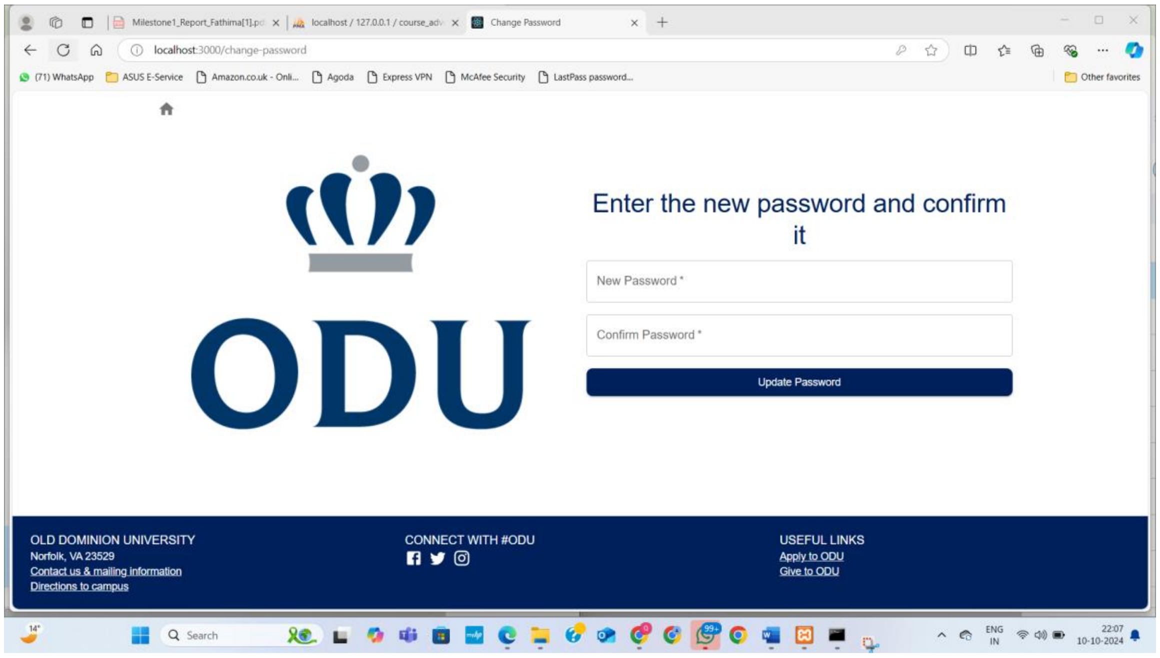
Department
Graduate

Degree
Computer Science

UIN
0123456

OLD DOMINION UNIVERSITY
CONNECT WITH #ODU
USEFUL LINKS

ChangePassword Page



localhost:3000/change-password

Enter the new password and confirm it

New Password *

Confirm Password *

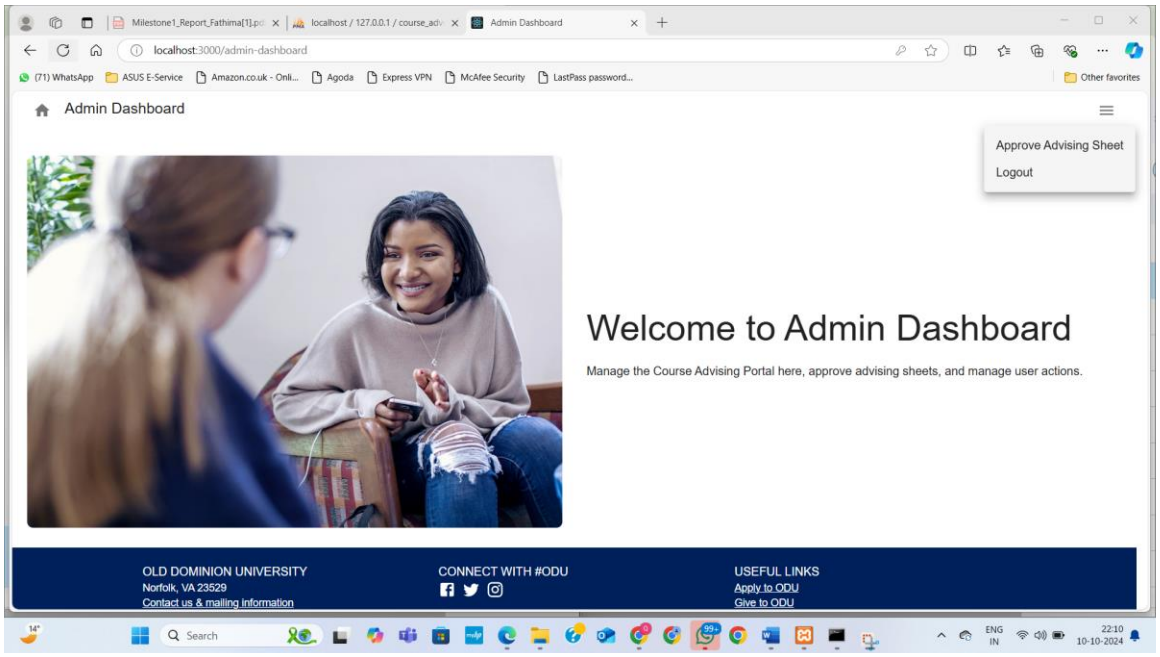
Update Password

OLD DOMINION UNIVERSITY
Norfolk, VA 23529
Contact us & mailing information
Directions to campus

CONNECT WITH #ODU

USEFUL LINKS
Apply to ODU
Give to ODU

AdminDashboard page:



Welcome to Admin Dashboard

Manage the Course Advising Portal here, approve advising sheets, and manage user actions.

2. Milestone Accomplishments

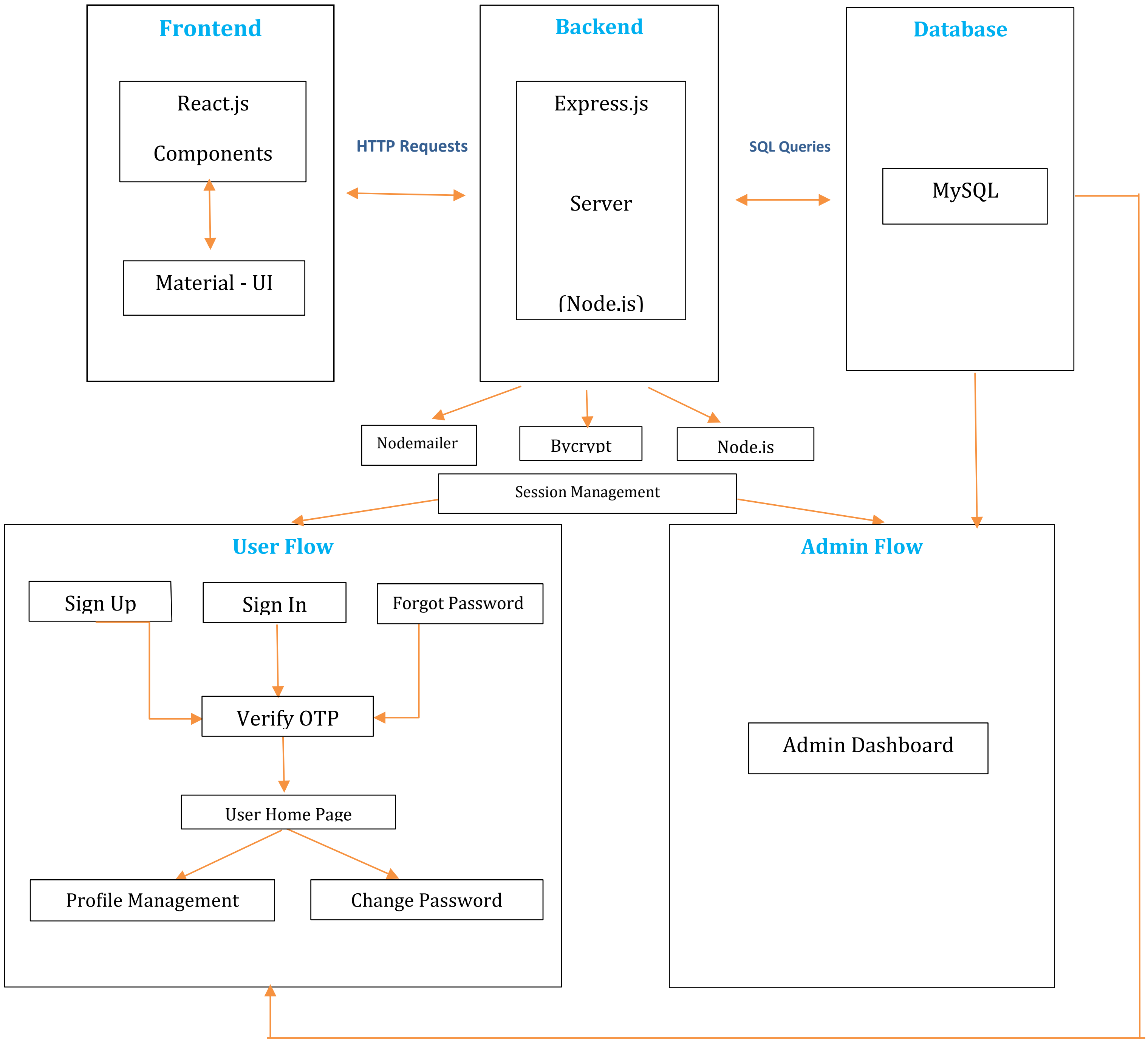
All specifications for this milestone have been implemented. The table below shows the status of each feature.

Table 1: Status of milestone specifications.

Fulfilled	Feature#	Specification
Yes	1	Users should be able to register new accounts using email addresses.
Yes	2	Users are identified by email address
Yes	3	Password must be encrypted before storing in the database**.**
Yes	4	Users cannot register duplicate accounts using the same email address.
Yes	5	The user should receive a verification email upon successful registration.
Yes	6	Users cannot log in to the system until their email has been verified.
Yes	7	Users should be able to log into your website using the accounts they registered.
Yes	8	Users should be able to reset their passwords if they forget it.
Yes	9	Users should be able to change their passwords after they login.
Yes	10	A 2-factor-authentication should be used when a user attempt to login. This can be done by email, phone text, or a DUO push. You can just

		implement one of them.
Yes	11	The website should have a homepage for each user, where they can view their profiles, change passwords, and update information. Email cannot not be changed.
Yes	12	An admin user should be created from the backend. (Only 1)
Yes	13	An admin user has a different view from a regular user. (Later admin will approve the submitted advising sheet by student)

3. Architecture



Flow of the Architecture:

1. **User Registration**:

   - **Frontend**: Users visit the registration page (via **SignUp.js**) and input details like their name, email, password, department, etc. The form validates the inputs before submission.

   - **Backend**: The registration data is sent to the server (**/signup** route in **server.js**) where the email is validated, the password is hashed using **Bcrypt**, and an OTP (One-Time Password) is generated.

   - **Email OTP**: Using **Nodemailer**, the OTP is sent to the user's email. The OTP is stored temporarily in the backend until verified.

   - **Frontend**: The user must verify the OTP (handled by **VerifyOtp.js**) to complete registration. Once verified, the user's data is saved to the **MySQL** database.

2. **User Login**:

   - **Frontend**: Users log in by entering their email and password in the login form (**Login.js**).

   - **Backend**: The server (**/login** route in **server.js**) compares the entered password (hashed) with the stored password using **Bcrypt**. If the password is correct, an OTP is generated and sent to the user's email for 2-factor authentication (2FA).

   - **OTP Verification**: Users must verify the OTP via the OTP verification page (**VerifyOtp.js**) to gain access to their account.

   - **Session Management**: Once verified, a session is created for the user using **express-session** to allow authenticated access to protected routes.

3. **Profile Management**:

   - **Frontend**: Once logged in, users can view and update their profile (except for the email) on the profile page (**Profile.js**). They can also change their password using **ChangePassword.js**.

   - **Backend**: Profile data and password updates are sent to the backend, which updates the corresponding entries in the **MySQL** database after validation. Passwords are hashed before being saved.

4. **Forgot Password**:
    - **Frontend**: Users who forget their password can request a password reset by entering their email on the **ForgotPassword.js** page.
    - **Backend**: The server generates and sends an OTP to the user's email. After verifying the OTP, the user can enter a new password, which is then hashed and saved in the database.
5. **Two-Factor Authentication (2FA)**:
    - **OTP Flow**: Both during login and password reset, an OTP is sent to the user's email as a second layer of authentication. This ensures that even if the password is compromised, the OTP adds an additional security layer.
6. **Admin User Flow**:
    - **Admin Creation**: A special admin user is created automatically when the system is initialized (via **createAdmin.js**).
    - **Admin Login**: The admin logs in similarly to a regular user, with an email, password, and OTP verification. However, their session grants access to a special **AdminDashboard.js**.
    - **Admin Dashboard**: Admins can perform actions like approving student advising sheets through the protected **admin-dashboard** route, which only allows access if the session belongs to an admin.
7. **Session and Authorization**:
    - **Session Management**: Once users (both regular and admin) log in and verify their OTP, their session is maintained using **express-session**, allowing them to access their profile, home, and admin dashboard without logging in again.
    - **Admin Protection**: Admin routes are protected by a middleware function (**verifyAdmin**) in **server.js**, ensuring that only users with admin privileges can access the admin dashboard.

5. Database Design

The database contains two main tables: **'users'** and **'temp_users'**.

Table 1: Users Table

Field	Type	Key	Example
id	INT	Primary	1
firstName	VARCHAR(100)		Quhura
lastName	VARCHAR(100)		Fathima
email	VARCHAR(255)		quhurafathima56@gmail.com
password	VARCHAR(255)		$2b$10$N9bNIp3cjTeQOBeWpzl
department	VARCHAR(100)		Computer Science
degree	VARCHAR(100)		Masters/Graduate
uin	VARCHAR(50)		01275914
isVerified	TINYINT(1)		0
verificationToken	VARCHAR(255)		token_value
otp	VARCHAR(6)		123456
otp_expiration	BIGINT(20)		1609459200
is_admin	TINYINT(1)		0 / 1

Table 2: Temp_Users Table

Field	Type	Key	Example
id	INT	Primary	1
firstName	VARCHAR(100)		Jane
lastName	VARCHAR(100)		Doe
email	VARCHAR(255)		janedoe@gmail.com
password	VARCHAR(255)		$2b$10$N9bNIp3cjTeQOBeWpzl
department	VARCHAR(100)		Engineering
degree	VARCHAR(100)		Bachelors/Undergraduate
uin	VARCHAR(50)		**01275910**
otp	VARCHAR(6)		654321
createdAt	TIMESTAMP		2023-10-01 10:00:00

5. Implementation

1. Users should be able to register new accounts using email addresses.

- **File**: SignUp.js, server.js
- **Frontend Function (SignUp.js)**:
    - **Function**: handleSignUp()
        - This function collects user details like First Name, Last Name, Email, Password, and other fields.
        - It sends a POST request to the server (backend) with the user's information to the /signup route.
    - **Backend (server.js)**:
        - **Route**: app.post('/signup')
        - **Details**:
            - The backend validates that all required fields are provided.
            - It checks if an account with the same email already exists using a MySQL SELECT query.
            - The password is hashed using bcrypt.hash() for security, and an OTP is generated and sent to the user's email via nodemailer.
            - The user's details are temporarily stored in the temp_users table.
        - **Function**: checkEmailDomain()
            - This function verifies if the email domain is valid by checking DNS MX records.
        - **Function**: generateOTP()
            - This function generates a 6-digit OTP using the crypto module.
        - **Function**: transporter.sendMail()

        - This function sends the OTP email to the user.

---

2. Users are identified by email address.

- **File**: server.js
- **Backend**:
    - In all user-related queries (signup, login, profile), the system uses email as the primary identifier.
    - For example, during login, the query **SELECT * FROM users WHERE email = ?** is used to retrieve user details based on the provided email.
    - The email is normalized (converted to lowercase and trimmed) to prevent case sensitivity issues.
    - **Function**: The backend uses normalizedEmail to ensure consistency when querying the database.

---

3. Password must be encrypted before storing in the database.

- **File**: server.js
- **Backend Function**:
    - **Function**: bcrypt.hash(password, saltRounds, callback)
        - In the app.post('/signup') route, before saving the password in the database, it is hashed using bcrypt.
        - bcrypt.hash() takes the plain password and applies a hashing algorithm (with a salt round for additional security), returning a hashed password.
        - This hashed password is stored in the database, preventing anyone (including admins) from viewing the plain text password.
    - **File**: createAdmin.js also uses this function for hashing the admin password.

---

4. Users cannot register duplicate accounts using the same email address.

- **File**: server.js
- **Backend Function**:
  - **Function**: db.query('SELECT * FROM users WHERE email = ?', [email], callback)
    - During registration (inside app.post('/signup')), this function checks if the email already exists in the users table.
    - If an existing user is found, the system responds with an error (User already exists with this email), preventing duplicate registrations.

---

5. The user should receive a verification email upon successful registration.

- **File**: server.js
- **Backend Function**:
  - **Function**: generateOTP() generates a 6-digit OTP.
  - **Function**: transporter.sendMail() sends the OTP email to the user's registered email address.
  - **Details**:
    - After registration, the OTP is sent using the nodemailer package configured with Gmail credentials (process.env.EMAIL_USER and process.env.EMAIL_PASS).
    - The user can use this OTP to verify their email during the signup process.

---

6. Users cannot log in to the system until their email has been verified.

- **File**: server.js
- **Backend Functions**:
  - **Function**: app.post('/verify-otp')

        - The OTP entered by the user is validated against the stored OTP (in memory for sign-up flow).
        - If the OTP is valid and not expired, the user is moved from temp_users to users, marking them as verified.
    - **Login Check**:
        - During login (app.post('/login')), if the user is not verified, they cannot log in.
        - The field isVerified = true is checked to ensure the user has verified their email before allowing access.

---

7. Users should be able to log into your website using the accounts they registered.

- **File**: SignIn.js, server.js
- **Frontend Function (SignIn.js)**:
    - **Function**: handleLogin()
        - Sends the user's email and password to the backend for authentication.
- **Backend Function (server.js)**:
    - **Route**: app.post('/login')
    - **Function**: bcrypt.compare()
        - This function compares the provided password with the hashed password stored in the database.
        - If the password matches, the user is authenticated.
    - If successful, the user's session is created and the user is logged in.

---

8. Users should be able to reset their passwords if they forget it.

- **File**: ForgotPassword.js, server.js

- **Frontend Function (ForgotPassword.js)**:
    - **Function**: handleSendOtp()
        - Sends the email to the backend requesting an OTP for password reset.
- **Backend Functions (server.js)**:
    - **Route**: app.post('/resend-otp')
    - **Function**: Sends an OTP for resetting the password to the user's email.
    - **Route**: app.post('/change-password')
    - **Function**: Allows the user to reset their password by providing the new password and verifying the OTP.

---

9. Users should be able to change their passwords after they login.

- **File**: ChangePassword.js, server.js
- **Frontend Function (ChangePassword.js)**:
    - **Function**: handleChangePassword()
        - After logging in, the user can change their password by providing the new password and confirmation.
- **Backend Function (server.js)**:
    - **Route**: app.post('/change-password')
    - **Function**: The backend hashes the new password and updates it in the database.

---

10. A 2-factor-authentication (2FA) should be used when a user attempts to login.

- **File**: server.js
- **Backend Functions**:

- **Route**: app.post('/login')
    - After successfully validating the password, the system generates a new OTP using generateOTP().
    - The OTP is sent to the user's email, and they must verify it via app.post('/verify-otp') before logging in.
- **Route**: app.post('/verify-otp')
    - This function checks if the OTP is correct and valid for 2FA.

---

11. The website should have a homepage for each user, where they can view their profiles, change passwords, and update information. Email cannot be changed.

- **File**: Home.js, Profile.js, server.js
- **Frontend Functions (Profile.js)**:
    - **Function**: handleSaveClick() allows users to update their personal information (First Name, Last Name, etc.) but does not allow changing the email.
    - **Function**: handleEditClick() enables the user to edit their profile fields.
- **Backend Functions**:
    - **Route**: app.get('/profile') retrieves the user's profile details from the database.
    - **Route**: app.post('/profile') updates the user's information (except email) in the database.

---

12. An admin user should be created from the backend. (Only 1)

- **File**: createAdmin.js, server.js
- **Backend Functions**:
    - **Function**: createAdminUser()

        - This function checks if an admin already exists in the users table.
        - If not, it creates an admin user with a predefined email and password (hashed using bcrypt).
    - **Code Location**: The createAdminUser() function is invoked in server.js when the server starts, ensuring that the admin user is created if it doesn't already exist.

---

13. An admin user has a different view from a regular user.

- **File**: AdminDashboard.js, server.js
- **Frontend (AdminDashboard.js)**:
    - **Function**: Renders a different UI for admins, allowing them to perform tasks like approving advising sheets.
- **Backend Function (server.js)**:
    - **Route**: app.get('/admin-dashboard')
    - **Middleware**: verifyAdmin checks if the user is an admin by verifying the isAdmin flag in the session before allowing access to the admin dashboard.
    - **Details**: If the user is not an admin, the middleware returns a 403 Access Denied response.