# Task Management System

Samar Ali Elsayed

# Backend

## 1. Set up Project

**1.1 Install Dependencies**: npm install

**1.2 Start the Server** : npm start

http://localhost:8000

**1.3 .env file:** Create a .env file in the root directory of your project , Add the following environment variables with your SQL database credentials:

```
PORT=8000
SQL_USER=sa
SQL_PASSWORD=admin
SQL_SERVER=SAMAR\SQLEXPRESS
SQL_DATABASE=task
JWT_SECRET_KEY = JWT_SECRET

apiKey=AIzaSyBrTZoVk9twsQATVOwFcJZwOYpzs5QxdtE
authDomian=fir-upload-47e23.firebaseapp.com
projectId=fir-upload-47e23
storageBucket=fir-upload-47e23.appspot.com
messagingSenderId=17198946976
appId=1:317198946976:web:511f1ee0e3bedfca5cefc5
FIREBASE_USER=engsamar105@gmail.com
FIREBASE_AUTH=samar1234
STRIPE_SECRET_KEY =
sk_test_51OibQFKccMjnDABQJumUnZn9sSdvRXIVxMmTyRPxAco1ri7mUj27DQ
8pPnv0k5iv0xCKeyVDrdhlPJ3ceTDk1n50003TBE8t2W
```

- **Note: Replace your _sql_user, your_sql_password, your_sql_server, and your_sql_database with your actual SQL server configuration details.**

## 2. User Endpoints

### 2.1 Postman Collection: [Task Management API]:

https://crimson-comet-907136.postman.co/workspace/Team-Workspace~846ef1f1-6a55-4853-8d31-1aad198ad721/collection/22502337-6657a6f0-81db-4e2b-9d59-062cfc55c0b2?action=share&creator=22502337

### 2.2 Register User

**2.2.1 Endpoint**: http://localhost:8000/api/v1/users/register (POST)

**2.2.2 Request Body**:

{ "password": " varchar ","firstname": " varchar ","lastname": "string", "email": " varchar"}

**2.2.3 Response on Success (Status Code 201):**

{"status": "success", "message": "User registered successfully"}

**2.2.4 Response on Duplicate Email (Status Code 400):**

{ "message": "Email already exists"}

**2.2.5 Response on Validation Errors (Status Code 400):**

{ "message": " Validation error message"}

**2.2.6 Response on Server Errorероore (Status Code 500):**

{ "message": " Internal server error message"}

### 2.3 Login User

**2.3.1 Endpoint**: http://localhost:8000/api/v1/users/login (POST)

**2.3.2 Request Body**:

{"email": " varchar", "password": " varchar "}

**2.3.3 Response on Success (Status Code 200):**

{"status": "success", "token": "token"}

**2.3.4 Response on Missing Email or Password (Status Code 400):**

{ "message": " Required email and password"}

**2.3.5 Response on User Email Not Found (Status Code 404):**

{ "message": " This email is not registered. Please sign up"}

**2.3.6 Response on Validation Errors (Status Code 400):**

{ "message": " Validation error message"}

**2.3.7 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

**Note**: **token in header** **key:** authorization**, value:** (Bearer {token})

### 2.4 Get User Profile

**2.4.1 Endpoint**: http://localhost:8000/api/v1/users/userProfile (Get)

**2.4.2 Middleware**: Authentication required (Token in header)

**2.4.3 Response on Success (Status Code 200):**

{"status": "success", ""data": {userData}}

**2.4.4 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**2.4.5 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

### 2.5 Edit User Profile

**2.5.1 Endpoint**: http://localhost:8000/api/v1/users/userProfile **(Put)**

**2.5.2 Middleware**: Authentication required **(Token in header)**

**2.5.3 Request Body (Optional Fields):** { "password": " varchar ","firstname": " varchar ","lastname": " varchar ", "email": " varchar ", "image": " varchar ", "phone": " varchar "}

Note: Use Multer for file upload and Firebase Storage to store the image. Save the image link in the database.

**2.5.4 Response on Success (Status Code 200):**

{"status": "success", "message": "User updated successfully"}

**2.5.5 Response on Authentication Error:**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**2.5.6 Response on Validation Errors :**

{ "message": " Validation error message"}

**2.5.7 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

## 3. Task Endpoints

### 3.1 Add Task to Logged-in User

**3.1.1 Endpoint**: http://localhost:3000/api/v1/tasks (POST)

**3.1.2 Middleware**: Authentication required **(Token in header)**

**3.1.3 Request Body (Required Fields)**:

{ "title": " varchar "," status ": " varchar (values: 'Pending', 'In Progress', 'Completed')"}

**(Optional Fields )** {"description": " varchar "}

**3.1.4 Response on Success (Status Code 201):**

{"status": "success", "message": "Task created successfully"}

**3.1.5 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.1.6 Response on Validation Errors (Status Code 400):**

{ "message": " Validation error message"}

**3.1.7 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

### 3.2 Edit Task to Logged-in User

**3.2.1 Endpoint**: http://localhost:3000/api/v1/tasks/:idTask (PUT)

**3.2.2 Middleware**: Authentication required **(Token in header)**

**3.2.3 Request Body (Optional Fields)**:

{ "title": " varchar ", description": " varchar "," status ": " varchar (values: 'Pending', 'In Progress', 'Completed')"}

**3.2.4 Response on Success (Status Code 200):**

{"status": "success", "message": " Task updated successfully ",data:{data To New Task Updated}}

**3.2.5 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.2.6 Response on Validation Errors (Status Code 400):**

{ "message": " Validation error message"}

**3.2.7 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

### 3.3 Get Tasks for Logged-in User

**3.3.1 Endpoint**: http://localhost:3000/api/v1/tasks (GET)

**3.3.2 Middleware**: Authentication required **(Token in header)**

**3.3.3 Response on Success (Status Code 200):**

{"status": "success", ",data:[{ Tasks To This User }]}

**3.3.4 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.3.5 Response on Not Found Tasks (Status Code 404):**

{ "message": " Tasks Not Found"}

**3.3.6 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

### 3.4 Get Tasks By Status for Logged-in User

**3.4.1 Endpoint**: http://localhost:3000/api/v1/tasks?status=status (GET)

**3.4.2 Middleware**: Authentication required **(Token in header)**

**3.4.3 Response on Success (Status Code 200):**

{"status": "success", ",data:[{ Tasks To This User }]}

**3.4.4 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.4.5 Response on Not Found Tasks (Status Code 404):**

{ "message": " Tasks Not Found"}

**3.4.6 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

### 3.5 Get Task for Logged-in User by Task ID

**3.5.1 Endpoint**: http://localhost:3000/api/v1/tasks/:idTask (GET)

**3.5.2 Middleware**: Authentication required **(Token in header)**

**3.5.3 Response on Success (Status Code 200):**

{"status": "success", ",data:{ Task To This User }}

**3.5.4 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.5.5 Response on Not Found Task (Status Code 404):**

{ "message": " Task  Not Found"}

**3.5.6 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

**3.6 Delete Task for Logged-in User by Task ID**

**3.6.1 Endpoint**: http://localhost:3000/api/v1/tasks/:idTask (Delete)

**3.6.2 Middleware**: Authentication required **(Token in header)**

**3.6.3 Response on Success (Status Code 200):**

{"status": "success", message: "Task deleted successfully"}

**3.6.4 Response on Authentication Error :**

{ "message": " Unauthorized. Authentication token is missing or invalid"}

**3.6.5 Response on Not Found Task (Status Code 404):**

{ "message": " Task  Not Found"}

**3.6.6 Response on Server Error (Status Code 500):**

{ "message": " Internal server error message"}

# FrontEnd

## 1. Set up Project

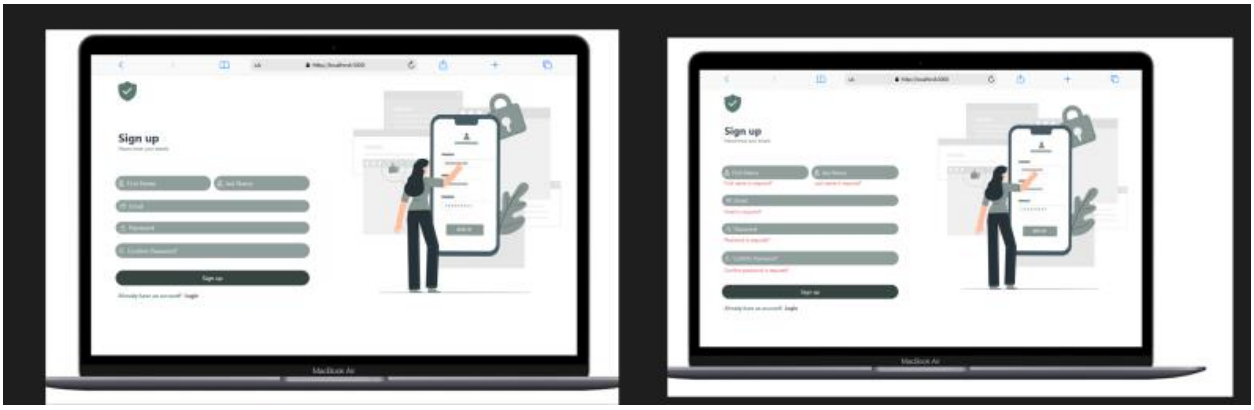
**1.1 Install Dependencies**: npm install

**1.2 Start the Server** : npm run start

http://localhost:3000

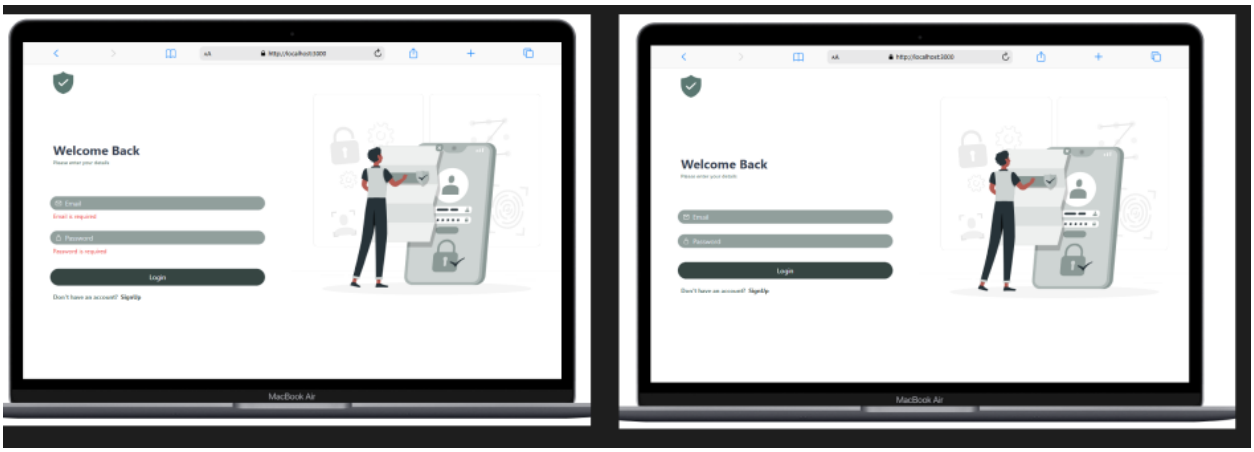
## 2. Register Page

**Endpoint**: http://localhost:3000/register



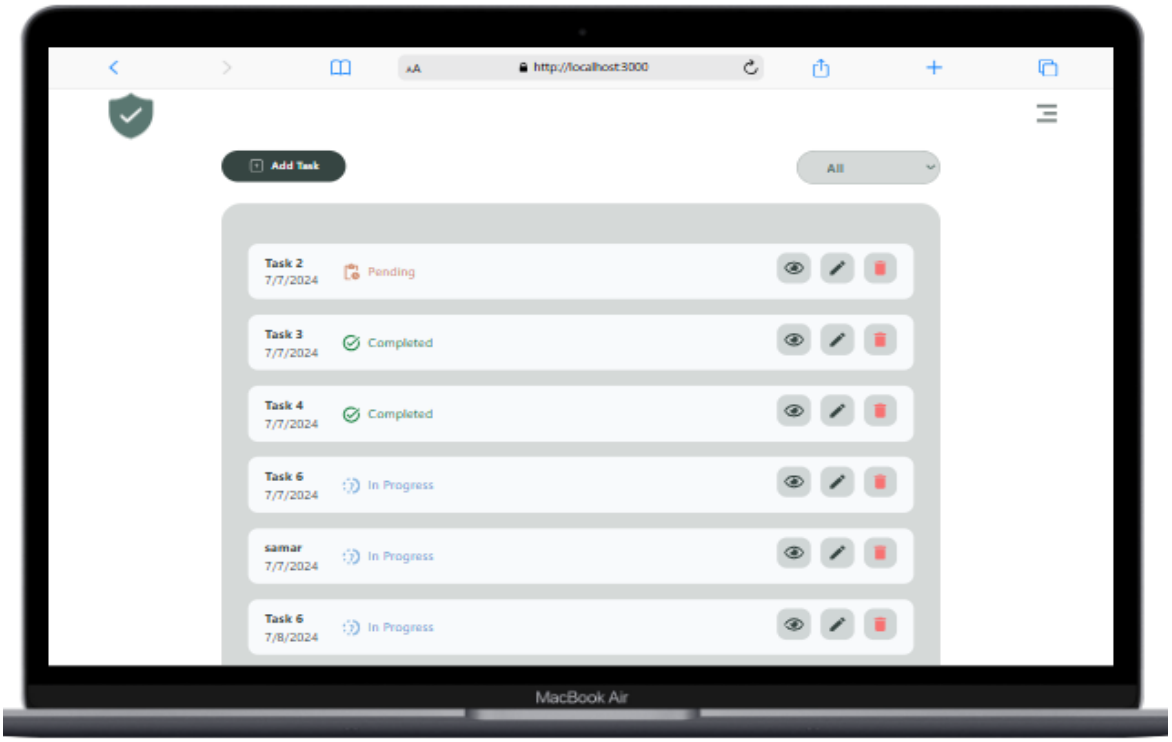

## 3. Login Page

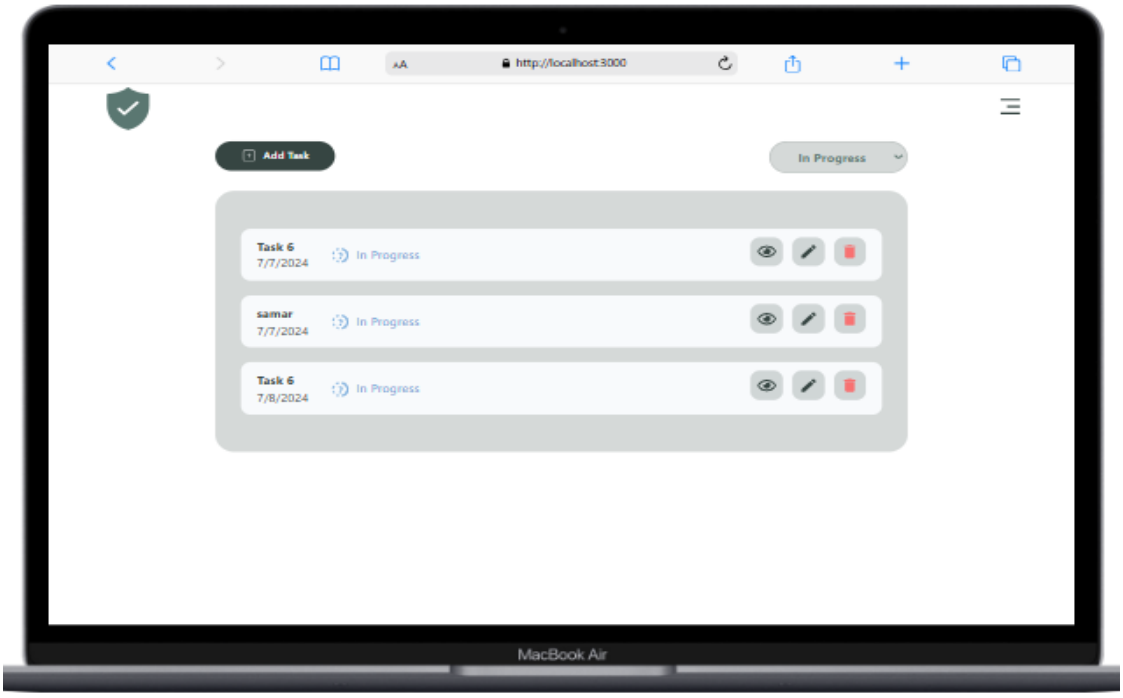
**Endpoint**: http://localhost:3000/login

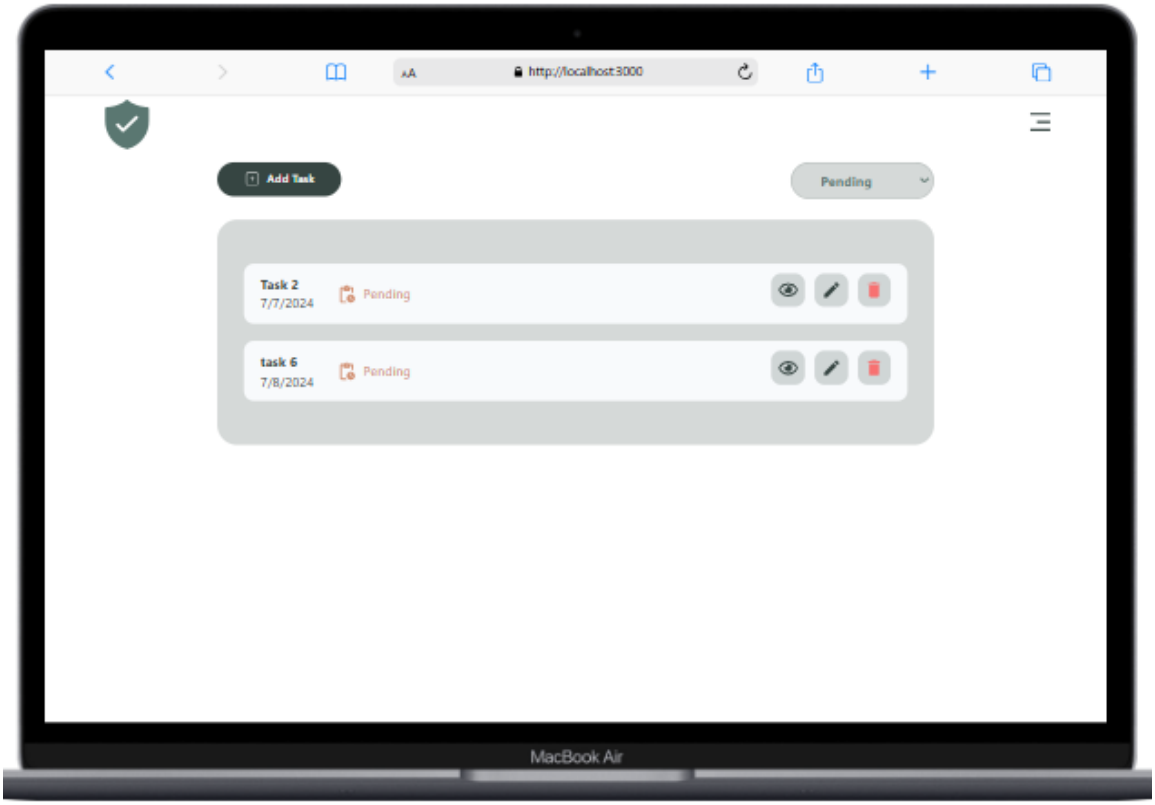


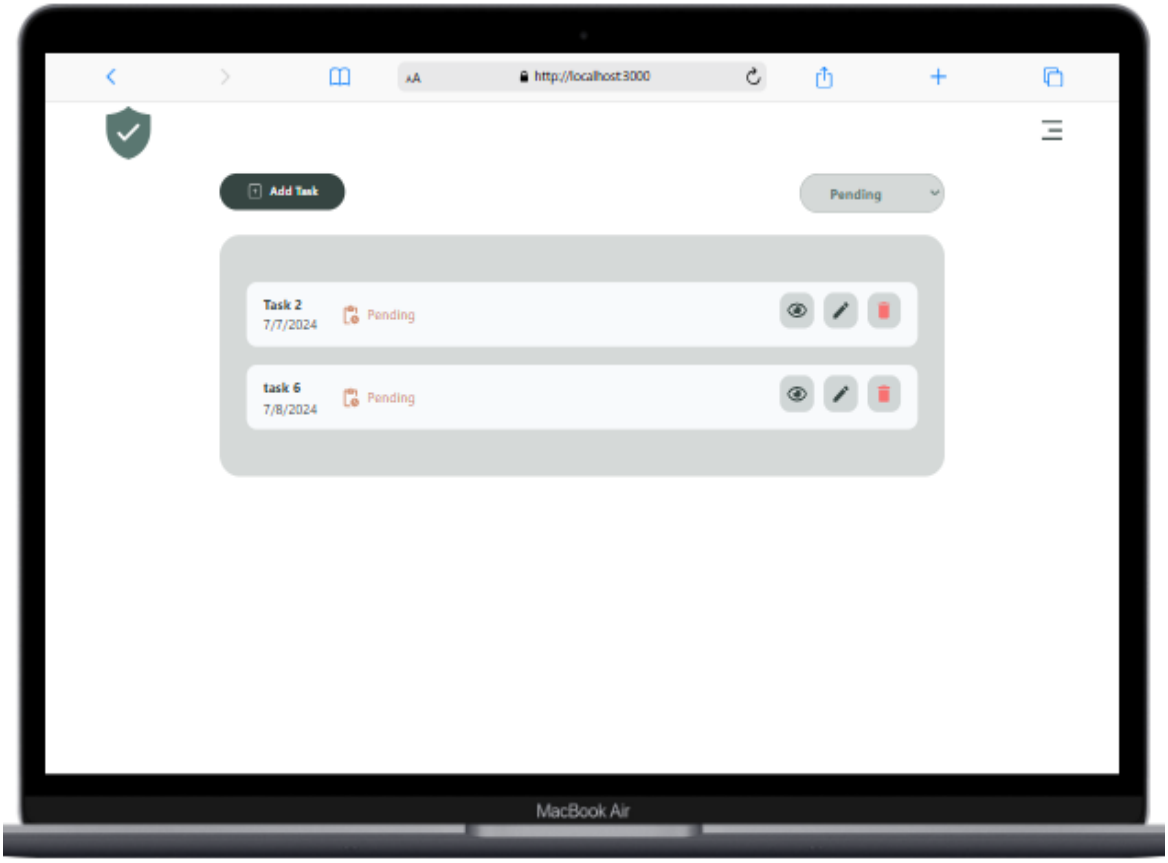
## 4. Home Page

**Endpoint**: http://localhost:3000



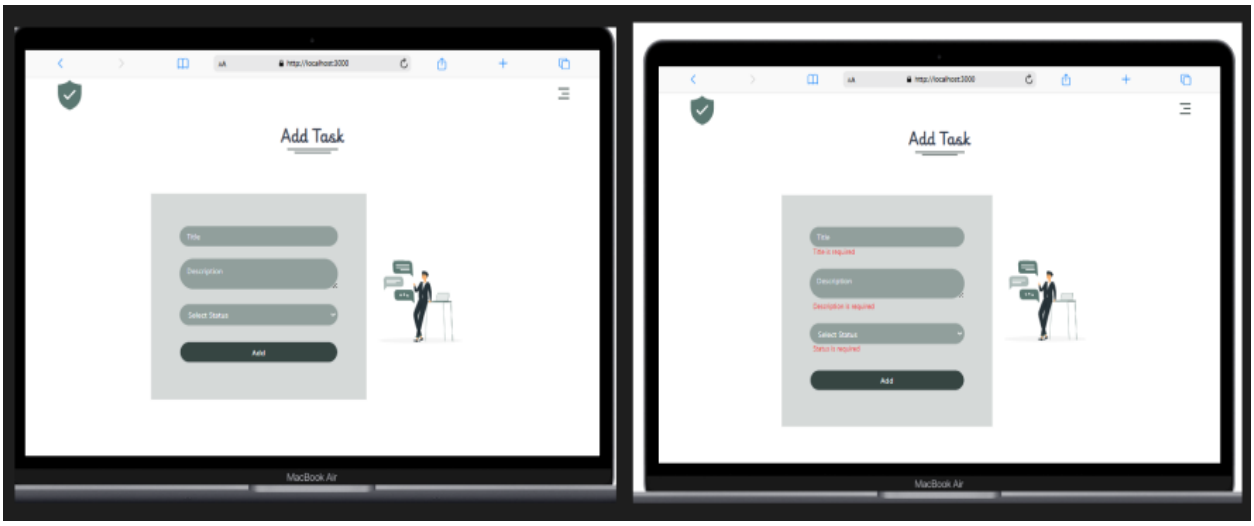




http://localhost:3000
Add Task
Pending
Task 2
7/7/2024
Pending
task 6
7/8/2024
Pending
MacBook Air


http://localhost:3000
Add Task
Pending
Task 2
7/7/2024
Pending
task 6
7/8/2024
Pending
MacBook Air

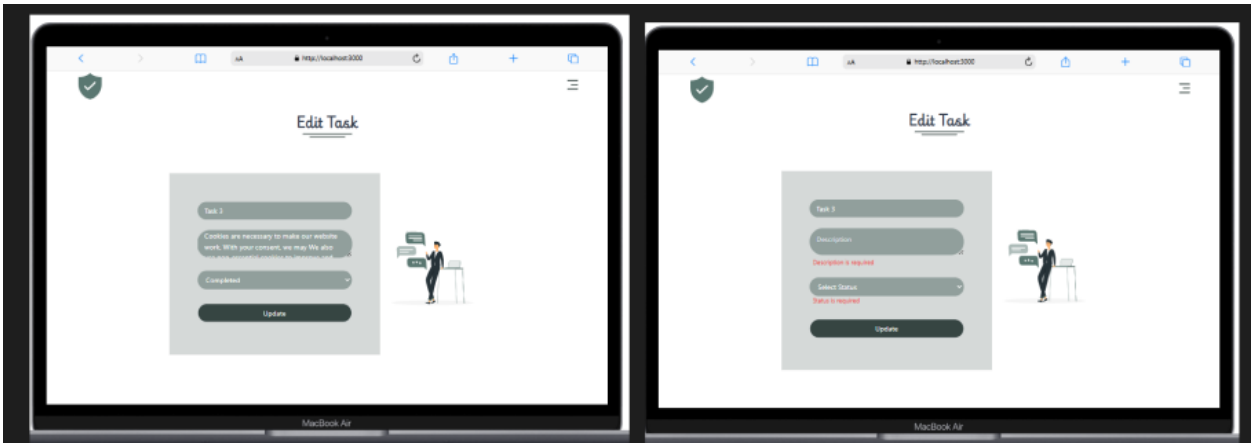
## 5. Add Task Page

**Endpoint**: http://localhost:3000/tasks/add

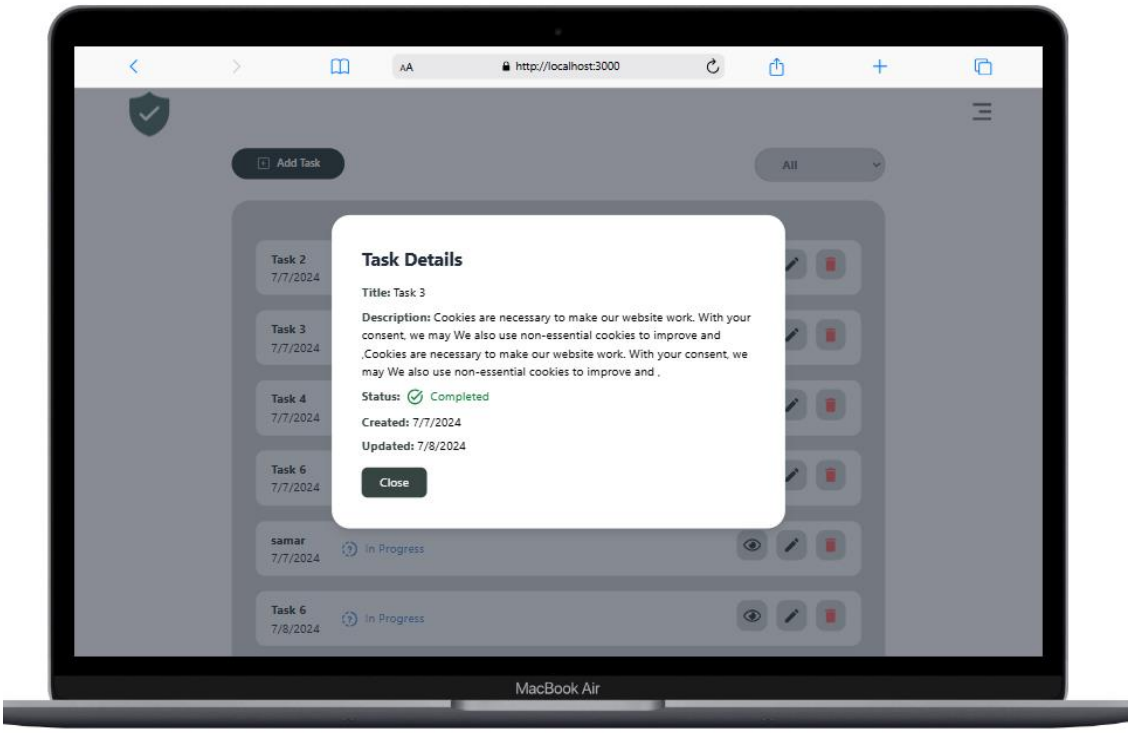



## 6. Edit Task Page

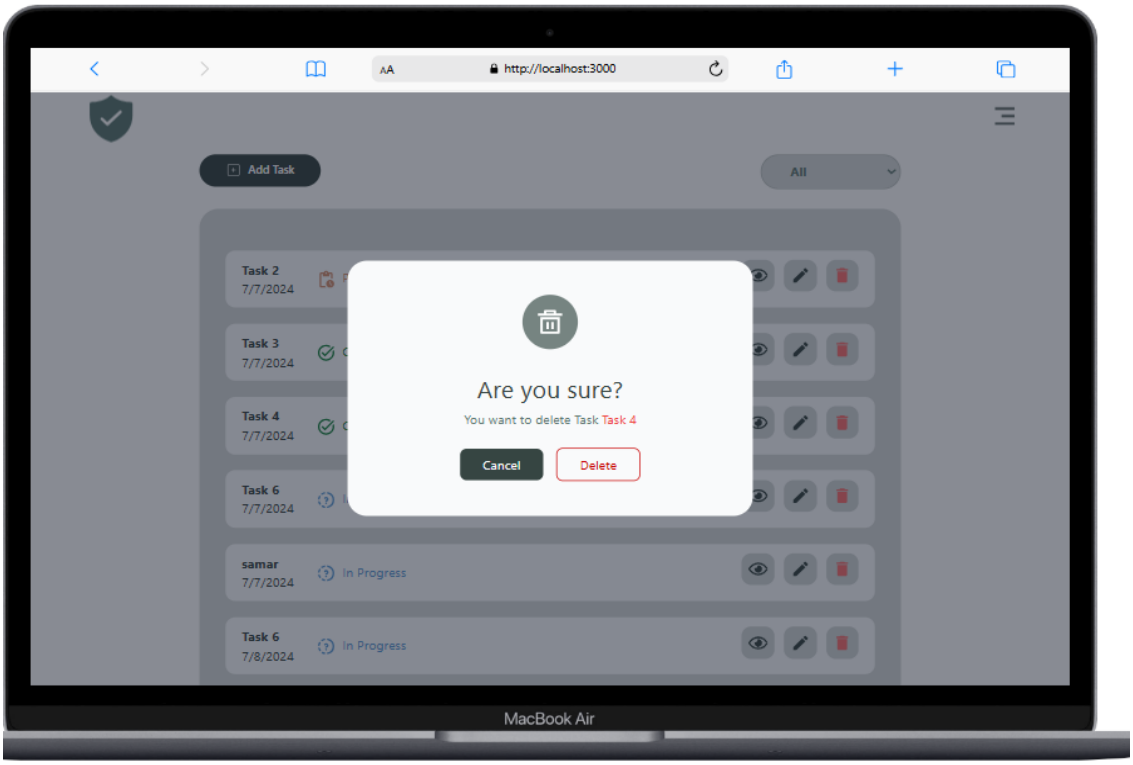
**Endpoint**: http://localhost:3000/tasks/edit/:id



## 7. Task Details

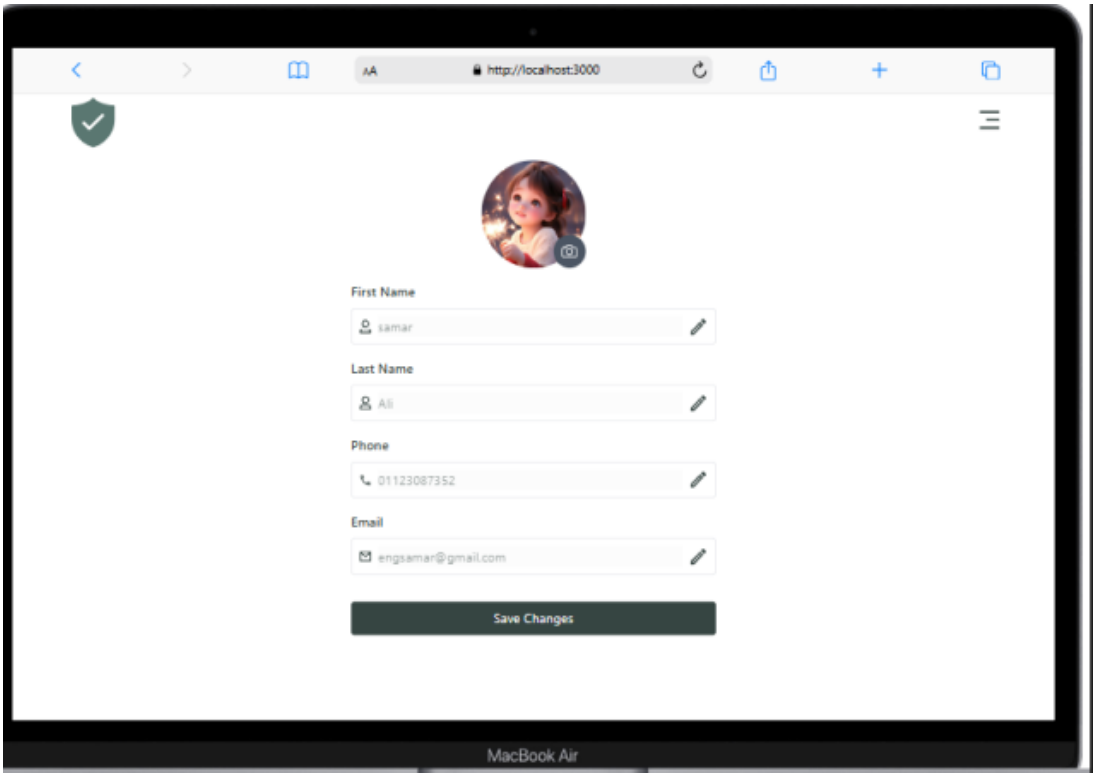



## 8. Conform Delete



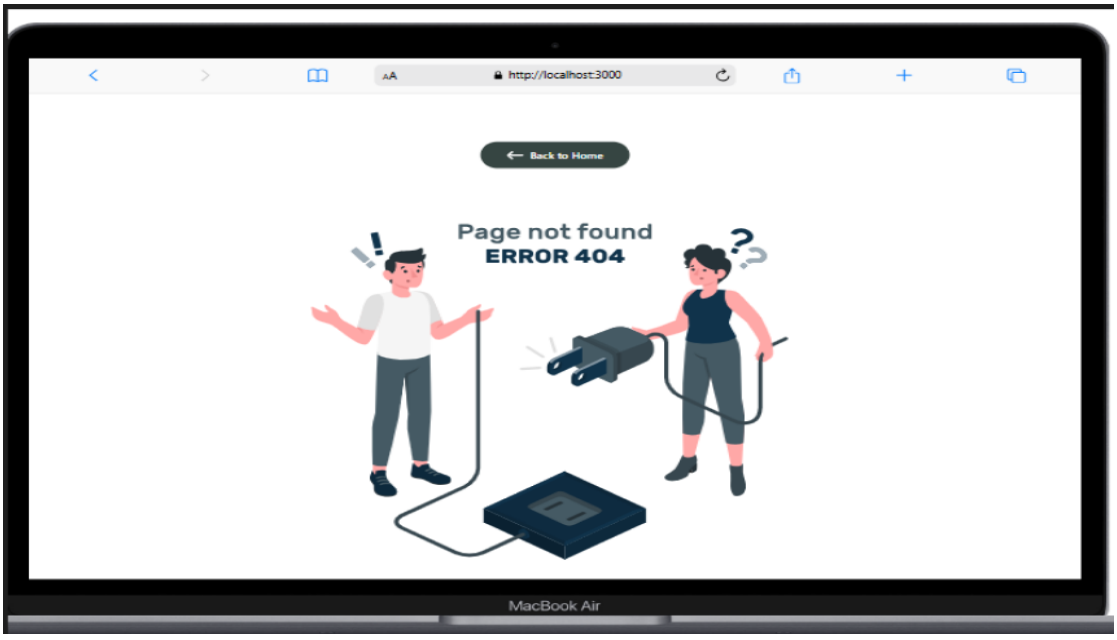
## 9. User Profile Page

**Endpoint**: http://localhost:3000/profile

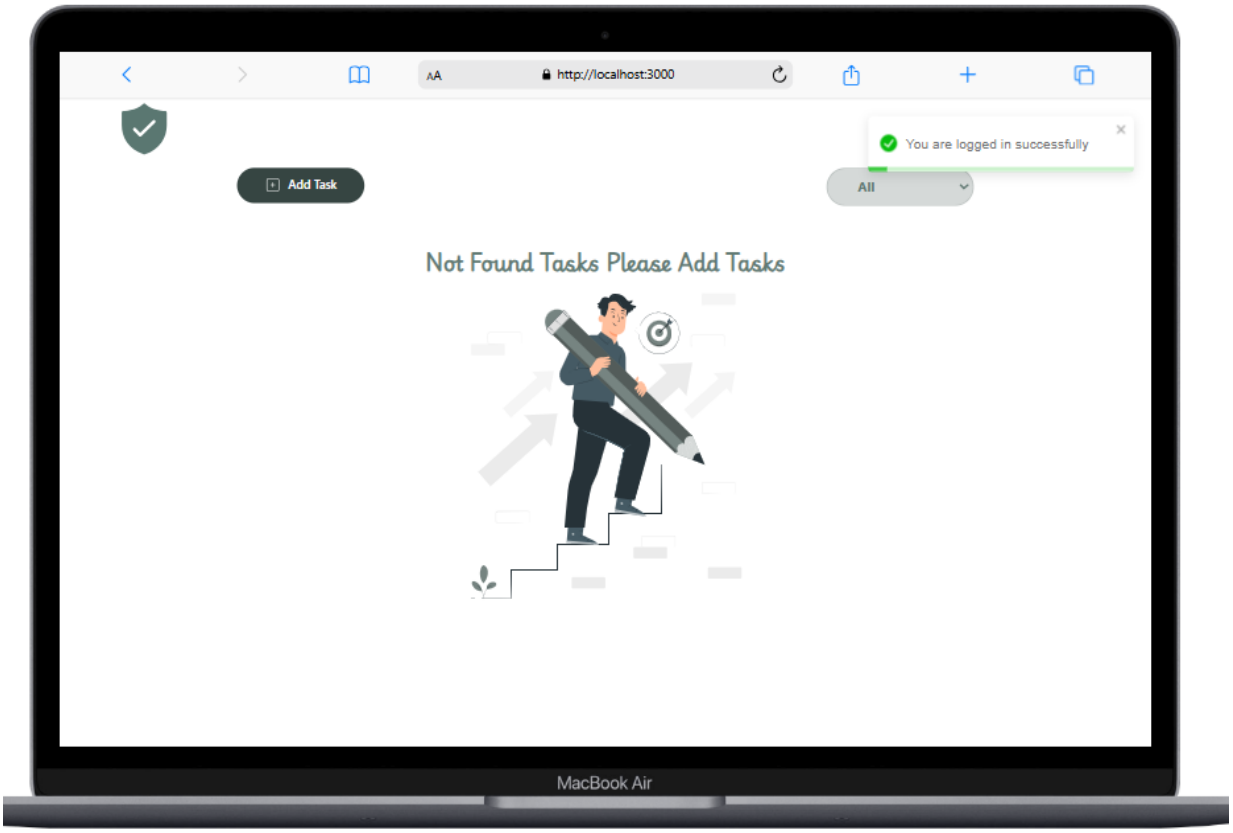



## 10. Page Not Found

**Endpoint**: http://localhost:3000/not-found



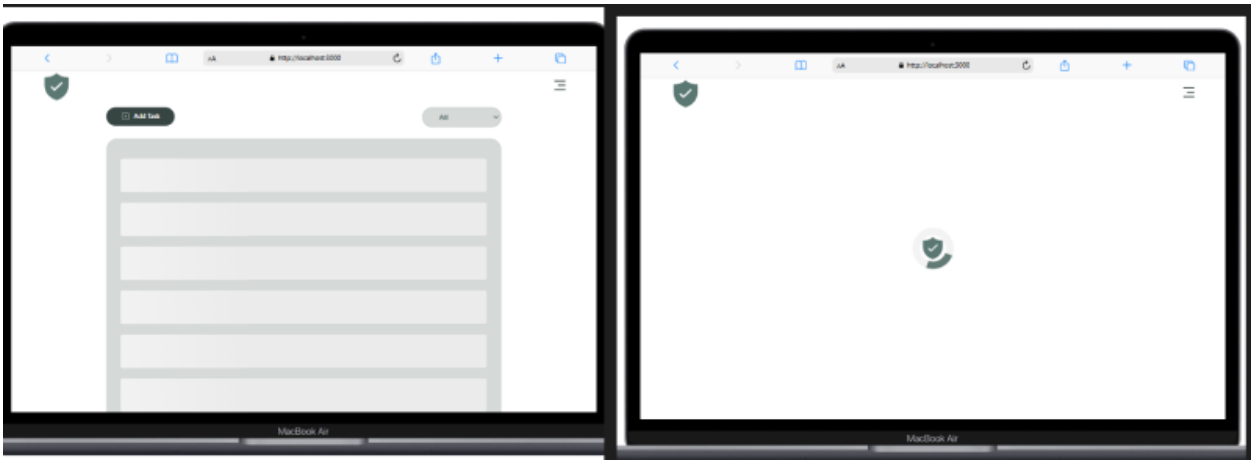
## 11. Page When No Tasks For User




## 12. Loader

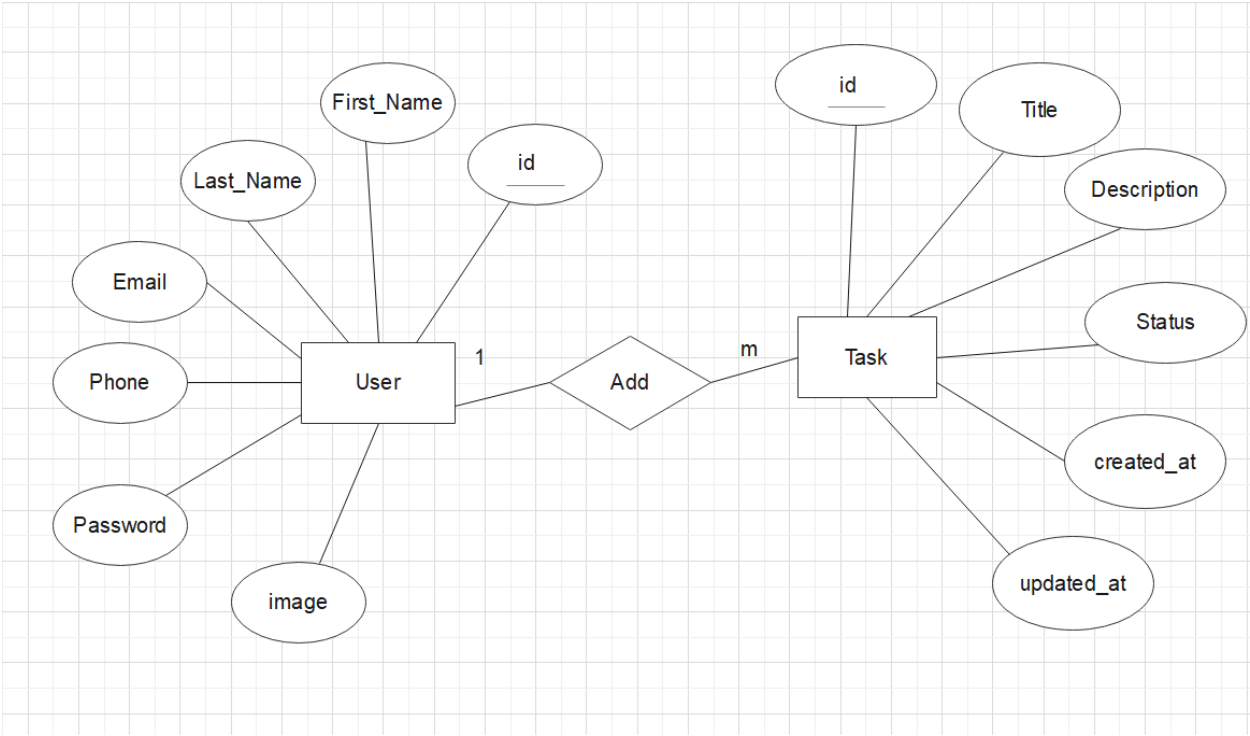
**12.1** Skeleton Loader: Display a skeleton loader while tasks are being fetched and displayed.

**12.2** Custom Loader for Adding or Editing Tasks.

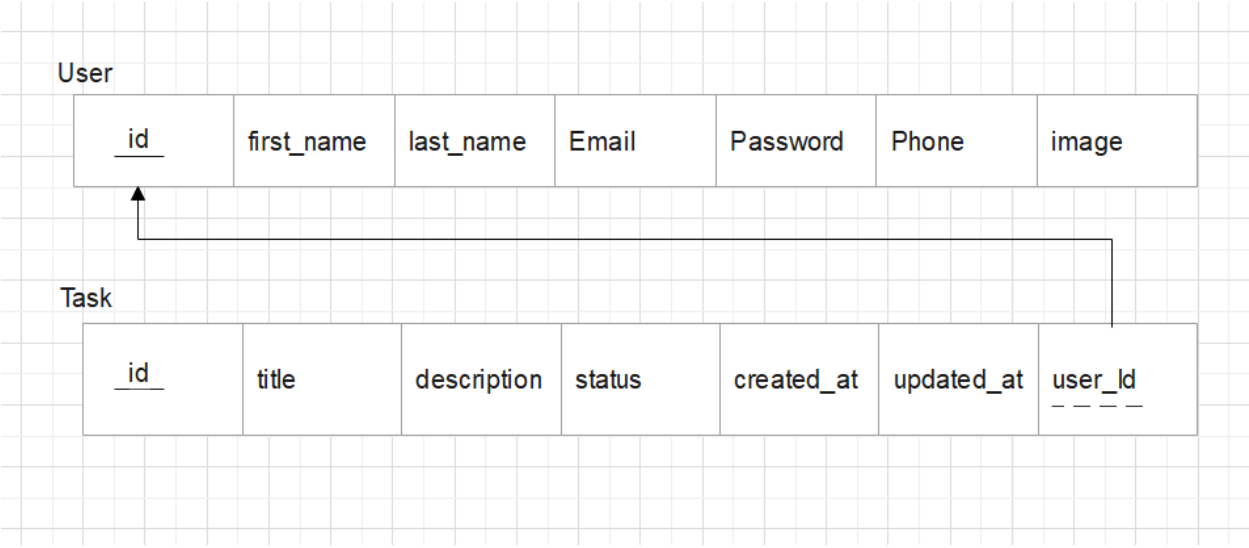


# Database (SQL Server):

## ERD




## Schema



SQl

1. Create the Database

```
CREATE DATABASE Task;
GO
```

2. Use the Database

```
USE Task;
GO
```

3. Create the Users Table

```
CREATE TABLE Users (
    id INT PRIMARY KEY IDENTITY(1,1),
    password VARCHAR(255) NOT NULL,
   image VARCHAR(255) DEFAULT '1.jpg' ,
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
   phone VARCHAR(20)

);
```

4. Create the Tasks Table

```
CREATE TABLE Tasks (
    id INT PRIMARY KEY IDENTITY(1,1),
    title VARCHAR(255) NOT NULL,
    description VARCHAR(255),
    status VARCHAR(50) NOT NULL CHECK (status IN ('Pending',
'In        Progress', 'Completed')),
     userId INT,
    FOREIGN KEY (userId) REFERENCES Users(id) ,
    created_at DATETIME DEFAULT GETDATE(),
    updated_at DATETIME DEFAULT GETDATE()
);
```

# 5. Write SQL Queries to Handle CRUD Operations

## 5.1 Tasks

### 5.1.1 Retrieve All Tasks for a User

```
SELECT * FROM Tasks WHERE userId = @userId;
```

### 5.1.2 Retrieve a Single Task by ID for a User

```
SELECT * FROM Tasks WHERE id = @id AND userId =
@userId;
```

### 5.1.3 Retrieve Tasks with Specific Status for a User

```
SELECT * FROM Tasks WHERE status = @status AND userId
= @userId;
```

### 5.1.4 Create a New Task

```
INSERT INTO Tasks (title, description, status, userId,
created_at, updated_at)
VALUES (@title, @description, @status, @userId,
GETDATE(), GETDATE());
```

### 5.1.5 Update an Existing Task

```
UPDATE Tasks
SET title = @title, description = @description, status
= @status, updated_at = GETDATE()
WHERE id = @id AND userId = @userId;
```

### 5.1.6 Delete a Task UPDATE Tasks

```
DELETE FROM Tasks WHERE id = @id AND userId = @userId;
```

## 5.2 Users

### 5.2.1 Retrieve a User by ID

```
SELECT * FROM Users WHERE id = @id;
```

### 5.2.2 Retrieve a User by Email

```
SELECT * FROM Users WHERE email = @email;
```

### 5.2.3 Update a User

```
UPDATE Users
SET firstname = @firstname, lastname = @lastname,
password = @password, phone = @phone, image = @image
WHERE id = @id;
```

### 5.2.4 Create a New User

```
INSERT INTO Users (firstname, lastname, email,
password)
VALUES (@firstname, @lastname, @email, @password);
```