WorkBuddy

###**This Description has only limited information. Kindly refer to the pdf document given in the learner course for more details****###**

Introduction:

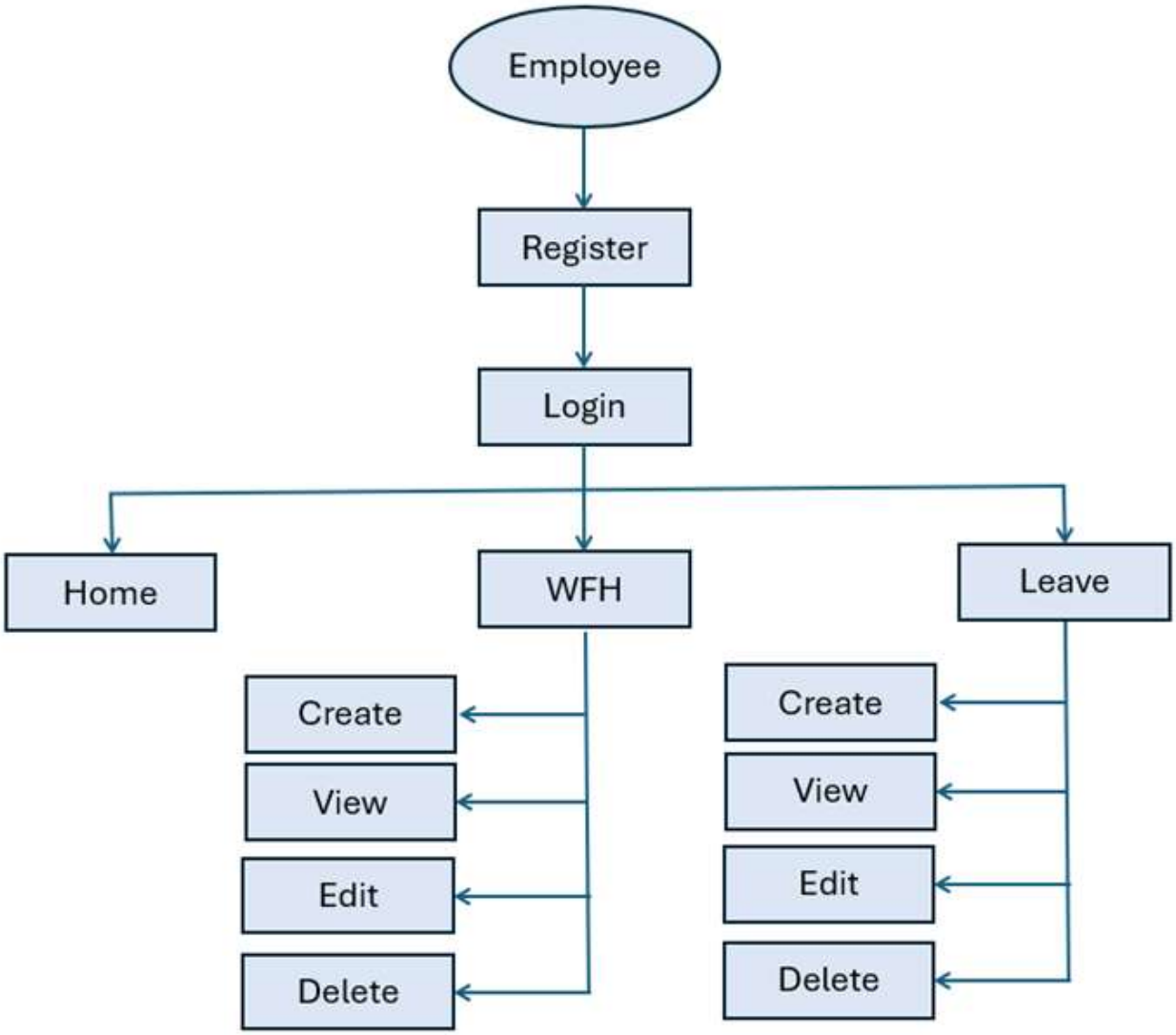
The WorkBuddy Application is a comprehensive software solution designed to streamline and enhance the interaction between employees and their managers within an organization. It aims to facilitate efficient communication, streamline administrative tasks, and provide valuable insights for improved decision-making. By centralizing employee-related processes and data, the application seeks to optimize productivity, foster collaboration, and promote transparency within the workplace. With user-friendly interfaces and robust backend functionality, the Employee-Manager Application empowers both employees and managers to effectively manage their responsibilities and achieve organizational goals.

Users of the System:

1. Manager
2. Employee

<u>Employee Actions</u>

Flow diagram:



Leave Requests: Employees can request leaves for specific durations, providing the start and end dates of their absence along with a reason for the leave.

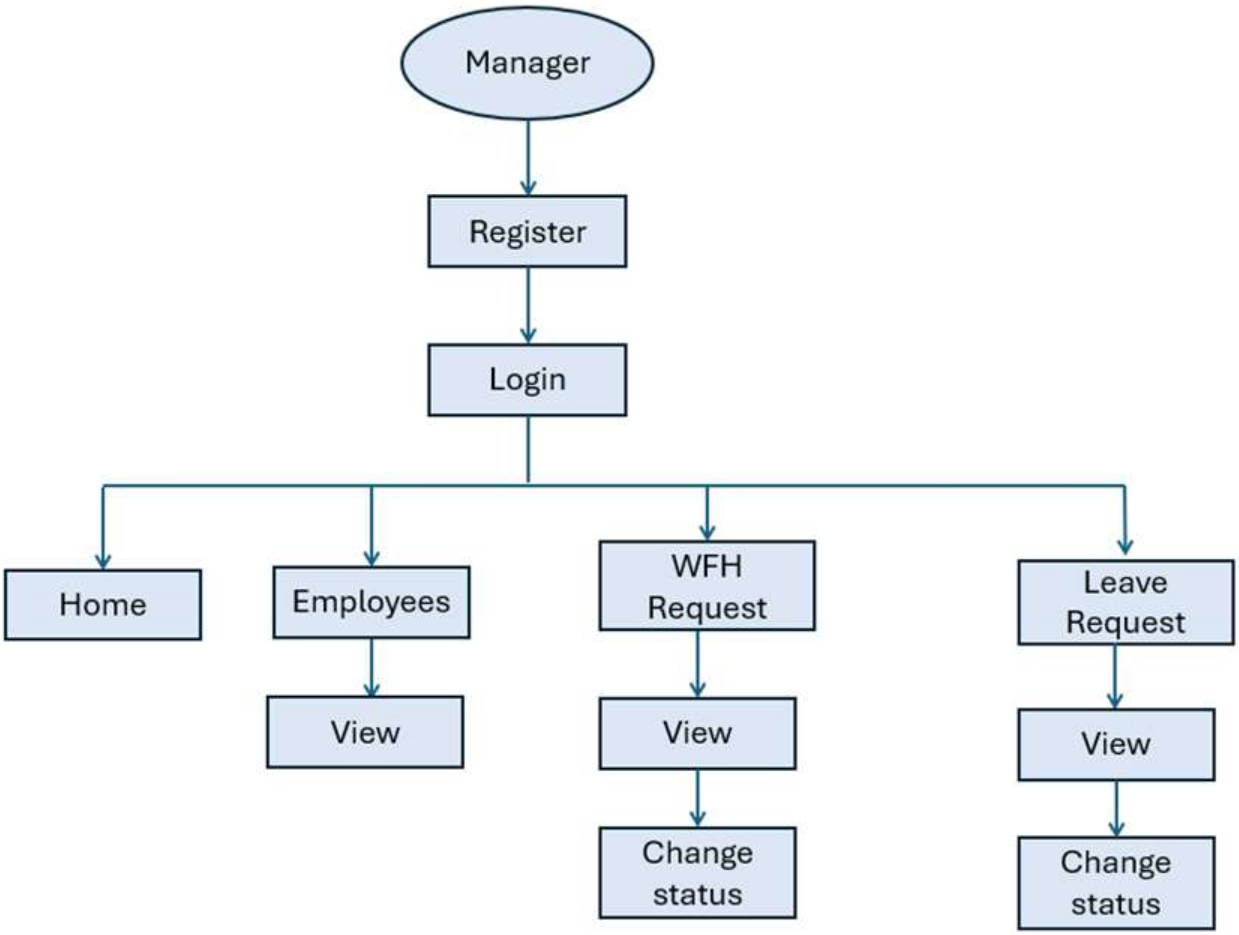
Leave History: Employees can view their past leave requests, including the status of each request (e.g., pending, approved, rejected).

Work From Home (WFH) Requests: Employees can request to work from home for specified periods, providing the start and end dates of their remote work period along with a reason for the request.

WFH History: Employees can access their previous WFH requests and see the status of each request (e.g., pending, approved, rejected).

Manager Actions

Flow diagram:



Approval of Leave Requests: Managers can review leave requests submitted by employees. They have the authority to approve or reject leave requests based on company policies and workload considerations.

Approval of Work From Home (WFH) Requests: Managers can review and approve or reject WFH requests submitted by employees.

Employees: Managers can view all the employees.

Modules of the Application:

Manager:

- Register
- Login
- Home
- Employees
- WFH Request
- Leave Request

Employee:

- Register
- Login
- Home
- WFH
  - Apply WFH
  - My WFH
- Leave

Apply Leave
My Leave

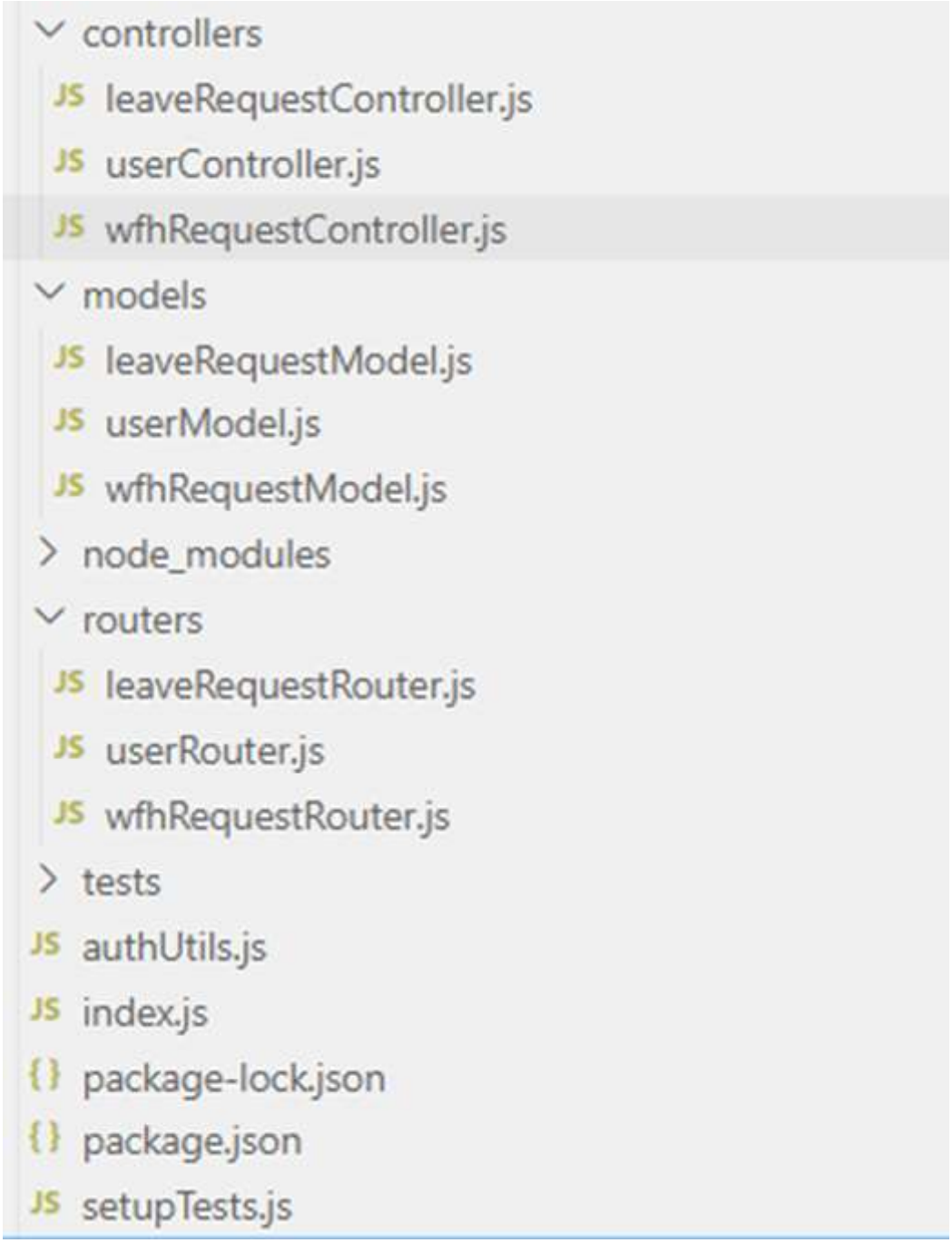
Technology Stack

Front End
Angular

Back End
NodeJs

Backend Requirements:

Create folders named as **models, controllers, router** inside **nodeapp** as mentioned in the below screenshot.



index.js:
- Express Setup: Sets up an Express server.
- Middleware: Adds middleware for parsing request bodies and enabling CORS.
- MongoDB Connection: Connects to a MongoDB database using **mongoose .connect("mongodb://127.0.0.1:27017/dbname")**.
- Routing: Defines routes for handling user, project, and project proposal requests.
- Server Start: Starts the server on port **8080** if the database connection is successful.

Routers:

- Under routers folder define the route's for userModel, leaveRequestModel and wfhRequestModel in the respective file.

Models:
UserSchema model:
Fields:

- userName:
  - Type: String
  - Constraints:
    - Required: true,
    - Unique: true
  - Description: Represents the username of the user.

- email:
  - Type: String
  - Constraints:
    - Required: true,
    - Unique: true
  - Description: Represents the email address of the user.

- mobile:
  - Type: String
  - Constraints:
    - Required: true,
    - Unique: true
  - Description: Represents the mobile number of the user.

- password:
  - Type: String
  - Constraints:
    - Required: true
  - Description: Represents the password of the user.

- role:
  - Type: String
  - Constraints:
    - Required: true
  - Description: Represents the role of the user, such as admin, user, etc.

LeaveRequest Model:
Fields:

- userId:
  - Type: ObjectId
  - Constraints:
    - Required: true
    - Reference: 'User'
  - Description: Represents the ID of the user making the leave request.

- startDate:
  - Type: Date
  - Constraints:
    - Required: true
  - Description: Represents the start date of the leave.

- endDate:
  - Type: Date
  - Constraints:
    - Required: true
  - Description: Represents the end date of the leave.

- reason:
  - Type: String
  - Constraints:
    - Required: true
  - Description: Represents the reason for the leave.

- leaveType:
  - Type: String

- Constraints:
  - Required: true
- Description: Represents the type of leave (e.g., sick leave, casual leave).

- status:
- Type: String
- Constraints:
  - Required: true
- Description: Represents the status of the leave request (e.g., pending, approved, rejected).

- createdOn:
- Type: Date
- Constraints:
  - Default: Date.now
- Description: Represents the date when the leave request was created.

- file:
- Type: String
- Constraints:
  - Required: true
- Description: Represents the file associated with the leave request (e.g., medical certificate).

WfhRequest Model:
Fields:

- userId:
- Type: ObjectId
- Constraints:
  - Required: true
  - Reference: 'User'
- Description: Represents the ID of the user making the WFH request.

- startDate:
- Type: Date
- Constraints:
  - Required: true
- Description: Represents the start date of the WFH request.

- endDate:
- Type: Date
- Constraints:
  - Required: true
- Description: Represents the end date of the WFH request.

- reason:
- Type: String
- Constraints:
  - Required: true
- Description: Represents the reason for the WFH request.

- status:
- Type: String
- Constraints:
  - Required: true
- Description: Represents the status of the WFH request (e.g., pending, approved, rejected).

- createdOn:
- Type: Date
- Constraints:
  - Default: Date.now
- Description: Represents the date when the WFH request was created.

authUtils.js:

- Inside authUtils.js file define the following functions.

generateToken(userId) Function:

1. This function generates a JWT (JSON Web Token) for user authentication.
2. It takes the userId as input and signs it using a secret key 'asdfgewlnclnlhjkl'.
3. The token is configured to expire after 1 hour (expiresIn: '1h').

validateToken(req, res, next) Function:
1. This middleware function validates JWT tokens sent in the request headers.
2. It extracts the token from the Authorization header of the incoming request.
3. The extracted token is then verified using the same secret key 'asdfgewlnclnlhjkl'.
4. If the token is valid, the middleware calls the next() function to proceed to the next middleware in the Express chain.
5. If the token is invalid or missing, it returns a 400 status code with a JSON response containing an "Authentication failed" message.

Controllers

UserController.js:
1. Login user:
a. Function Name: getUserByEmailAndPassword (req, res)
b. Description:
i. Retrieves a user from the database by their email and password using User.findOne({ email, password }).
ii. If a user is found, it generates a token for the user's ID using the generateToken function and constructs a response object containing the user's username, role, token, and ID.
iii. Sends the constructed response object with a status code of 200.
iv. If no user is found, it responds with a status code of 404 and a message stating "User not found".
v. If an error occurs during the process, it responds with a status code of 500 and the error message.

2. Register User:
a. Function Name: addUser(req, res)
b. Description:
i. Adds a new user to the database with the data provided in the request body using User.create(req.body).
ii. Responds with a status code of 200 and a success message if the user is added successfully.
iii. If an error occurs during the process, it responds with a status code of 500 and the error message.

3. Get All Employees:
a. Function Name: getAllEmployees(req, res)
b. Description:
i. Retrieves all users with the role of 'Employee' from the database using User.find({ role: 'Employee' }).
ii. Responds with a status code of 200 and sends the employees in JSON format if successful.
iii. Responds with a status code of 500 along with the error message if an error occurs.

LeaveRequestController.js
1. Get All Leave Requests
a. Function Name: getAllLeaveRequests(req, res)
b. Description:
i. Retrieves all leave requests from the database with pagination, search, and sort functionality.
ii. Uses LeaveRequest.find(searchQuery) with dynamic search, sort, and pagination parameters.
iii. Responds with a status code of 200 and sends the leave requests along with the total number of requests in JSON format if successful.
iv. Responds with a status code of 500 along with the error message if an error occurs.

2. Get Leave Request by ID
a. Function Name: getLeaveRequestById(req, res)
b. Description:
i. Retrieves a specific leave request by its ID from the database using LeaveRequest.findById(id).
ii. Responds with a status code of 200 and sends the leave request details in JSON format if the request is found.
iii. Responds with a status code of 404 along with an error message if the leave request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

3. Add a New Leave Request
a. Function Name: addLeaveRequest(req, res)
b. Description:
i. Adds a new leave request to the database with the details provided in req.body using LeaveRequest.create(req.body).
ii. Responds with a status code of 200 and a success message if the leave request is added successfully.
iii. Responds with a status code of 500 along with the error message if an error occurs.

4. Update an Existing Leave Request
a. Function Name: updateLeaveRequest(req, res)
b. Description:
i. Updates an existing leave request based on its ID with the details provided in req.body using LeaveRequest.findByIdAndUpdate(id, req.body, { new: true }).
ii. Responds with a status code of 200 and sends the updated leave request details in JSON format along with a success message if the update is successful.

iii. Responds with a status code of 404 along with an error message if the leave request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

5. Get Leave Requests by User ID
a. Function Name: getLeaveRequestsByUserId(req, res)
b. Description:
i. Retrieves leave requests for a specific user based on the user ID using LeaveRequest.find({ userId }).
ii. Responds with a status code of 200 and sends the leave requests in JSON format if successful.
iii. Responds with a status code of 500 along with the error message if an error occurs.

6. Delete a Leave Request
a. Function Name: deleteLeaveRequest(req, res)
b. Description:
i. Deletes a leave request from the database based on its ID using LeaveRequest.findByIdAndDelete(id).
ii. Responds with a status code of 200 and a success message if the leave request is deleted successfully.
iii. Responds with a status code of 404 along with an error message if the leave request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

WfhRequestController.js
1. Get All WFH Requests
a. Function Name: getAllWfhRequests(req, res)
b. Description:
i. Retrieves all WFH requests from the database with pagination, search, and sort functionality.
ii. Uses WfhRequest.find(searchQuery) with dynamic search, sort, and pagination parameters.
iii. Responds with a status code of 200 and sends the WFH requests along with the total number of requests in JSON format if successful.
iv. Responds with a status code of 500 along with the error message if an error occurs.

2. Get WFH Request by ID
a. Function Name: getWfhRequestById(req, res)
b. Description:
i. Retrieves a specific WFH request by its ID from the database using WfhRequest.findById(id).
ii. Responds with a status code of 200 and sends the WFH request details in JSON format if the request is found.
iii. Responds with a status code of 404 along with an error message if the WFH request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

3. Get WFH Requests by User ID
a. Function Name: getWfhRequestsByUserId(req, res)
b. Description:
i. Retrieves WFH requests for a specific user based on the user ID using WfhRequest.find({ userId }).
ii. Responds with a status code of 200 and sends the WFH requests in JSON format if successful.
iii. Responds with a status code of 500 along with the error message if an error occurs.

4. Add a New WFH Request
a. Function Name: addWfhRequest(req, res)
b. Description:
i. Adds a new WFH request to the database with the details provided in req.body using WfhRequest.create(req.body).
ii. Responds with a status code of 200 and a success message if the WFH request is added successfully.
iii. Responds with a status code of 500 along with the error message if an error occurs.

5. Update an Existing WFH Request
a. Function Name: updateWfhRequest(req, res)
b. Description:
i. Updates an existing WFH request based on its ID with the details provided in req.body using WfhRequest.findByIdAndUpdate(id, req.body, { new: true }).
ii. Responds with a status code of 200 and sends the updated WFH request details in JSON format along with a success message if the update is successful.
iii. Responds with a status code of 404 along with an error message if the WFH request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

6. Delete a WFH Request
a. Function Name: deleteWfhRequest(req, res)
b. Description:
i. Deletes a WFH request from the database based on its ID using WfhRequest.findByIdAndDelete(id).
ii. Responds with a status code of 200 and a success message if the WFH request is deleted successfully.
iii. Responds with a status code of 404 along with an error message if the WFH request is not found.
iv. Responds with a status code of 500 along with the error message if an error occurs.

Endpoints

UserController.js

Register User(Employee)

POST http://localhost:8080/user/signup Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "userName": "employee1",
3   "email": "employee1@example.com",
4   "mobile": "1234567890",
5   "password": "password123",
6   "role": "Employee"
7 }
```

Status: 200 OK Size: 21 Bytes Time: 10 ms

Response Headers 7 Cookies Results Docs

```
1 {
2   "message": "Success"
3 }
```

(Manager)

POST http://localhost:8080/user/signup Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "userName": "manager1",
3   "email": "manager1@example.com",
4   "mobile": "0987654321",
5   "password": "password123",
6   "role": "Manager"
7 }
```

Status: 200 OK Size: 21 Bytes Time: 12 ms

Response Headers 7 Cookies Results Docs

```
1 {
2   "message": "Success"
3 }
```

Login User

POST http://localhost:8080/user/login Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "email": "employee1@example.com",
3   "password": "password123"
4 }
```

Status: 200 OK Size: 261 Bytes Time: 10 ms

Response Headers 7 Cookies Results Docs


```
1 {
2   "userName": "employee1",
3   "role": "Employee",
4   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJ1c2VySWQiOiI2NjVlYjUyNmNiMWFhOTEwMWY5MzFiODUiLCJpYXQiOjE3MTc0ODI5MDgsImV4cCI6MTcxNzQ4
      NjUwOH0.Q-JAvW0QRo016qFJsaUPSXirw6QSqTezqXGr-zwYeWQ",
5   "id": "665eb526cb1aa9101f931b85"
6 }
```


GetAllEmployees

GET http://localhost:8080/user/getAllEmployees Send

Status: 200 OK Size: 459 Bytes Time: 9 ms

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Response Headers 7 Cookies Results Docs


```
[
  {
    "_id": "665af38a0e1ad604c2dbf6c1",
    "userName": "demo",
    "email": "emp@gmail.com",
    "mobile": "9688499919",
    "password": "Demo@123",
    "role": "Employee",
    "__v": 0
  },
  {
    "_id": "665ea9d9cb1aa9101f931b52",
    "userName": "emp2",
    "email": "emp2@gmail.com",
    "mobile": "4256789222",
    "password": "Demo@123",
    "role": "Employee",
    "__v": 0
  },
  {
    "_id": "665eb526cb1aa9101f931b85",
    "userName": "employee1",
    "email": "employee1@example.com",
    "mobile": "1234567890",
    "password": "password123",
    "role": "Employee",
    "__v": 0
  }
]
```


LeaveRequestController.js

Add a New Leave Request

POST http://localhost:8080/leave/addLeaveRequest Send

Status: 200 OK Size: 46 Bytes Time: 9 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format


```
{
  "userId": "665eb526cb1aa9101f931b85",
  "startDate": "2022-04-01T00:00:00.000Z",
  "endDate": "2022-04-05T00:00:00.000Z",
  "reason": "Going on vacation",
  "leaveType": "Paid Leave",
  "status": "Pending",
  "file": "data:image/jpeg;base64"
}
```


Response Headers 7 Cookies Results Docs


```
{
  "message": "Leave Request Added Successfully"
}
```


Get All Leave Requests

POST http://localhost:8080/leave/getAllLeaveRequests Send

Status: 200 OK Size: 957.18 KB Time: 76 ms

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

Response Headers 7 Cookies Results Docs

Copy


```
{
  "data": [
    {
      "_id": "665af47c0e1ad604c2dbf6cd",
      "userId": "665af38a0e1ad604c2dbf6c1",
      "startDate": "2024-06-16T00:00:00.000Z",
      "endDate": "2024-06-21T00:00:00.000Z",
      "reason": "due to family function",
      "leaveType": "PTO",
      "status": "Pending",
      "file": "
      "createdOn": "2024-06-01T10:14:20.701Z",
      "__v": 0
    },
    {
      "_id": "665af53e0e1ad604c2dbf6db",
      "userId": "665af38a0e1ad604c2dbf6c1",
      "startDate": "2024-06-16T00:00:00.000Z",
      "endDate": "2024-06-16T00:00:00.000Z",
      "reason": "fever",
      "leaveType": "Unpaid",
      "status": "Approved",
      "file": "
      "createdOn": "2024-06-01T10:17:34.491Z",
      "__v": 0
    },
    {
      "_id": "665eb7c0cb1aa9101f931b8c",
      "userId": "665eb526cb1aa9101f931b85",
      "startDate": "2022-04-01T00:00:00.000Z",
      "endDate": "2022-04-05T00:00:00.000Z",
      "reason": "Going on vacation",
      "leaveType": "Paid Leave",
      "status": "Pending",
      "file": "data:image/jpeg;base64",
      "createdOn": "2024-06-04T06:44:16.230Z",
      "__v": 0
    },
    {
```

POST http://localhost:8080/leave/getAllLeaveRequests Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{
  "page": 1,
  "pageSize": 5,
  "searchValue": "vacation",
  "sortValue": "asc",
  "statusFilter": "Pending",
  "sortBy": "createdOn"
}
```

Status: 200 OK Size: 319 Bytes Time: 179 ms

Response Headers 7 Cookies Results Docs

```
{
  "data": [
    {
      "_id": "665eb7c0cb1aa9101f931b8c",
      "userId": "665eb526cb1aa9101f931b85",
      "startDate": "2022-04-01T00:00:00.000Z",
      "endDate": "2022-04-05T00:00:00.000Z",
      "reason": "Going on vacation",
      "leaveType": "Paid Leave",
      "status": "Pending",
      "file": "data:image/jpeg;base64",
      "createdOn": "2024-06-04T06:44:16.230Z",
      "__v": 0
    }
  ],
  "total": 1
}
```

Get Leave Request by ID

GET http://localhost:8080/leave/getLeaveRequestById/665eb7c0cb1aa9101f931b8c Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 298 Bytes Time: 9 ms

Response Headers 7 Cookies Results Docs

```
{
  "_id": "665eb7c0cb1aa9101f931b8c",
  "userId": "665eb526cb1aa9101f931b85",
  "startDate": "2022-04-01T00:00:00.000Z",
  "endDate": "2022-04-05T00:00:00.000Z",
  "reason": "Going on vacation",
  "leaveType": "Paid Leave",
  "status": "Pending",
  "file": "data:image/jpeg;base64",
  "createdOn": "2024-06-04T06:44:16.230Z",
  "__v": 0
}
```

Update an Existing Leave Request

PUT http://localhost:8080/leave/updateLeaveRequest/665eb7c0cb1aa9101f931b8c Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{
  "status": "Approved"
}
```

Status: 200 OK Size: 363 Bytes Time: 13 ms

Response Headers 7 Cookies Results Docs

```
{
  "message": "Leave Request Updated Successfully",
  "leaveRequest": {
    "_id": "665eb7c0cb1aa9101f931b8c",
    "userId": "665eb526cb1aa9101f931b85",
    "startDate": "2022-04-01T00:00:00.000Z",
    "endDate": "2022-04-05T00:00:00.000Z",
    "reason": "Going on vacation",
    "leaveType": "Paid Leave",
    "status": "Approved",
    "file": "data:image/jpeg;base64",
    "createdOn": "2024-06-04T06:44:16.230Z",
    "__v": 0
  }
}
```

Get Leave Requests by User ID

GET http://localhost:8080/leave/getLeaveRequestsByUserId/665af38a0e1ad604c2dbf6c1 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 956.58 KB Time: 53 ms

Response Headers 7 Cookies Results Docs


```
[
  {
    "_id": "665af47c0e1ad604c2dbf6cd",
    "userId": "665af38a0e1ad604c2dbf6c1",
    "startDate": "2024-06-16T00:00:00.000Z",
    "endDate": "2024-06-21T00:00:00.000Z",
    "reason": "due to family function",
    "leaveType": "PTO",
    "status": "Pending",
    "file": ".
    "createdOn": "2024-06-01T10:14:20.701Z",
    "__v": 0
  },
  {
    "_id": "665af53e0e1ad604c2dbf6db",
    "userId": "665af38a0e1ad604c2dbf6c1",
    "startDate": "2024-06-16T00:00:00.000Z",
    "endDate": "2024-06-16T00:00:00.000Z",
    "reason": "fever",
    "leaveType": "Unpaid",
    "status": "Approved",
    "file": "
    "createdOn": "2024-06-01T10:17:34.491Z",
    "__v": 0
  }
]
```


Delete a Leave Request

DELETE http://localhost:8080/leave/deleteLeaveRequest/665eb7c0cb1aa9101f931b8c Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1
```

Status: 200 OK Size: 48 Bytes Time: 8 ms

Response Headers 7 Cookies Results Docs


```
{
  "message": "Leave Request Deleted Successfully"
}
```


WfhRequestController.js

Add a New WFH Request

POST http://localhost:8080/wfh/addWfhRequest Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format


```
{
  "userId": "665af38a0e1ad604c2dbf6c1",
  "startDate": "2022-01-01T00:00:00.000Z",
  "endDate": "2022-01-07T00:00:00.000Z",
  "reason": "Taking a vacation",
  "status": "Pending"
}
```


Status: 200 OK Size: 44 Bytes Time: 26 ms

Response Headers 7 Cookies Results Docs


```
{
  "message": "WFH Request Added Successfully"
}
```


Get All WFH Requests

POST http://localhost:8080/wfh/getAllWfhRequests Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 1.19 KB Time: 11 ms

Response Headers 7 Cookies Results Docs

Copy


```
{
  "data": [
    {
      "_id": "665af43e0e1ad604c2dbf6c6",
      "userId": "665af38a0e1ad604c2dbf6c1",
      "startDate": "2024-06-12T00:00:00.000Z",
      "endDate": "2024-06-12T00:00:00.000Z",
      "reason": "Due to rain requesting WFH.",
      "status": "Approved",
      "createdOn": "2024-06-01T10:13:18.029Z",
      "__v": 0
    },
    {
      "_id": "665af5200e1ad604c2dbf6d8",
      "userId": "665af38a0e1ad604c2dbf6c1",
      "startDate": "2024-06-02T00:00:00.000Z",
      "endDate": "2024-06-04T00:00:00.000Z",
      "reason": "summa",
      "status": "Rejected",
      "createdOn": "2024-06-01T10:17:04.353Z",
      "__v": 0
    },
```


POST http://localhost:8080/wfh/getAllWfhRequests Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{
  "page": 1,
  "pageSize": 5,
  "searchValue": "vacation",
  "sortValue": "asc",
  "statusFilter": "Pending",
  "sortBy": "createdOn"
}
```

Status: 200 OK Size: 262 Bytes Time: 18 ms

Response Headers 7 Cookies Results Docs


```
{
  "data": [
    {
      "_id": "665ec3c2e3aa0dd9f5863280",
      "userId": "665af38a0e1ad604c2dbf6c1",
      "startDate": "2022-01-01T00:00:00.000Z",
      "endDate": "2022-01-07T00:00:00.000Z",
      "reason": "Taking a vacation",
      "status": "Pending",
      "createdOn": "2024-06-04T07:35:30.072Z",
      "__v": 0
    }
  ],
  "total": 1
}
```


Get WFH Request by ID

GET http://localhost:8080/wfh/getWfhRequestById/665ec3c2e3aa0dd9f5863280 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 241 Bytes Time: 7 ms

Response Headers 7 Cookies Results Docs


```
{
  "_id": "665ec3c2e3aa0dd9f5863280",
  "userId": "665af38a0e1ad604c2dbf6c1",
  "startDate": "2022-01-01T00:00:00.000Z",
  "endDate": "2022-01-07T00:00:00.000Z",
  "reason": "Taking a vacation",
  "status": "Pending",
  "createdOn": "2024-06-04T07:35:30.072Z",
  "__v": 0
}
```


Get WFH Requests by User ID

GET http://localhost:8080/wfh/getWfhRequestsByUserId/665af38a0e1ad604c2dbf6c1 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 963 Bytes Time: 6 ms

Response Headers 7 Cookies Results Docs


```
[
  {
    "_id": "665af43e0e1ad604c2dbf6c6",
    "userId": "665af38a0e1ad604c2dbf6c1",
    "startDate": "2024-06-12T00:00:00.000Z",
    "endDate": "2024-06-12T00:00:00.000Z",
    "reason": "Due to rain requesting WFH.",
    "status": "Approved",
    "createdOn": "2024-06-01T10:13:18.029Z",
    "__v": 0
  },
  {
    "_id": "665af5200e1ad604c2dbf6d8",
    "userId": "665af38a0e1ad604c2dbf6c1",
    "startDate": "2024-06-02T00:00:00.000Z",
    "endDate": "2024-06-04T00:00:00.000Z",
    "reason": "summa",
    "status": "Rejected",
    "createdOn": "2024-06-01T10:17:04.353Z",
    "__v": 0
  },
  {
    "_id": "665ec3c2e3aa0dd9f5863280",
    "userId": "665af38a0e1ad604c2dbf6c1",
    "startDate": "2022-01-01T00:00:00.000Z",
    "endDate": "2022-01-07T00:00:00.000Z",
    "reason": "Taking a vacation",
    "status": "Pending",
    "createdOn": "2024-06-04T07:35:30.072Z",
    "__v": 0
  },
```

Update an Existing WFH Request

PUT http://localhost:8080/wfh/updateWfhRequest/665ec3c2e3aa0dd9f5863280 Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "status":"Approved"
3 }
```

Status: 200 OK Size: 302 Bytes Time: 10 ms

Response Headers 7 Cookies Results Docs

```
1  {
2    "message": "WFH Request Updated Successfully",
3    "wfhRequest": {
4      "_id": "665ec3c2e3aa0dd9f5863280",
5      "userId": "665af38a0e1ad604c2dbf6c1",
6      "startDate": "2022-01-01T00:00:00.000Z",
7      "endDate": "2022-01-07T00:00:00.000Z",
8      "reason": "Taking a vacation",
9      "status": "Approved",
10     "createdOn": "2024-06-04T07:35:30.072Z",
11     "__v": 0
12   }
13 }
```

Delete a WFH Request

DELETE http://localhost:8080/wfh/deleteWfhRequest/665ec3c2e3aa0dd9f5863280 Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1
```

Status: 200 OK Size: 46 Bytes Time: 8 ms

Response Headers 7 Cookies Results Docs

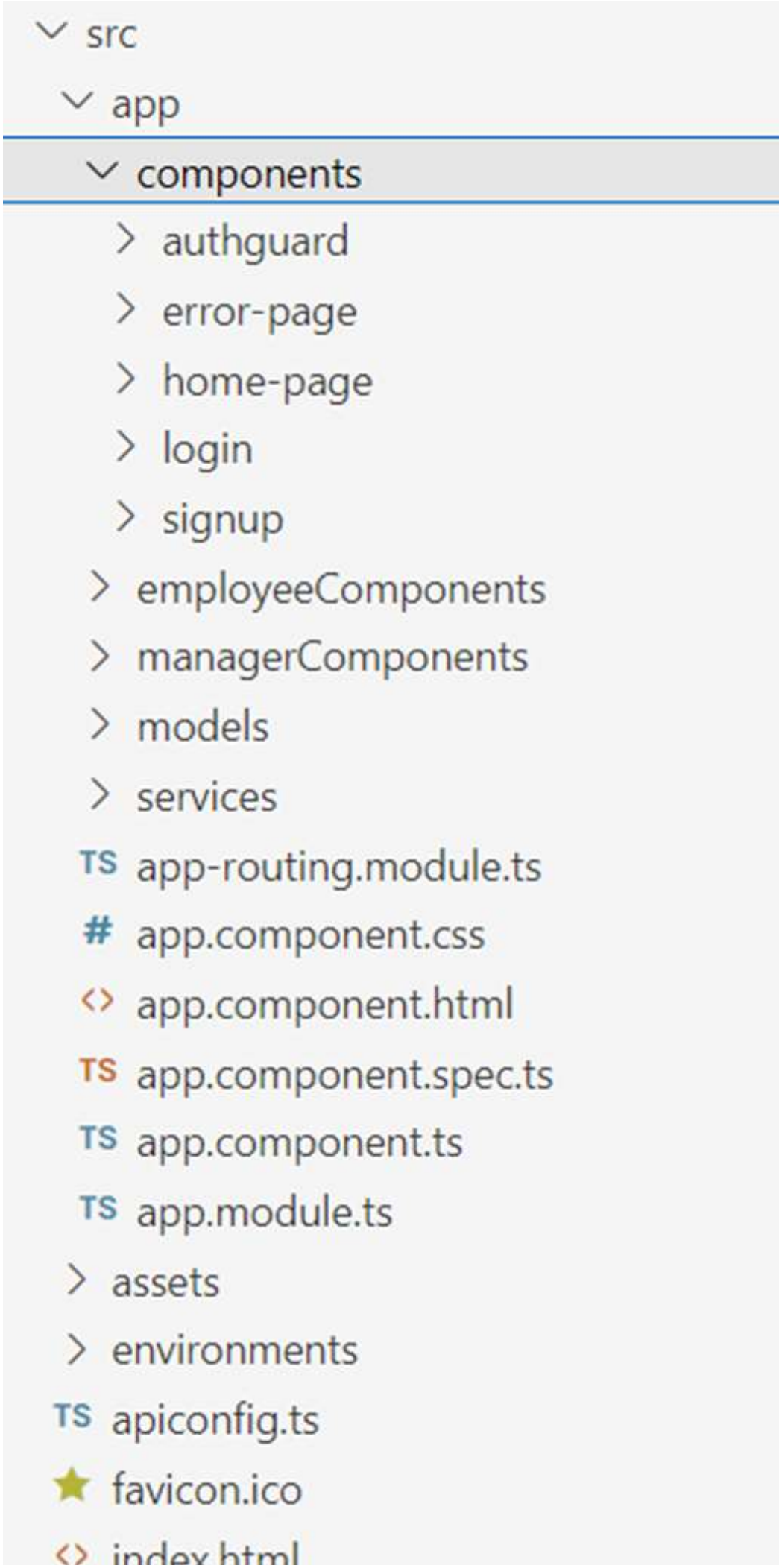
```
1 {
2   "message": "WFH Request Deleted Successfully"
3 }
```

Frontend Requirements:

- Create a folder named components inside the src to store all common components. (Refer to project structure screenshots).
- Create a folder named employeeComponents inside the src to store all components for Employee. (Refer project structure screenshots).
- Create a folder named managerComponents inside the src to store all components for Manager. (Refer project structure screenshots).
- Create components, models, and services mentioned in the project folder screenshot also include your own components based on the application requirements.

Project Folder Screenshot:

Components:

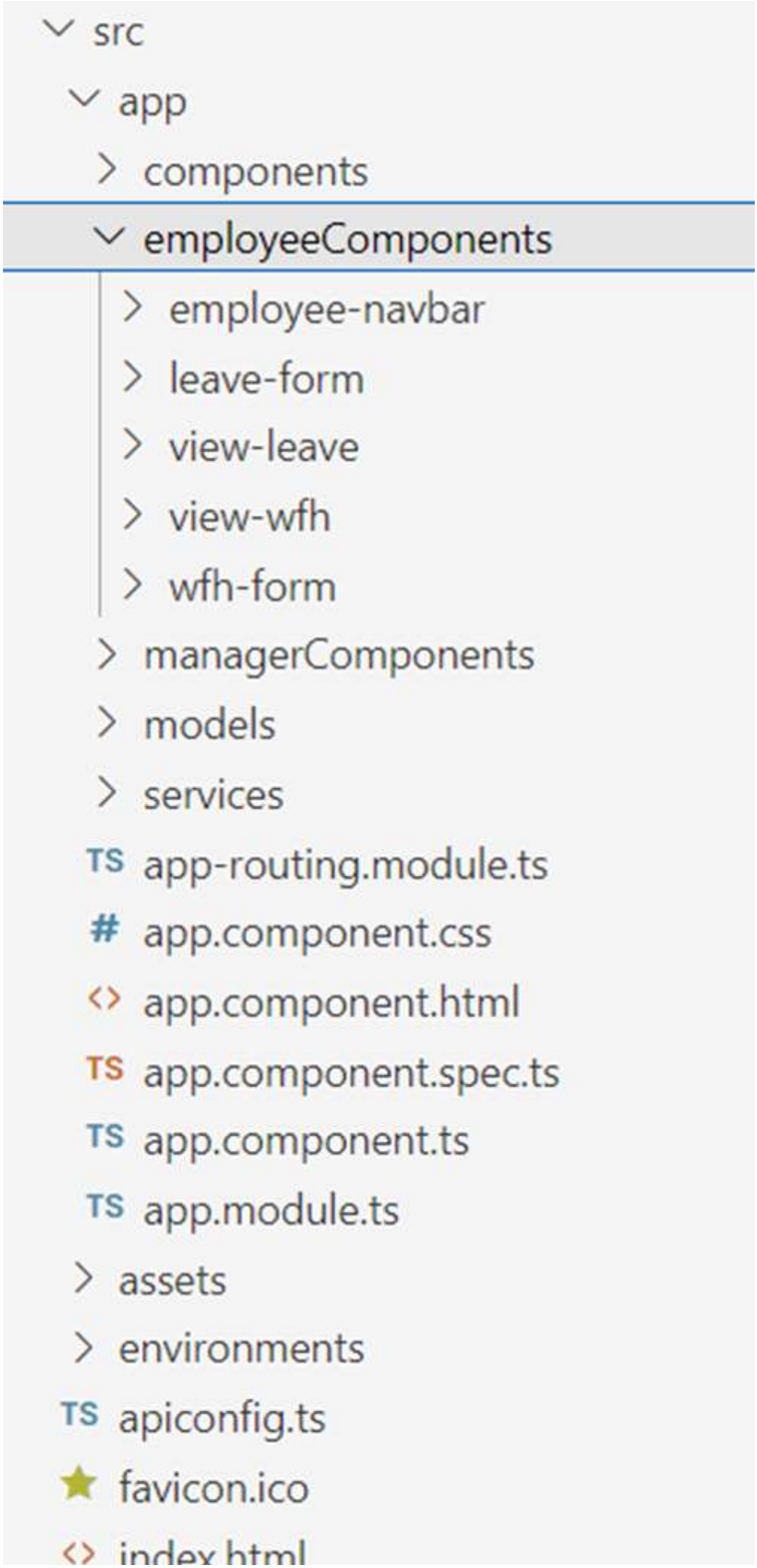

src
app
components
authguard
error-page
home-page
login
signup
employeeComponents
managerComponents
models
services
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
apiconfig.ts
favicon.ico
index.html

```
index.html
```

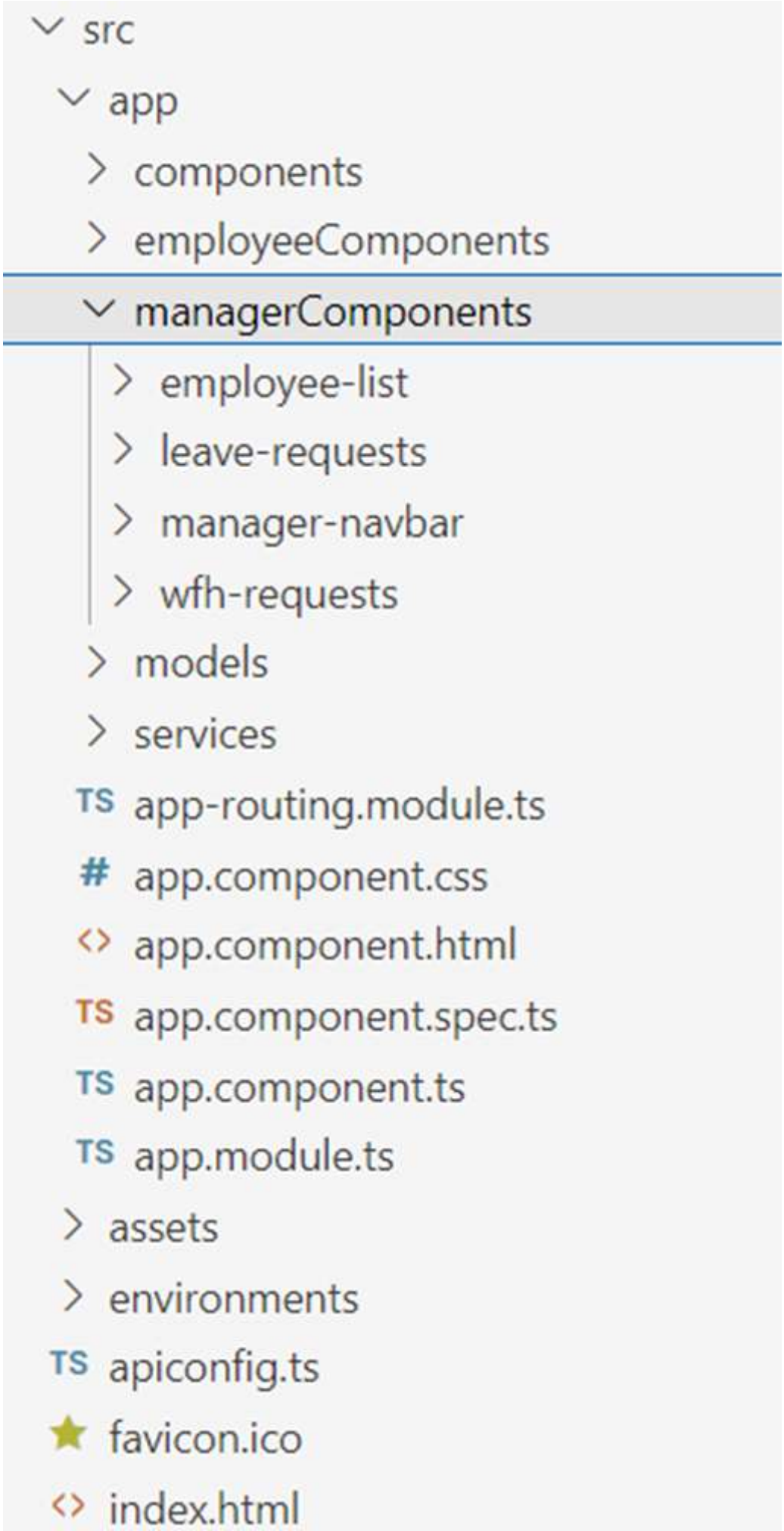
Structure:

- src
  - app
    - components
      - authguard
        - auth.guard.spec.ts
        - auth.guard.ts
      - error-page
        - error-page.component.css
        - error-page.component.html
        - error-page.component.spec.ts
        - error-page.component.ts
      - home-page
        - home-page.component.css
        - home-page.component.html
        - home-page.component.spec.ts
        - home-page.component.ts
      - login
        - login.component.css
        - login.component.html
        - login.component.spec.ts
        - login.component.ts
      - signup
        - signup.component.css

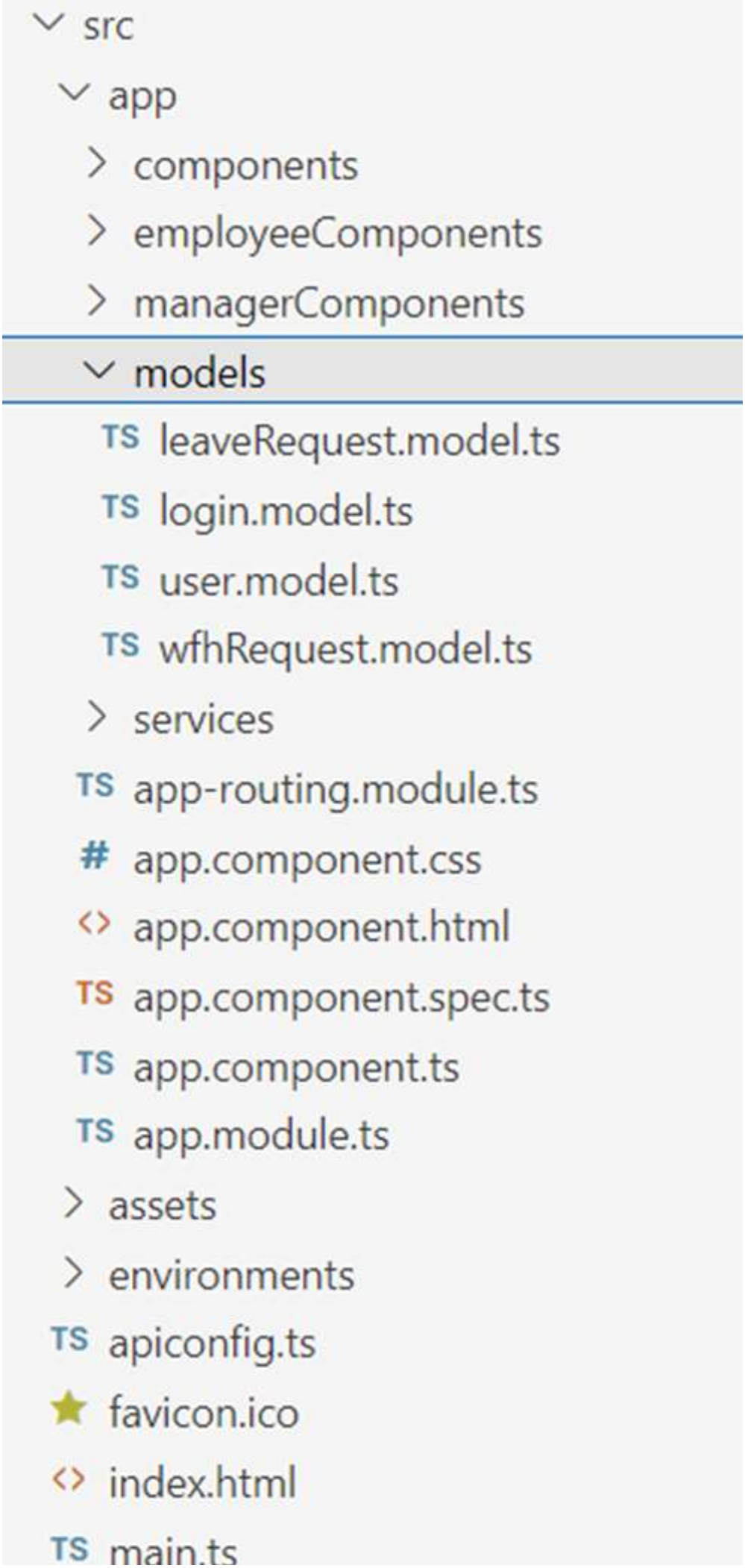
EmployeeComponents:
Inside the **EmployeeComponents** folder create the components mentioned in the below image:


src
app
components
employeeComponents
employee-navbar
leave-form
view-leave
view-wfh
wfh-form
managerComponents
models
services
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
apiconfig.ts
favicon.ico
index.html

ManagerComponents:
Inside the **ManagerComponents** folder create the components mentioned in the below image:

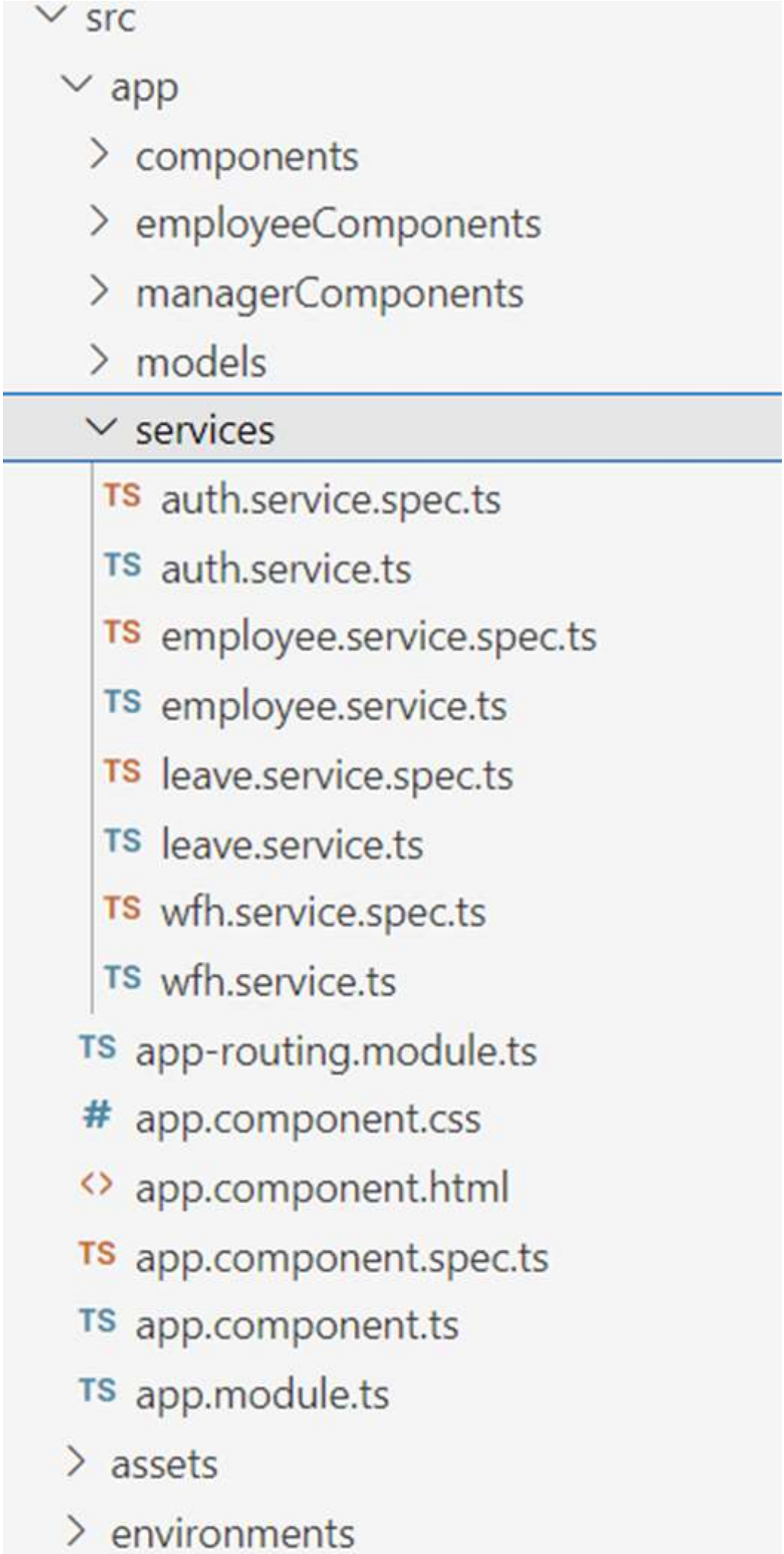

src
app
components
employeeComponents
managerComponents
employee-list
leave-requests
manager-navbar
wfh-requests
models
services
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
apiconfig.ts
favicon.ico
index.html

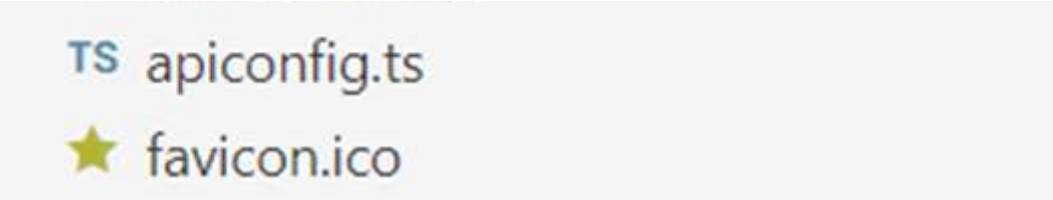
Models:


src
app
components
employeeComponents
managerComponents
models
leaveRequest.model.ts
login.model.ts
user.model.ts
wfhRequest.model.ts
services
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
apiconfig.ts
favicon.ico
index.html
main.ts

TS polyfills.ts

Services:


src
app
components
employeeComponents
managerComponents
models
services
auth.service.spec.ts
auth.service.ts
employee.service.spec.ts
employee.service.ts
leave.service.spec.ts
leave.service.ts
wfh.service.spec.ts
wfh.service.ts
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments



Frontend services:
- Declare a public property apiUrl to store the backend URL in all the services.
- For example, public apiUrl = 'http://localhost:8080'. Instead of 'localhost', replace it with the URL of your workspace port 8080 URL.
- For the API's to be used please refer the API Table.
- Authorized token to be passed in headers for all end points.

1. **AuthService(auth.service.ts):**

a. Create a service name as auth inside app/services folder to implement the following functions.
b. Methods Overview:
i. register(user: User): Observable<any>:
Use this method to register a new user. It sends a POST request to the '/user/signup' endpoint with the user data provided as the body.
ii. login(login : Login): Observable<any>:
This method is used to authenticate a user by logging them in. It sends a POST request to the '/user/login' endpoint with the user's email and password. Upon successful login, it stores the JWT token, role, id, userName in localStorage and updates the user's role and ID using BehaviorSubjects.

EmployeeService (Services/employee.service.ts)

Create a service named **employee** inside the app/services folder and implement the following functions in it.

Methods Overview:
- **getAllEmployees()**: Observable<User[]>:
  - Use this method to fetch all employees from the server.
  - It sends a GET request to the /user/getAllEmployees endpoint with the authorization token prefixed with 'Bearer' stored in localStorage.

LeaveService (Services/leave.service.ts)
Create a service named leave inside the app/services folder and implement the following functions in it.

Methods Overview:
- **getLeaveRequestById(id: string)**: Observable<LeaveRequest>:
  - This method retrieves a specific leave request by its ID.
  - It sends a GET request to the /leave/getLeaveRequestById/${id} endpoint with the leave request ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **addLeaveRequest(requestObject: LeaveRequest)**: Observable<any>:
  - Use this method to add a new leave request.
  - It sends a POST request to the /leave/addLeaveRequest endpoint with the leave request data provided as the body and the authorization token prefixed with 'Bearer' stored in localStorage.

- **updateLeaveRequest(id: string, requestObject: LeaveRequest)**: Observable<any>:
  - Call this method to update an existing leave request.
  - It sends a PUT request to the /leave/updateLeaveRequest/${id} endpoint with the leave request ID, updated leave request data provided as the body, and the authorization token prefixed with 'Bearer' stored in localStorage.

- **getLeaveRequestsByUserId(userId: string)**: Observable<LeaveRequest[]>:
  - This method retrieves all leave requests submitted by a specific user.
  - It sends a GET request to the /leave/getLeaveRequestsByUserId/${userId} endpoint with the user ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **deleteLeaveRequest(leaveId: string)**: Observable<any>:
  - This method deletes a leave request with the specified ID.
  - It sends a DELETE request to the /leave/deleteLeaveRequest/${leaveId} endpoint with the leave request ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **getAllLeaveRequests(reqObj: any)**: Observable<any>:
  - Use this method to fetch all leave requests based on the provided filter criteria.
  - It sends a POST request to the /leave/getAllLeaveRequests endpoint with the filter criteria and the authorization token prefixed with 'Bearer' stored in localStorage.

WfhService (Services/wfh.service.ts)
Create a service named wfh inside the app/services folder and implement the following functions in it.

Methods Overview:
- **getWfhRequestById(id: string): Observable<any>**:
  - This method retrieves a specific work-from-home (WFH) request by its ID.
  - It sends a GET request to the /wfh/getWfhRequestById/${id} endpoint with the WFH request ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **addWfhRequest(request: any)**: Observable<any>:
  - Use this method to add a new WFH request.
  - It sends a POST request to the /wfh/addWfhRequest endpoint with the WFH request data provided as the body and the authorization token prefixed with 'Bearer' stored in localStorage.

- **updateWfhRequest(id: string, request: any)**: Observable<any>:
  - Call this method to update an existing WFH request.
  - It sends a PUT request to the /wfh/updateWfhRequest/${id} endpoint with the WFH request ID, updated WFH request data provided as the body, and the authorization token prefixed with 'Bearer' stored in localStorage.

- **getWfhRequestsByUserId(userId: string)**: Observable<any[]>:
  - This method retrieves all WFH requests submitted by a specific user.
  - It sends a GET request to the /wfh/getWfhRequestsByUserId/${userId} endpoint with the user ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **deleteWfhRequest(wfhId: string)**: Observable<any>:
  - This method deletes a WFH request with the specified ID.
  - It sends a DELETE request to the /wfh/deleteWfhRequest/${wfhId} endpoint with the WFH request ID and the authorization token prefixed with 'Bearer' stored in localStorage.

- **getAllWfhRequests(reqObj: any)**: Observable<any>:
  - Use this method to fetch all WFH requests based on the provided filter criteria.
  - It sends a POST request to the /wfh/getAllWfhRequests endpoint with the filter criteria and the authorization token prefixed with 'Bearer' stored in localStorage.

Validations:
Client-Side Validation:
- Implement client-side validation using HTML5 attributes and JavaScript to validate user input before making API requests.
- Provide immediate feedback to users for invalid input, such as displaying error messages near the input fields.

Server-Side Validation:
- Implement server-side validation in the controllers to ensure data integrity.
- Validate user input and API responses to prevent unexpected or malicious data from affecting the application.
- Return appropriate validation error messages to the user interface for any validation failures.

Exception Handling:

- Implement server-side validation in the controllers to ensure data integrity.
- Validate user input and API responses to prevent unexpected or malicious data from affecting the application.
- Return appropriate validation error messages to the user interface for any validation failures.

Error Pages:

Create custom error pages for different HTTP status codes (e.g., 404 Not Found, 500 Internal Server Error) to provide a consistent and user-friendly error experience. Ensure that error pages contain helpful information and guidance for users.

Thus, create a reliable and user-friendly web application that not only meets user expectations but also provides a robust and secure experience, even when faced with unexpected situations. Error page must be displayed if something goes wrong.

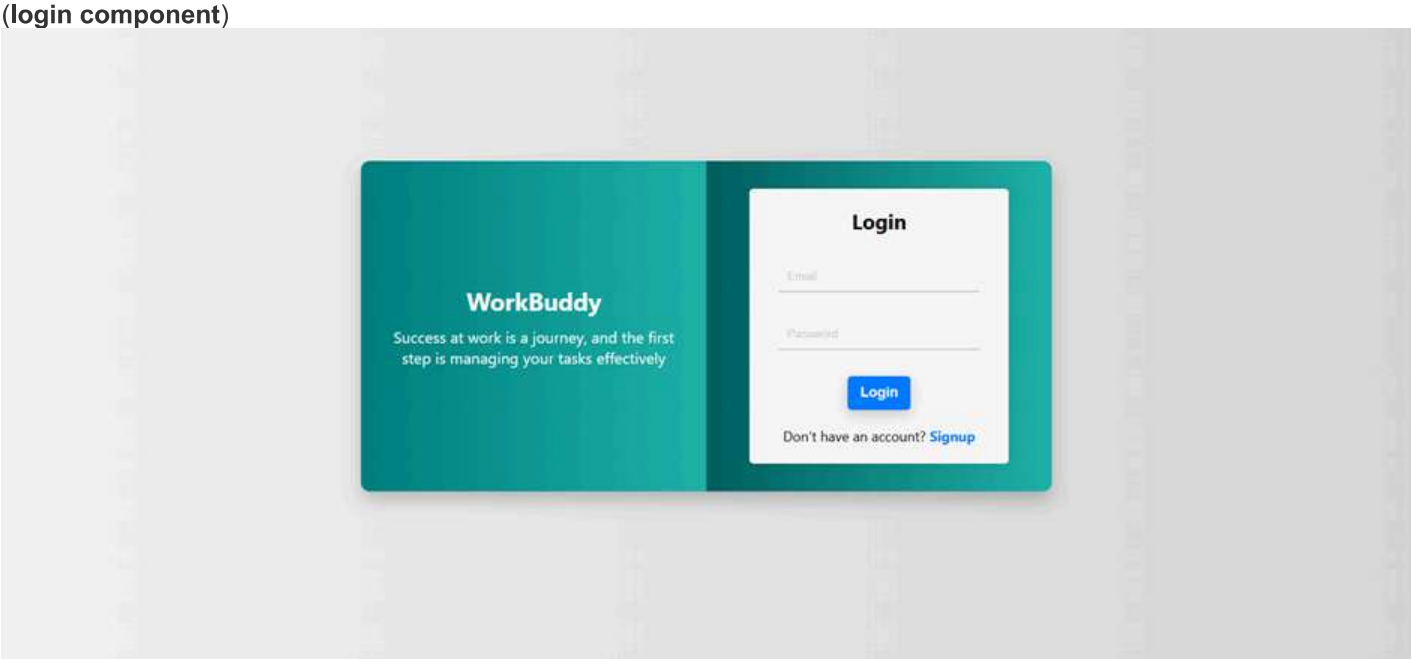
Frontend Screenshots:

• All the asterisk (*) marked fields are mandatory in the form. Make sure to mark all the field names with * symbol followed by the validations.

Navigation Bar:
(employeeNavbar component)

(managerNavbar component)

Landing Page:
(**login component**)



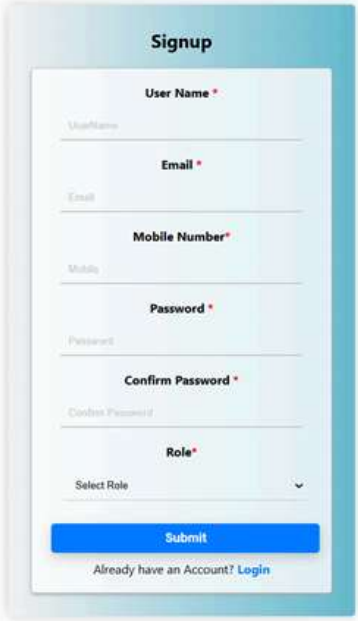
Component features a heading "**WorkBuddy**" accompanied by an introductory message that provides an overview of the application with login form**.**

Common Screens (Employee and Manager)

Signup Page:

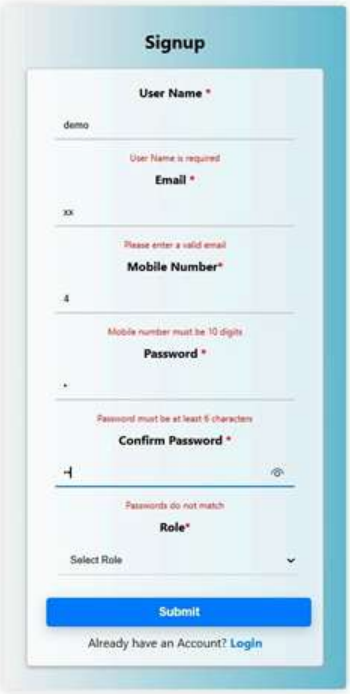
(**signup component**)

Clicking "Signup" in the login page will navigate to the registration page for both roles. Please refer to the screenshots below for validation.
Use heading as "Signup".

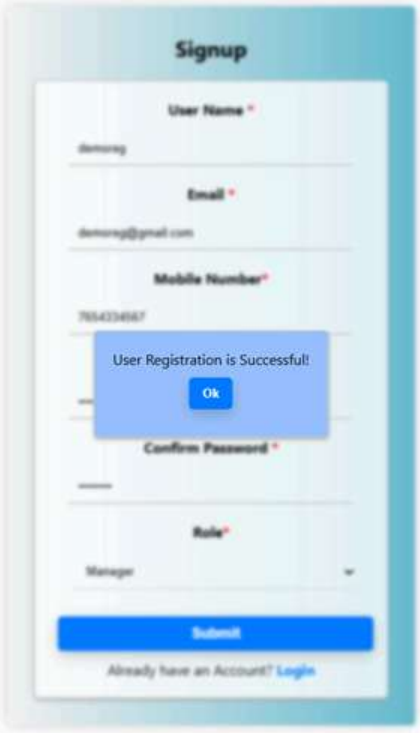


If the value of all input fields is an empty string, dispatches an input event for each input field to simulate user interaction.
Onclicking the “Submit” button with an empty field will display the error message.
Implement placeholder as **UserName**, **Email**, **Mobile**, **Password,** and **Confirm Password** for respective fields and labels as **User Name**, **Email**, **Mobile Number**, **Password** and **Confirm Password**.
Clicking the Submit button with empty fields should display the error messages "User Name is required", "Email is required", "Mobile Number is required", "Password is required", and "Confirm Password is required".

Regular expressions (Regex) are also included to validate email formats, mobile numbers, and to ensure password matching.



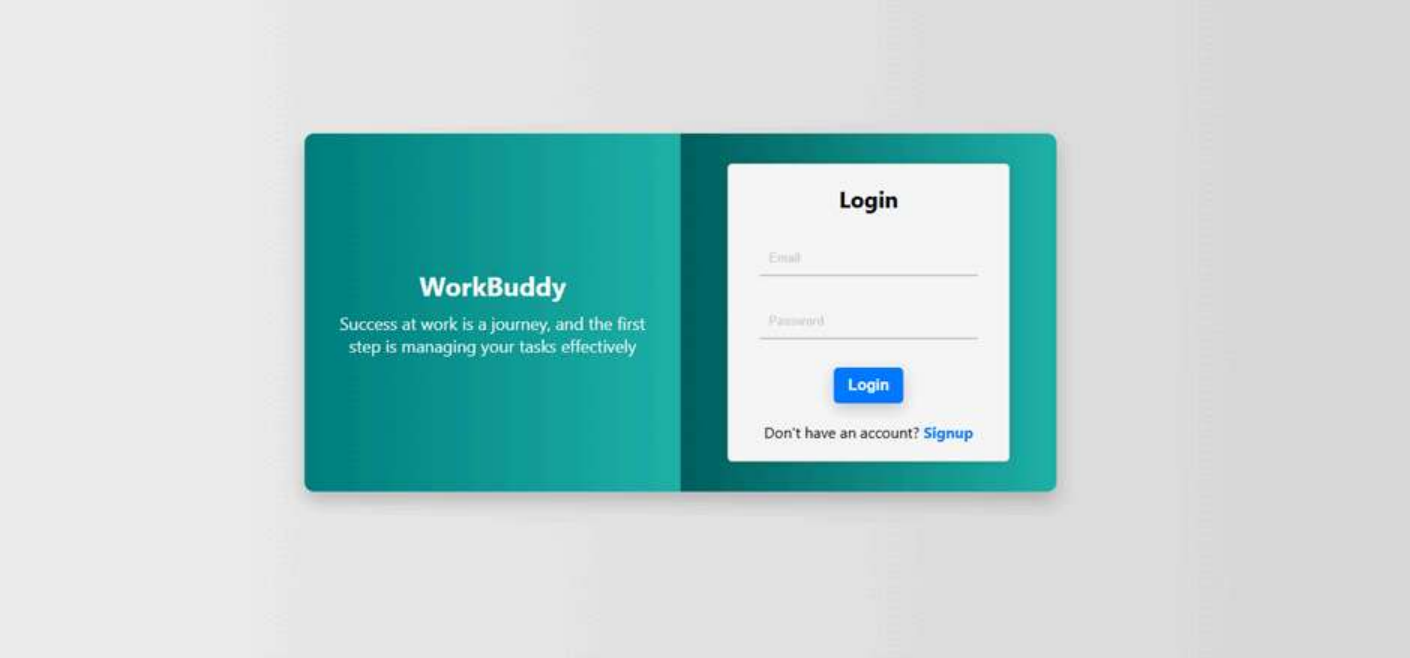
When the "Submit" button is clicked, upon successful submission, the user must be navigated to the login page.



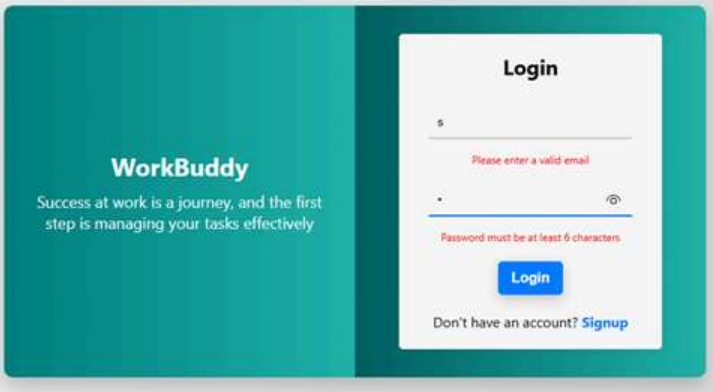
Upon successful completion, a modal will appear, prompting users to click the "Ok" button. Upon clicking this button, users will be directed to the login page, signifying a successful signup process

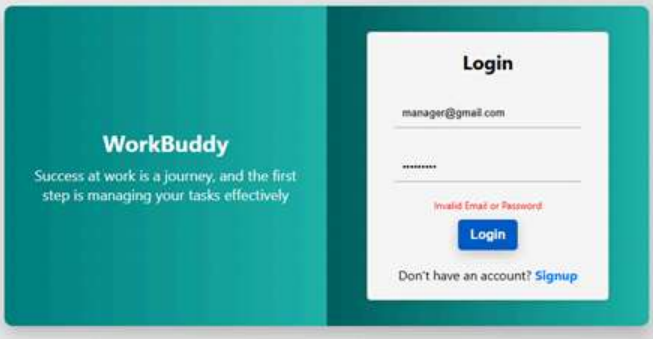
Login Page
(login component)

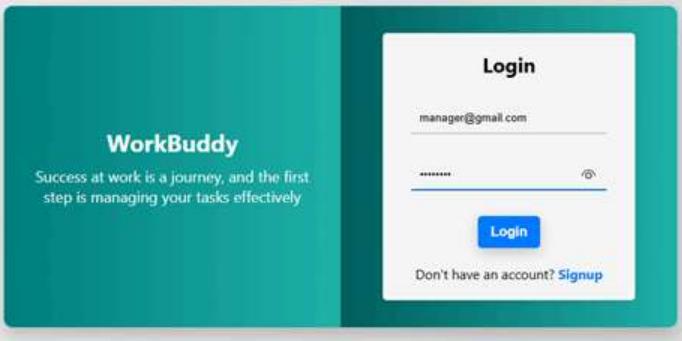
This page is used for logging in to the application. On providing the valid email and password, the user will be logged in. Display the heading as "**Login**".



Perform validations for email and password fields.
Use the placeholder as **Email** and **Password** and input type as **email** and **password** for respective fields.
If the value of both the email and password input fields is an empty string, by clicking the Login button, then dispatch an input event on both the email and password input fields to simulate user interaction and display the error messages.
The error messages are "**Email is required**" and "**Password is required**".





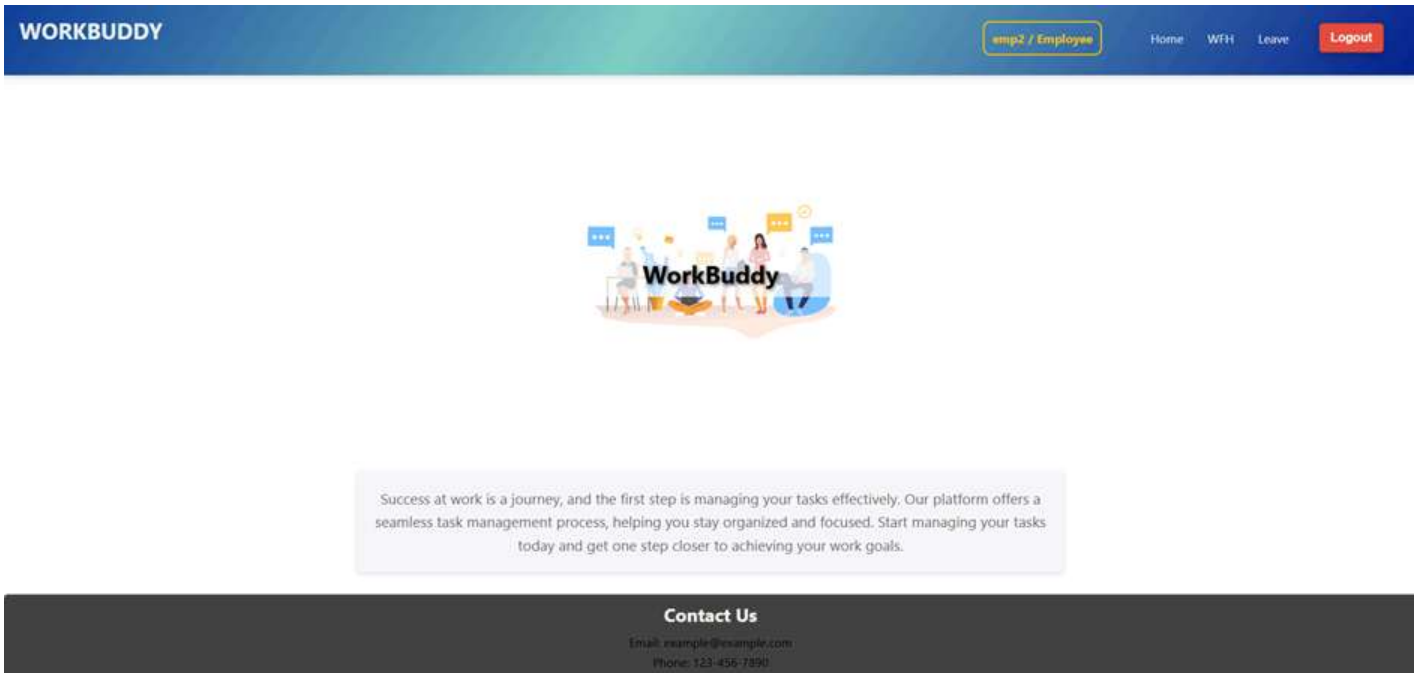


On Clicking the “Login” button, the user will be navigated to the (EmployeeNavbar or ManagerNavbar component) based on their roles.

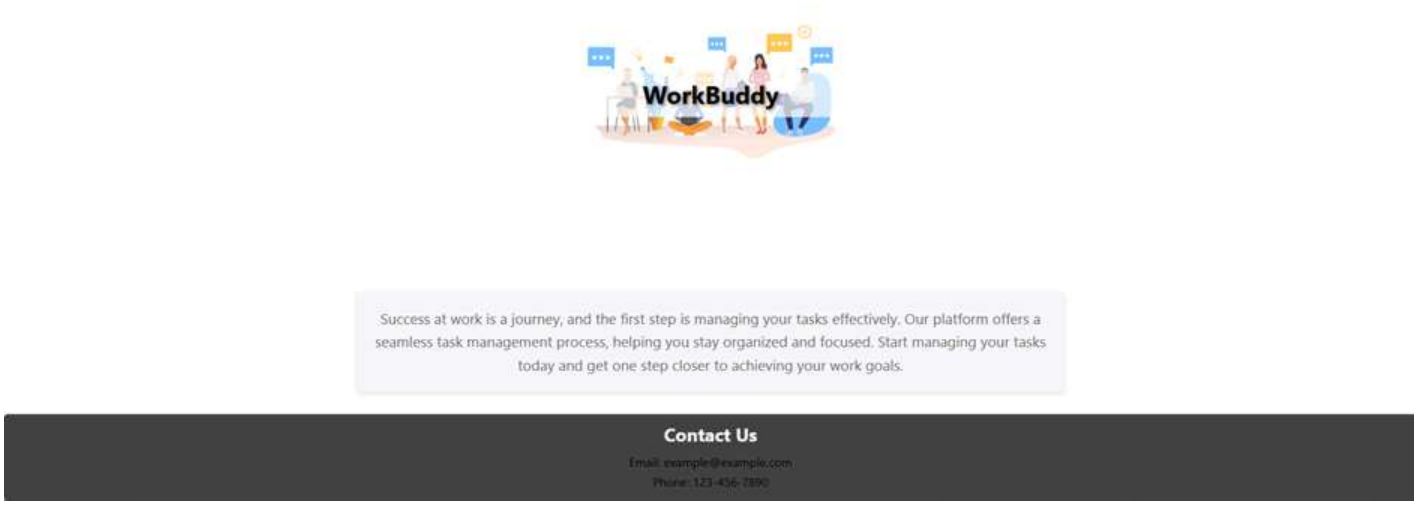
Employee Side:

employeeNavbar component:

Upon successful login, if the user is an employee, the (employeeNavbar component) will be displayed. If the user is a Manager, the (managerNavbar component) will be displayed. Additionally, the role-based navigation bar will also display login information such as the username and role.

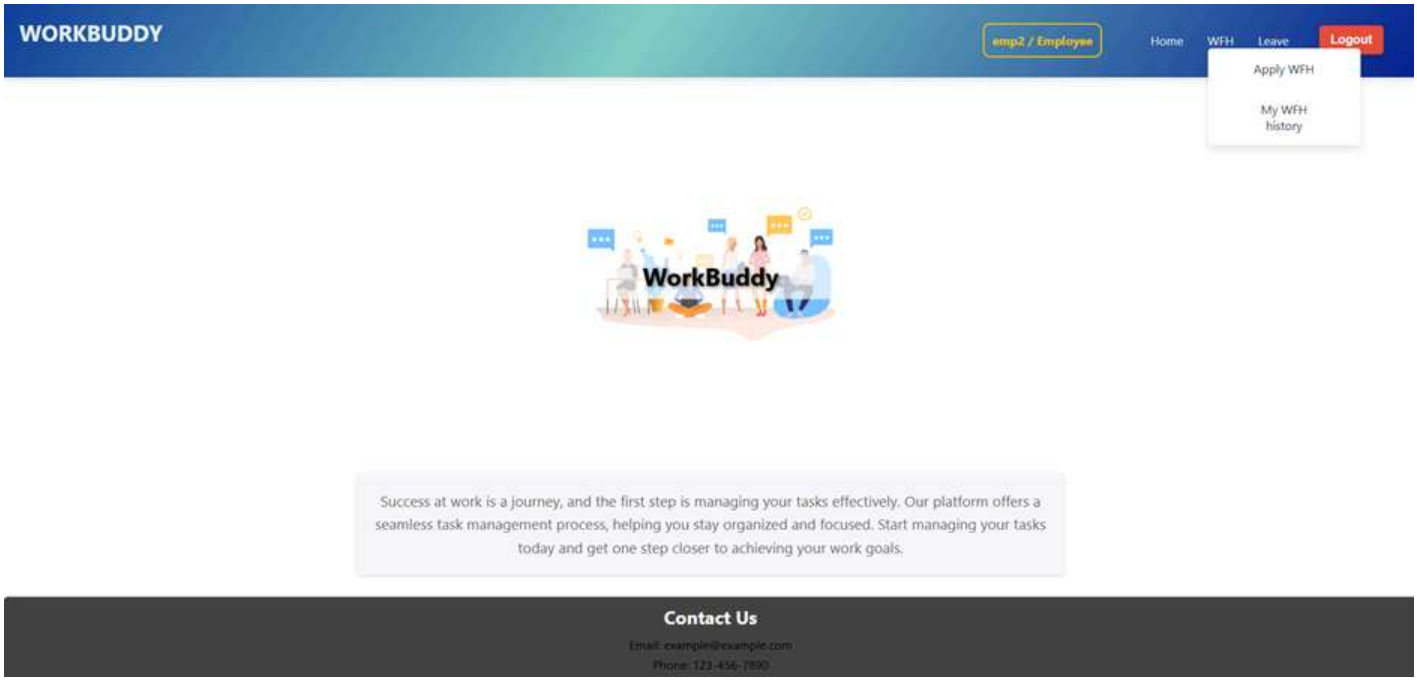


homePage Component: This page is used to display the information about workbuddy application. On clicking the 'Home' tab, user can view the information about the application. (workbuddycoverimage.png provided in assets/images folder)
Display the heading as "WorkBuddy", a paragraph starting with "Success at work is a journey" and use the "Contact Us" section.

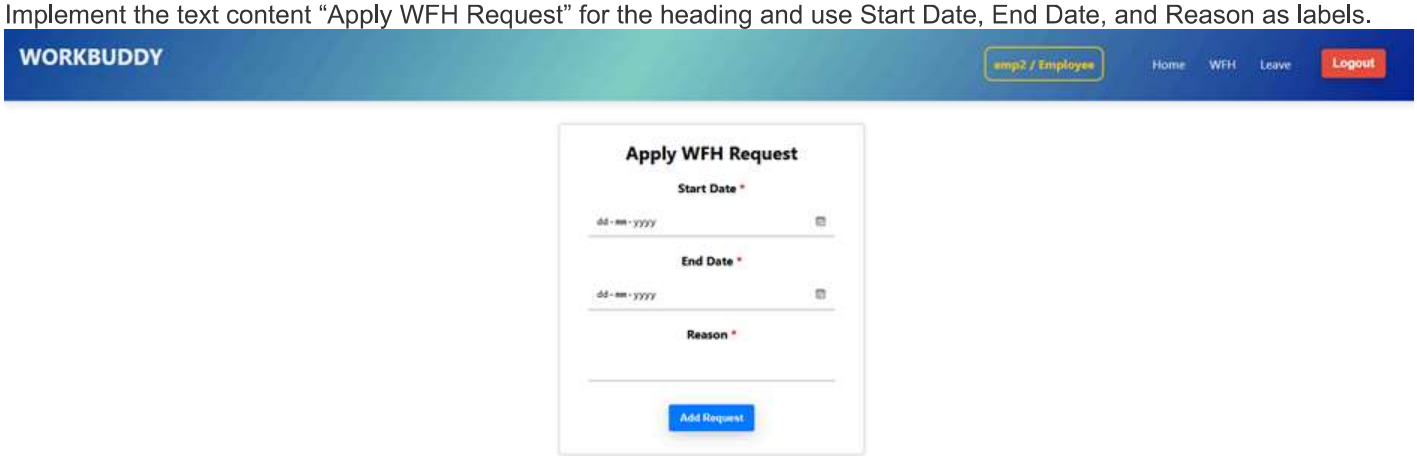


Employee can navigate to other pages by clicking on the menu available in the navigation bar.

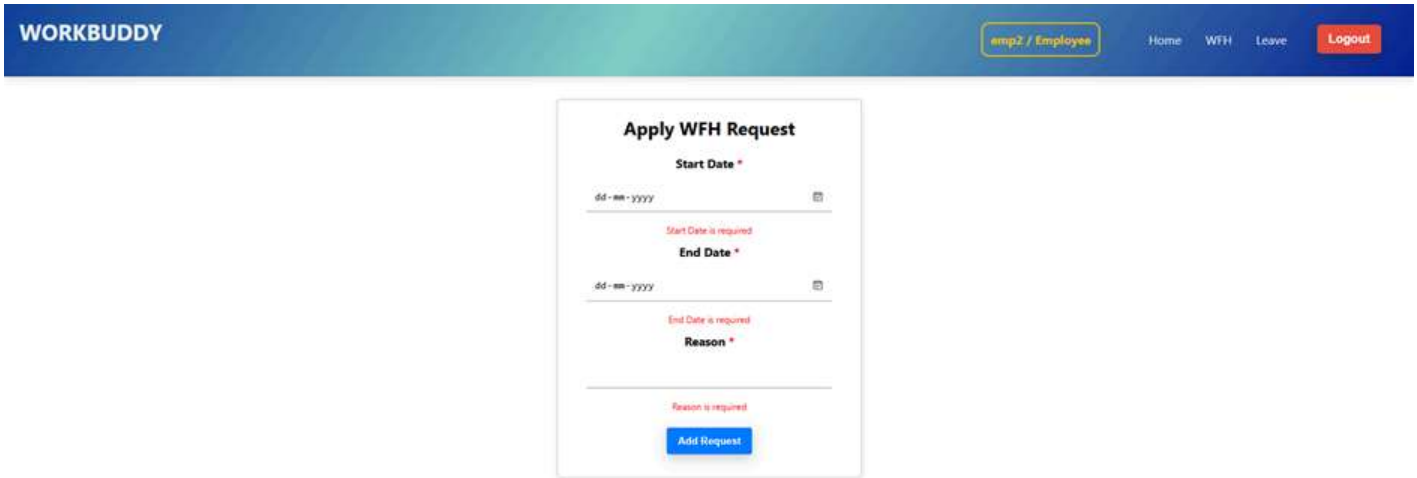
Apply WFH:
Clicking the "WFH" option in the navbar will reveal "Apply WFH" and "My WFH history" options, allowing the employee to apply new wfh request and view existing ones.

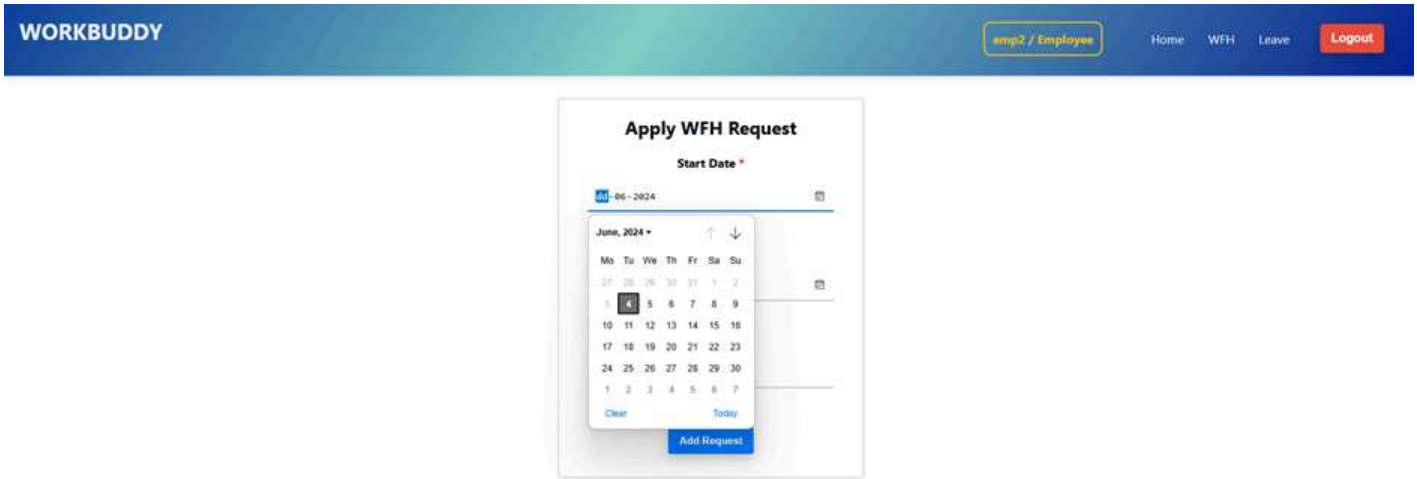


By clicking the “Apply WFH” option will navigate to the “wfhForm” component, where the employee can apply wfh. Implement the text content “Apply WFH Request” for the heading and use Start Date, End Date, and Reason as labels.

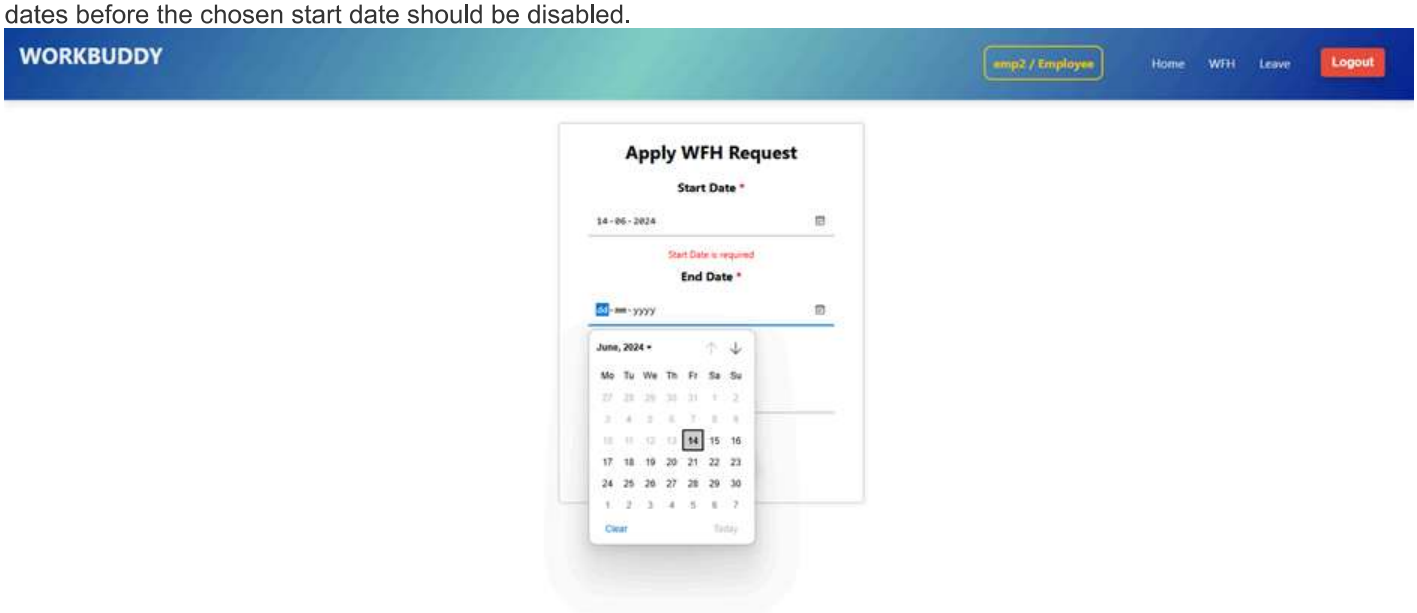


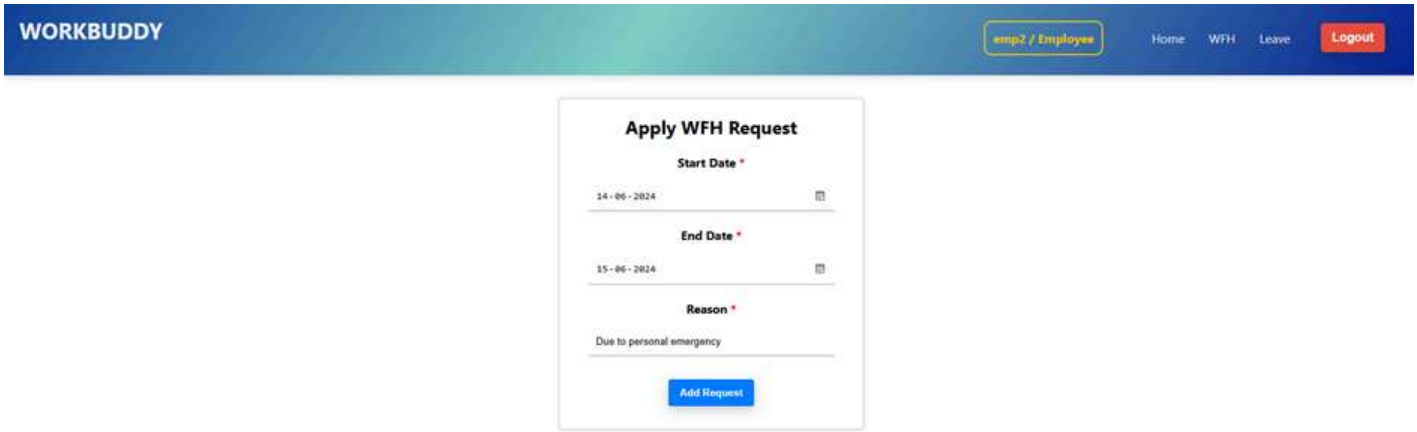
On Clicking the “**Add Request**” button with an empty field will display a validation error message "**Start Date is required**", "**End Date is required**" and "**Reason is required**".



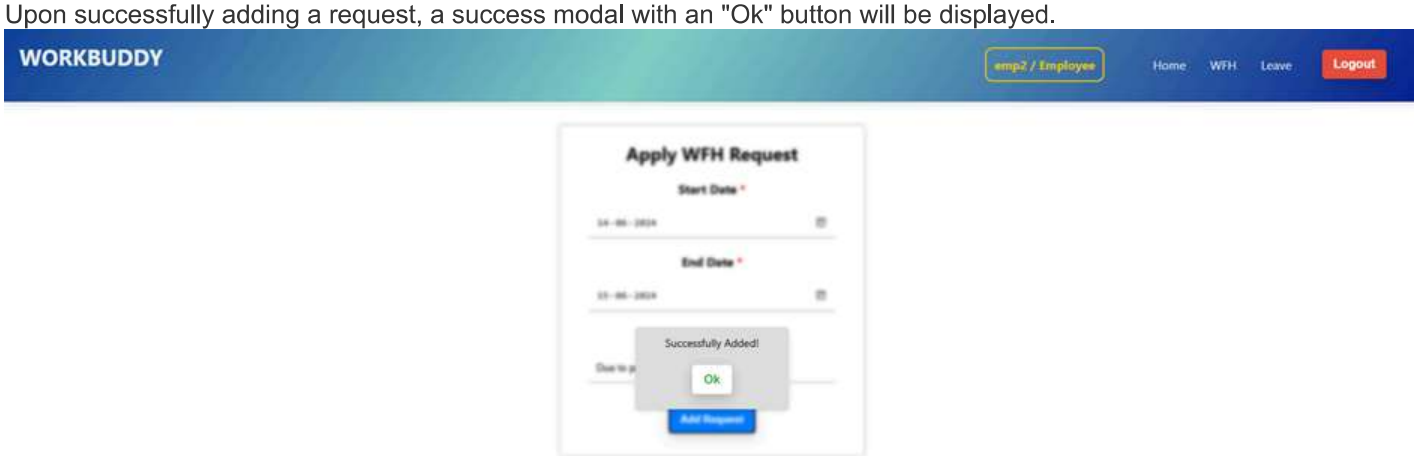


When selecting a start date, dates before the current date should be disabled. Similarly, when selecting an end date, dates before the chosen start date should be disabled.

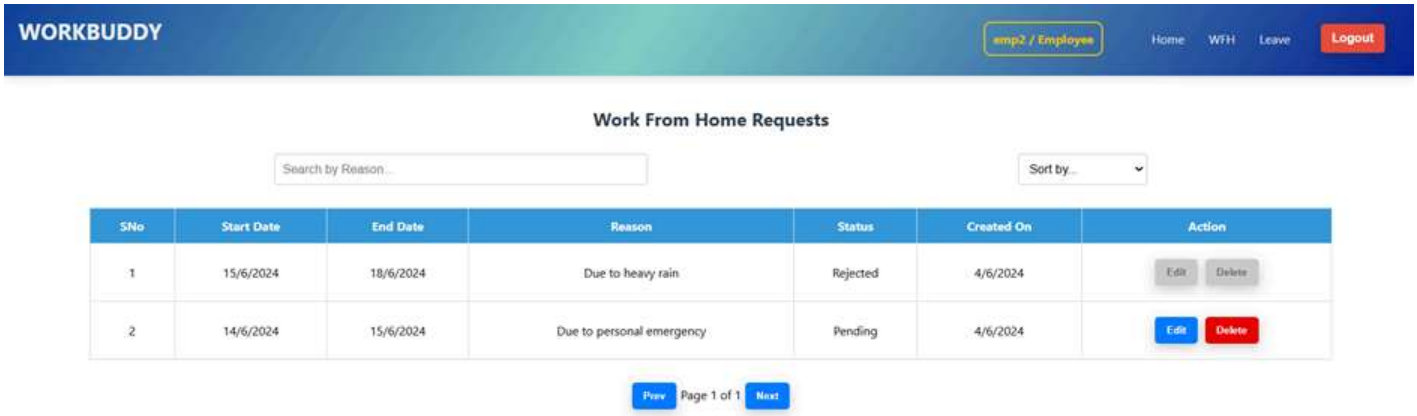




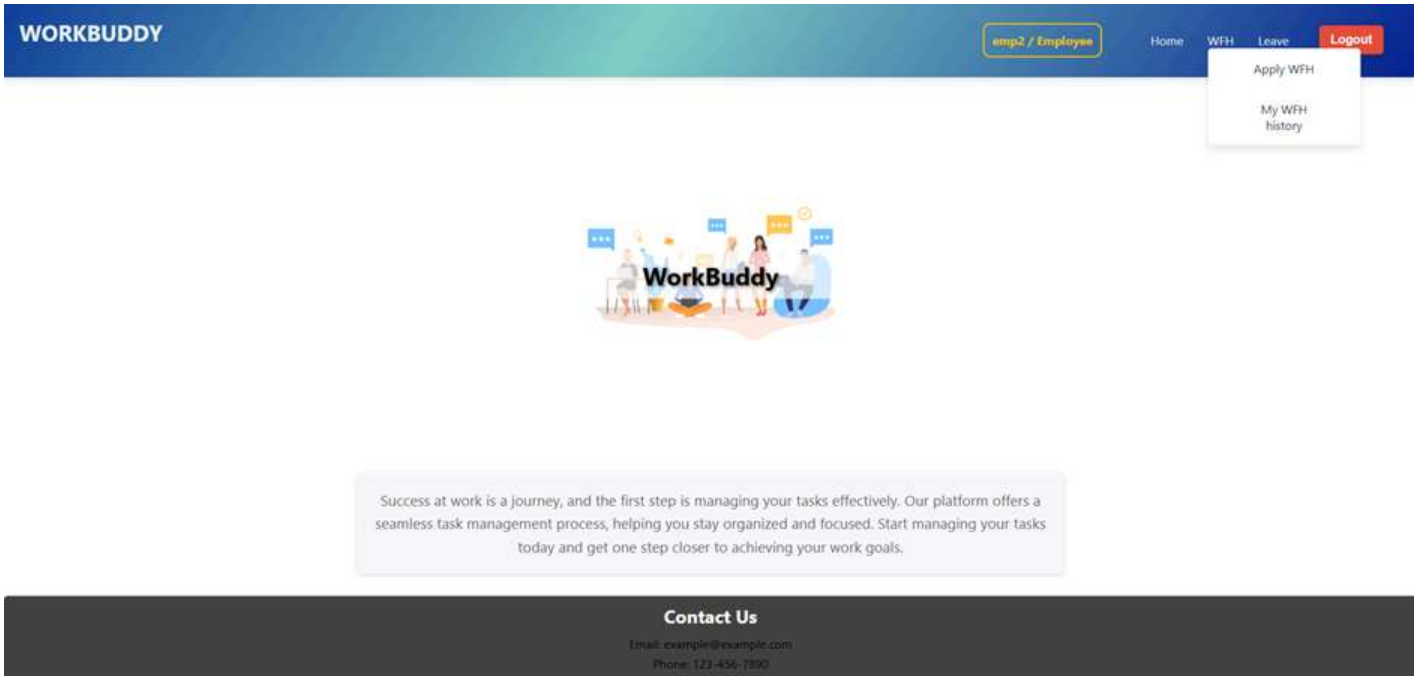
Upon successfully adding a request, a success modal with an "Ok" button will be displayed.



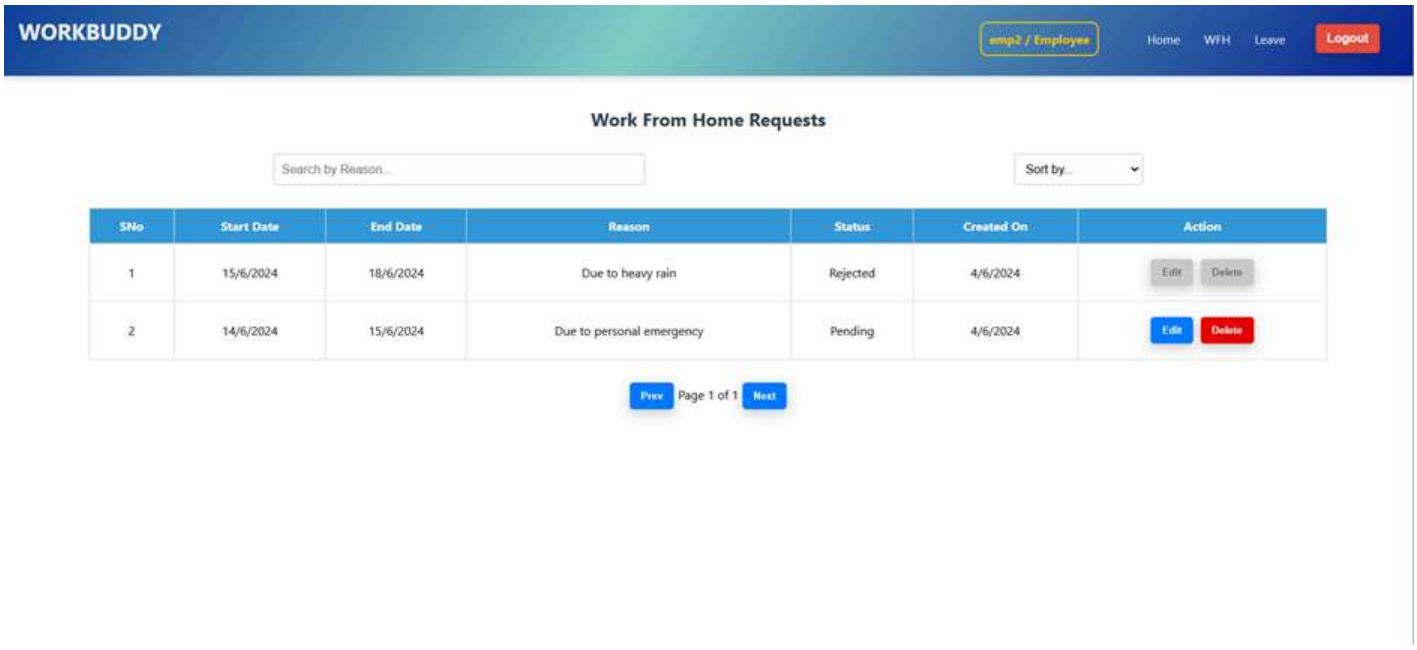
Clicking on “Ok” button will navigate to “viewWfh” component. If no records are available in the Work From Home Requests, it should display the “**Oops! No records Found**”.
Implement the text content “**Work From Home Requests**” for the heading and appropriate headers "SNo", "Start Date", "End Date", "Reason", "Status", "Created On" and "Action" for the table as shown in the viewWfh component image should be presented.



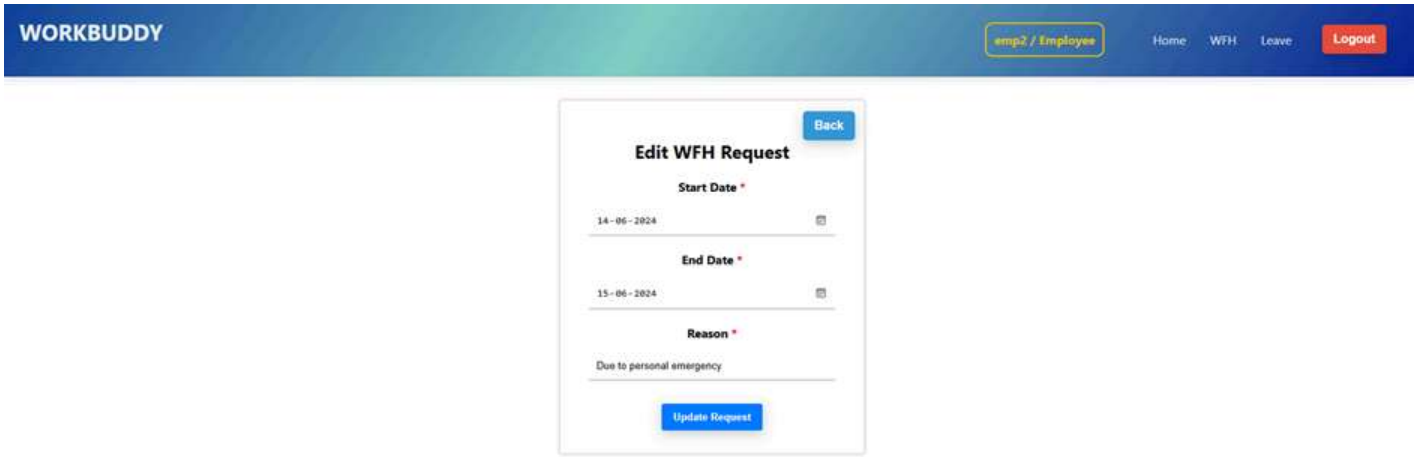
My WFH history
Clicking the “WFH” option in the navbar will reveal the “Apply WFH” and “My WFH history” options, allowing the employee to apply new wfh request and view existing ones.



On Clicking the “My WFH history” option under request it navigates to the list of requests.
“viewWfh Component” contains feature like searching based on reason , filter by startdate and pagination.

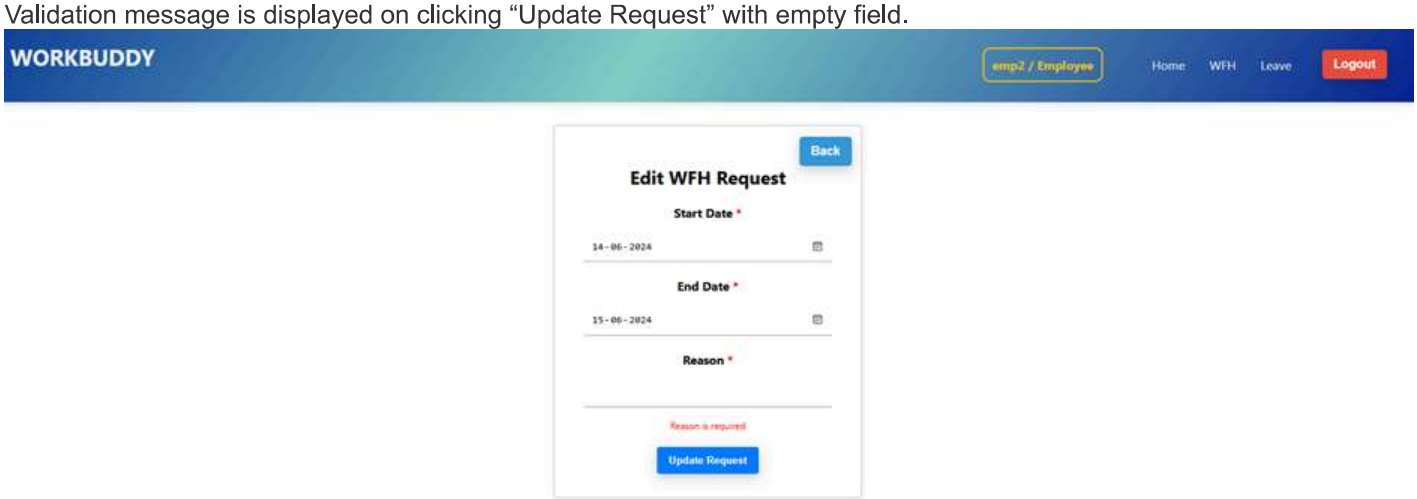


Clicking the "Edit" button will navigate to the wfhForm component, displaying the edit form prepopulated with the data of the selected wfh request.



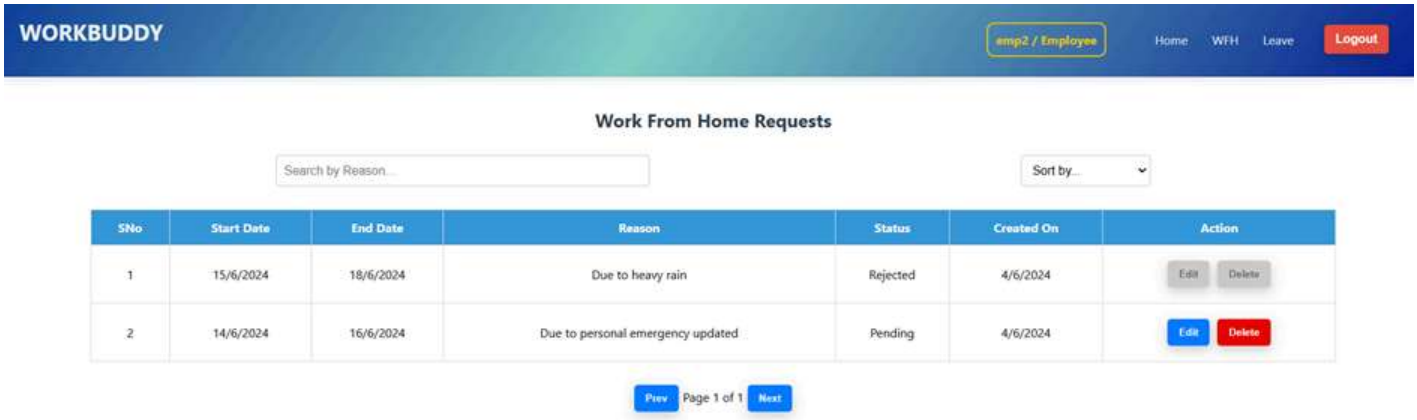
Form includes “Back” button, will navigate to previous page.

Validation message is displayed on clicking “Update Request” with empty field.



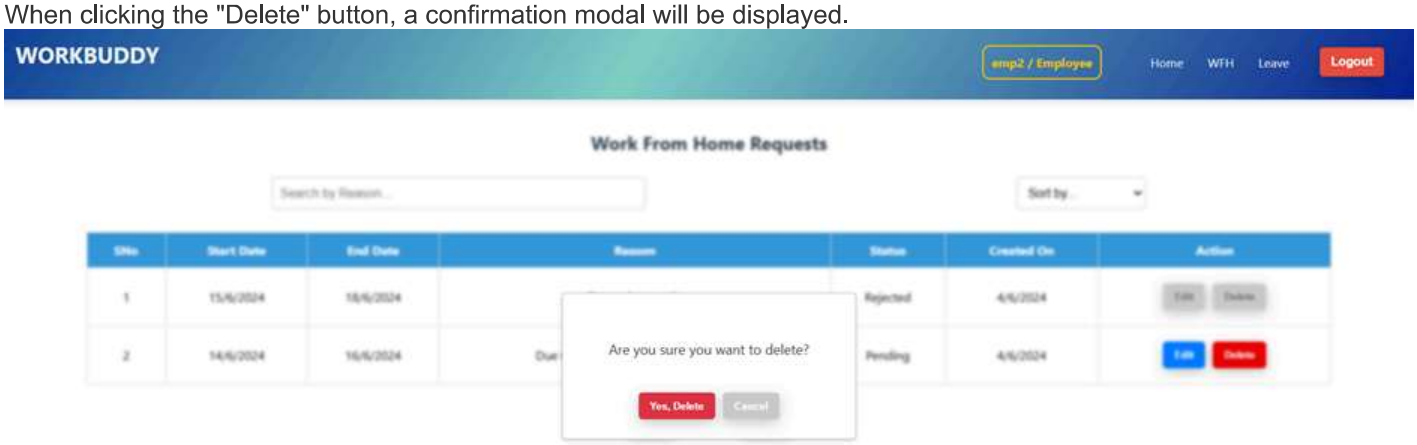
Upon successfully updating a request, a success modal with an "Ok" button will be displayed.

Clicking on “Ok” button will navigate to “viewWfh” component.



The Edit and Delete buttons can only be accessed if the status is in the Pending state.

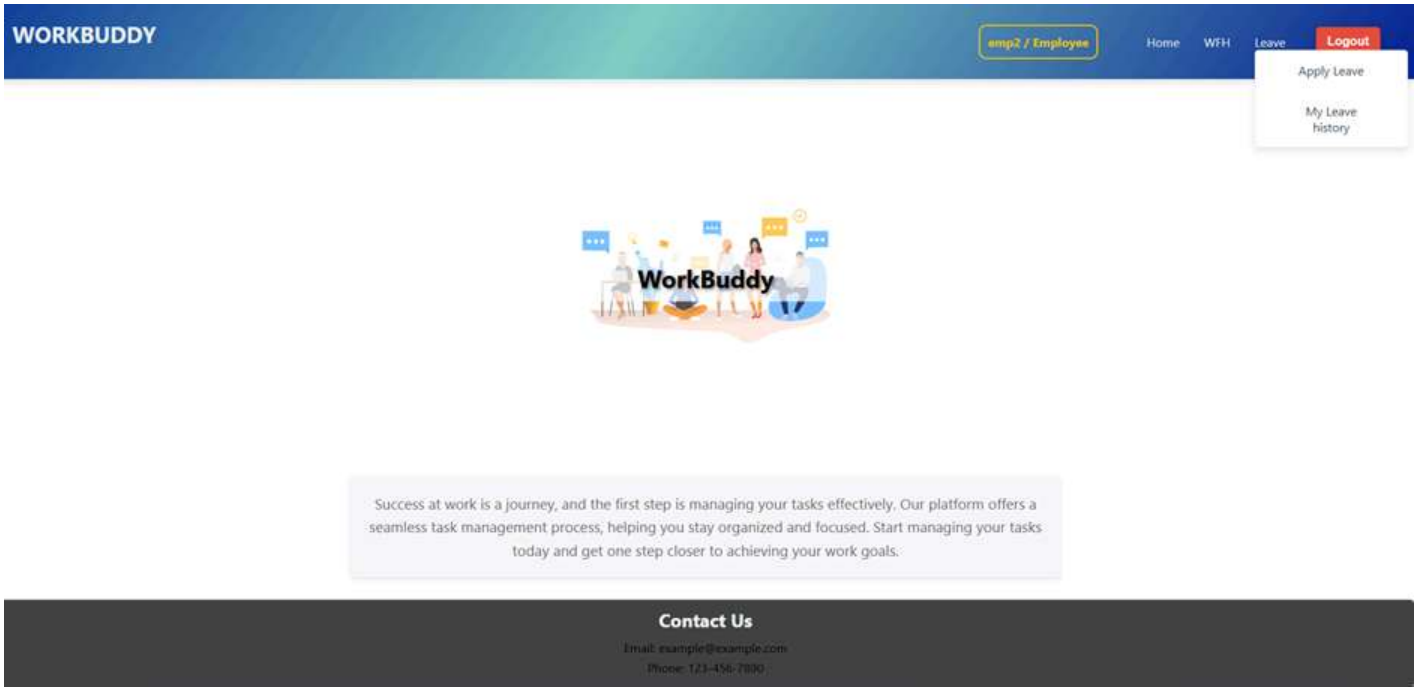
When clicking the "Delete" button, a confirmation modal will be displayed.



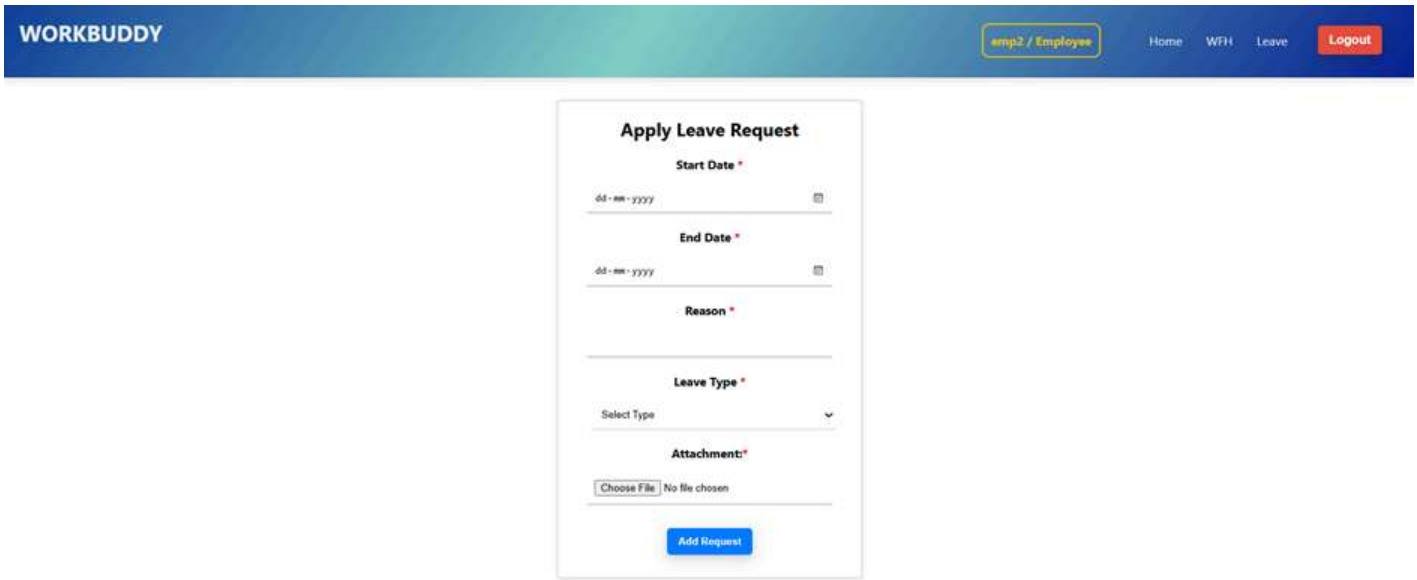
Clicking "Yes, Delete" will remove the selected request and refresh the displayed data.
Clicking “Cancel” will close the popup window.

Apply Leave:

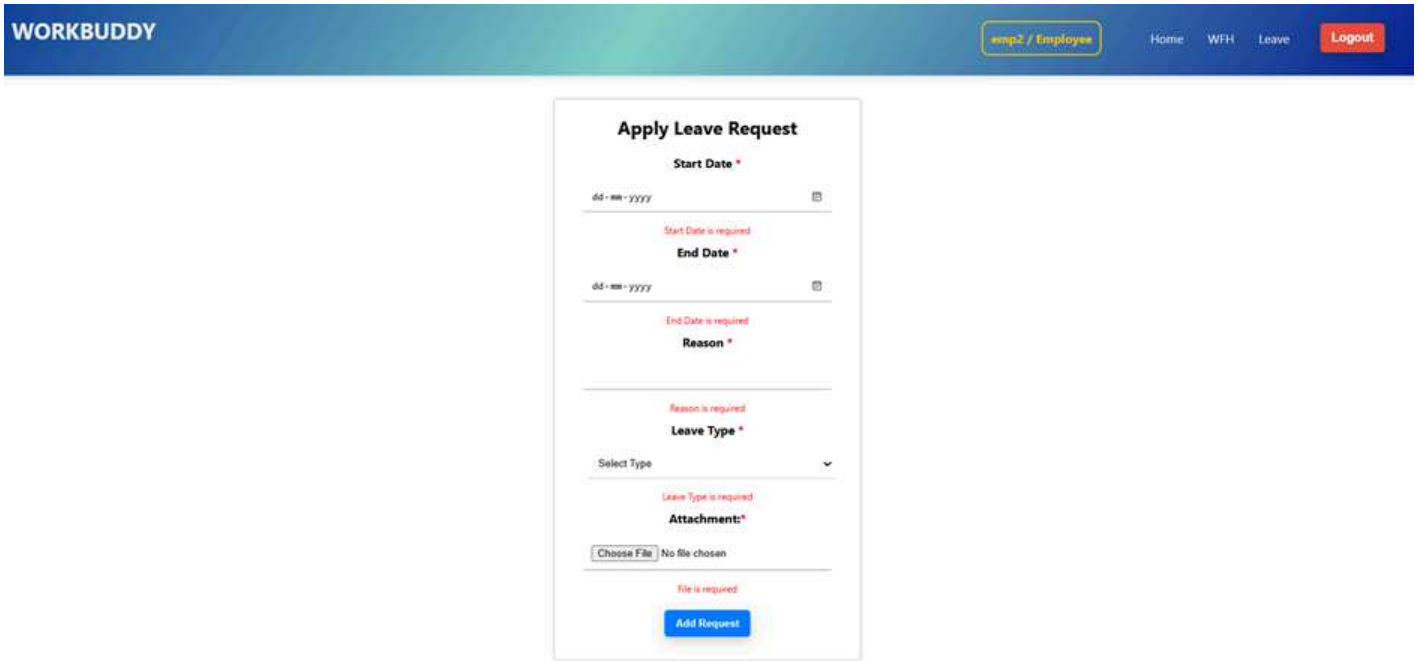
Clicking the “Leave” option in the navbar will reveal “Apply Leave” and “My Leave history” options, allowing the employee to apply new leave request and view existing ones.

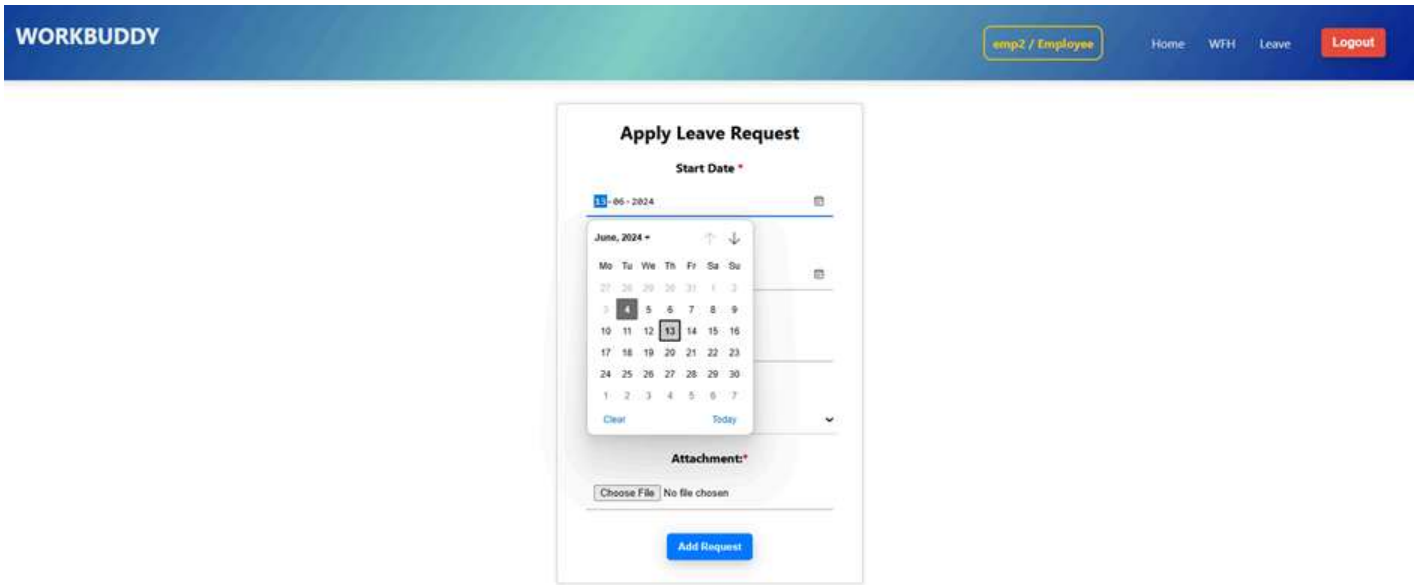


By clicking the “Apply Leave” option will navigate to the “**leaveForm**” component, where the employee can apply leave. Use the text content “Apply Leave Request” for the heading and Start Date, End Date, Reason, Leave Type, and Attachment as labels.

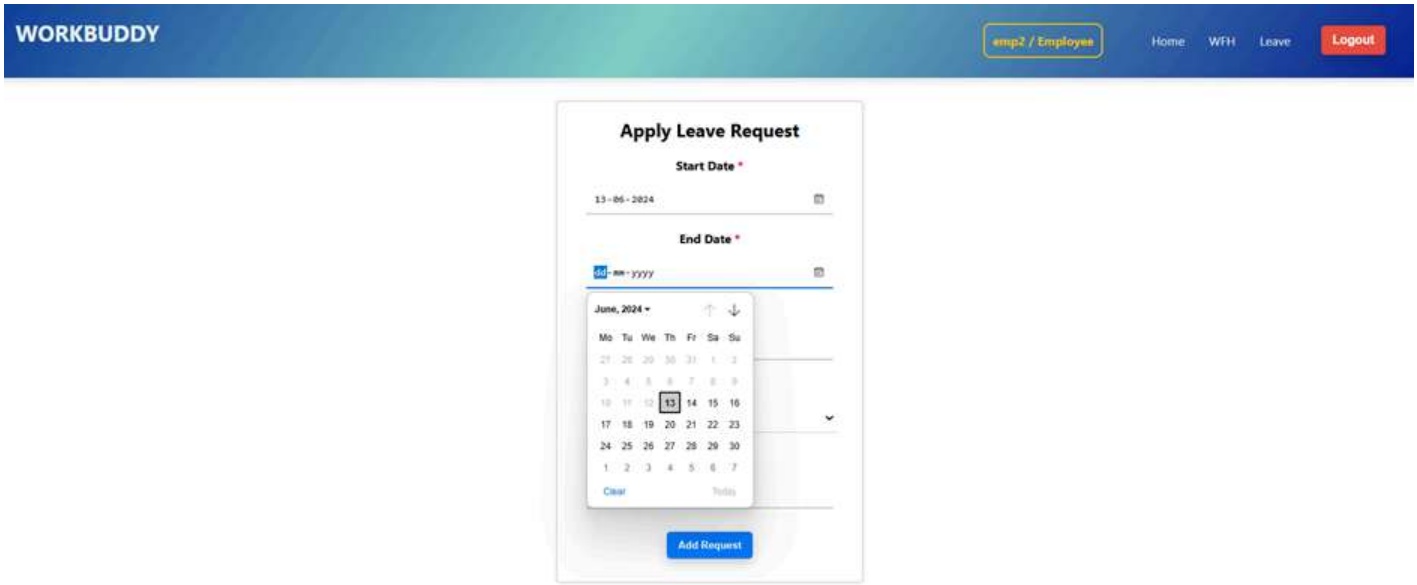


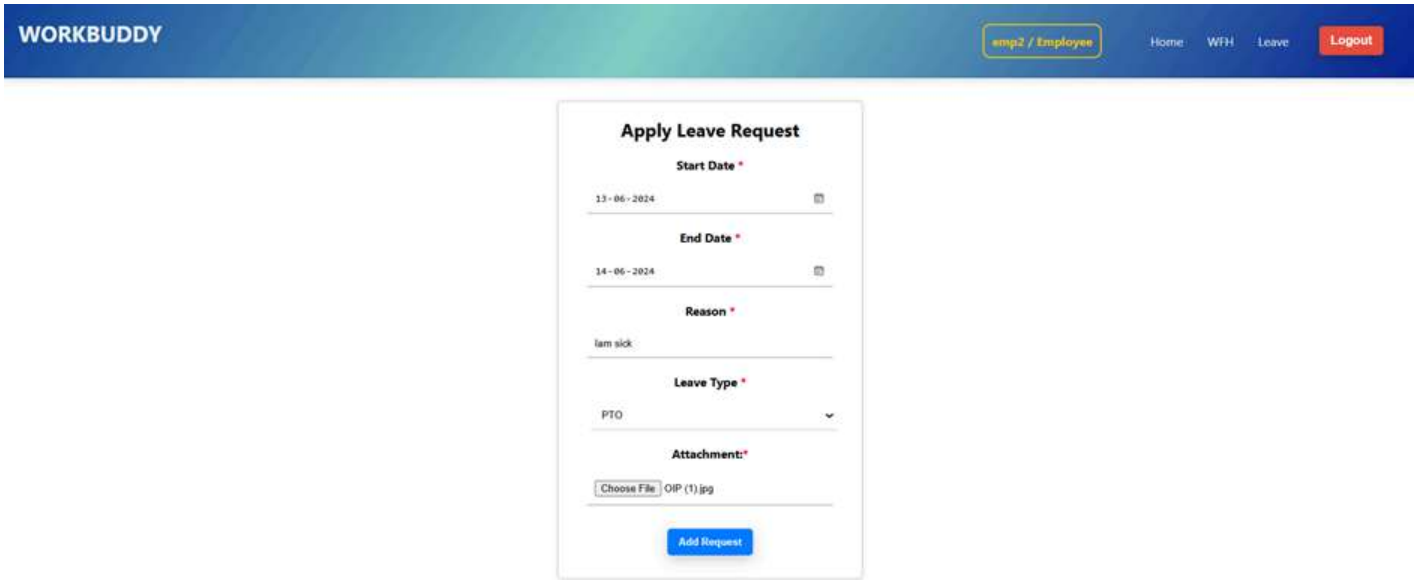
On Clicking the “**Add Request**” button with empty input fields will display a validations error message "Start Date is required", "End Date is required", "Reason is required", "Leave Type is required" and "File is required".





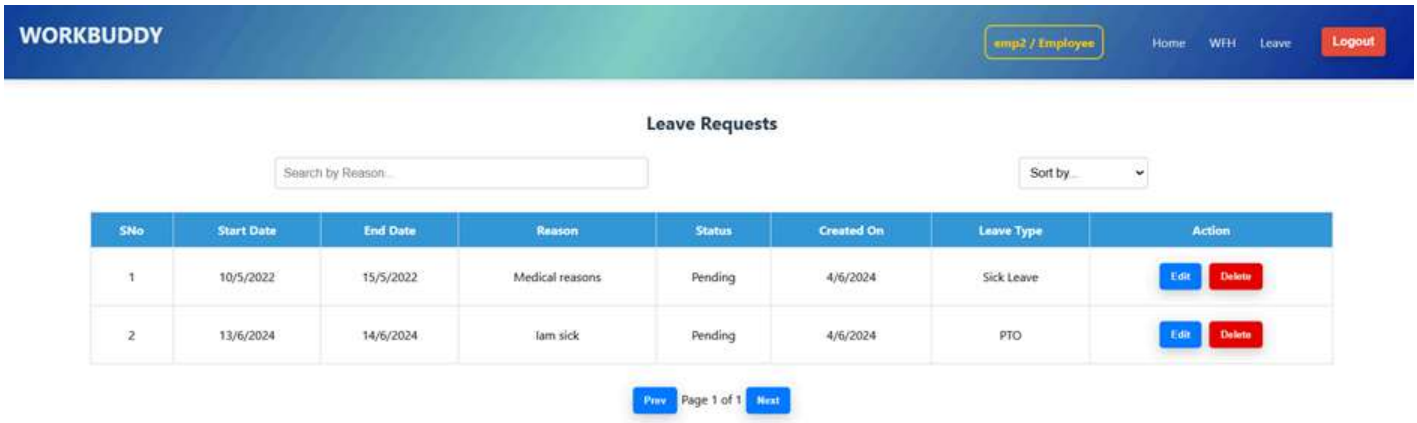
When selecting a start date, dates before the current date should be disabled. Similarly, when selecting an end date, dates before the chosen start date should be disabled.





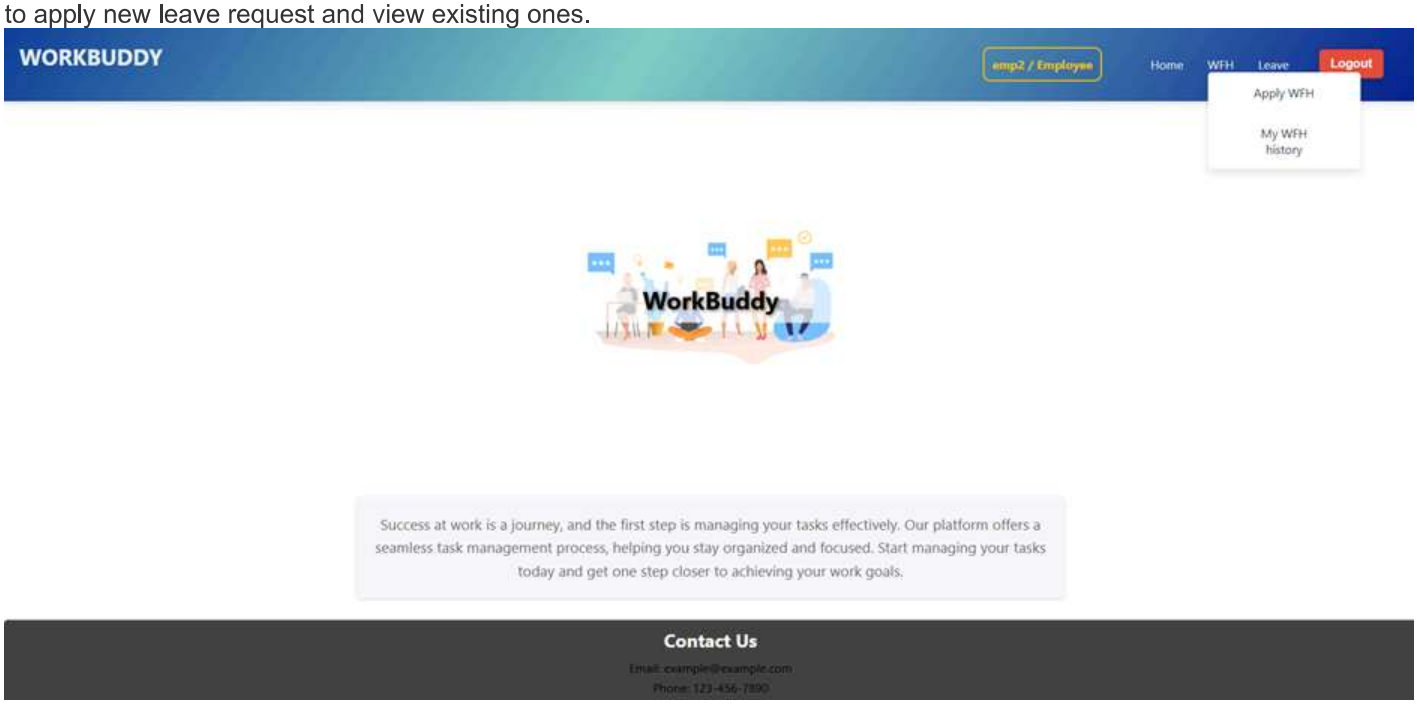
Upon successfully adding a request, a success modal with an "Ok" button will be displayed.

Clicking on “Ok” button will navigate to the “viewLeave” component.
Implement the text content “Leave Requests” for heading, use the appropriate table headers "SNo", "Start Date", "End Date", "Reason", "Status", "Created On", "Leave Type" and "Action" as shown in the below image. Use search input element for searching.



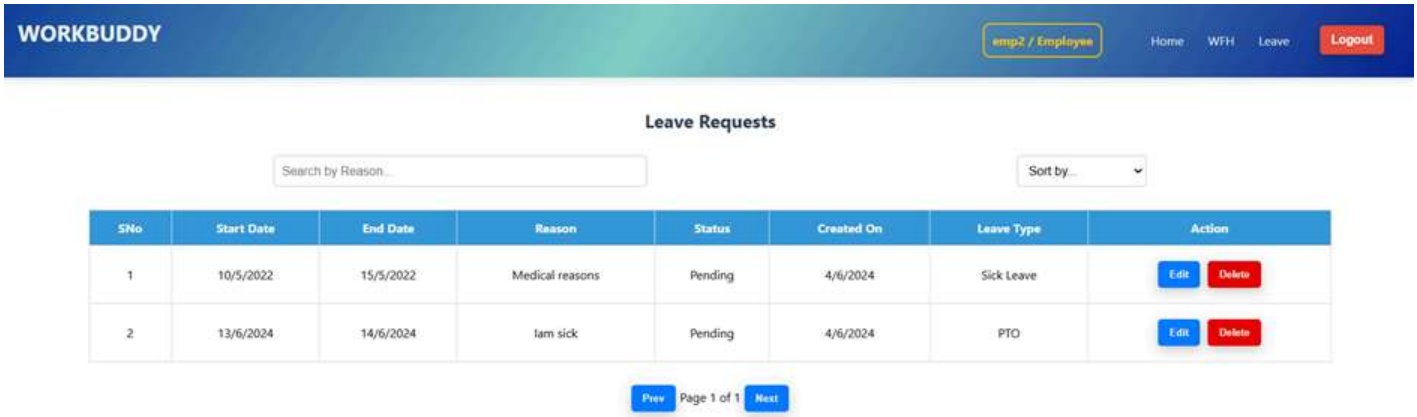
My Leave history

Clicking the “Leave” option in the navbar will reveal “Apply Leave” and “My Leave history” options, allowing the employee to apply new leave request and view existing ones.

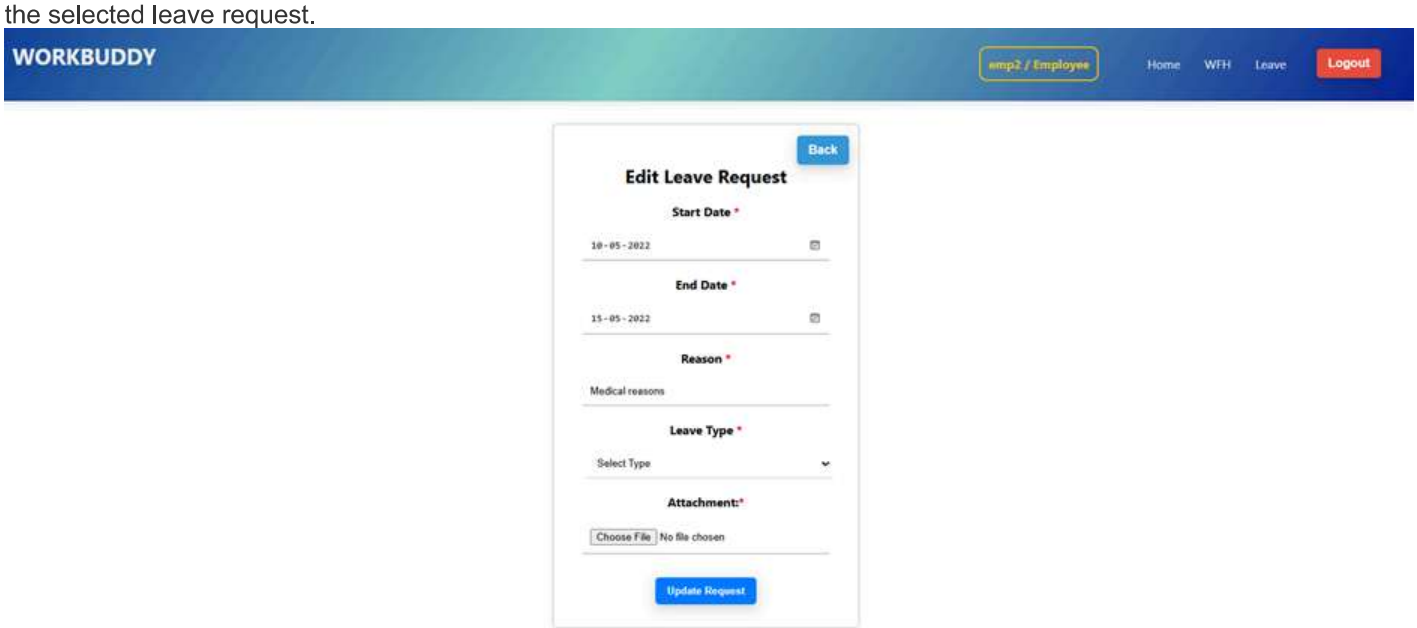


On Clicking the “My Leave history” option under Leave it navigates to the “viewLeave” component.

“viewLeave Component” contains feature like searching based on reason , filter by startdate and pagination.

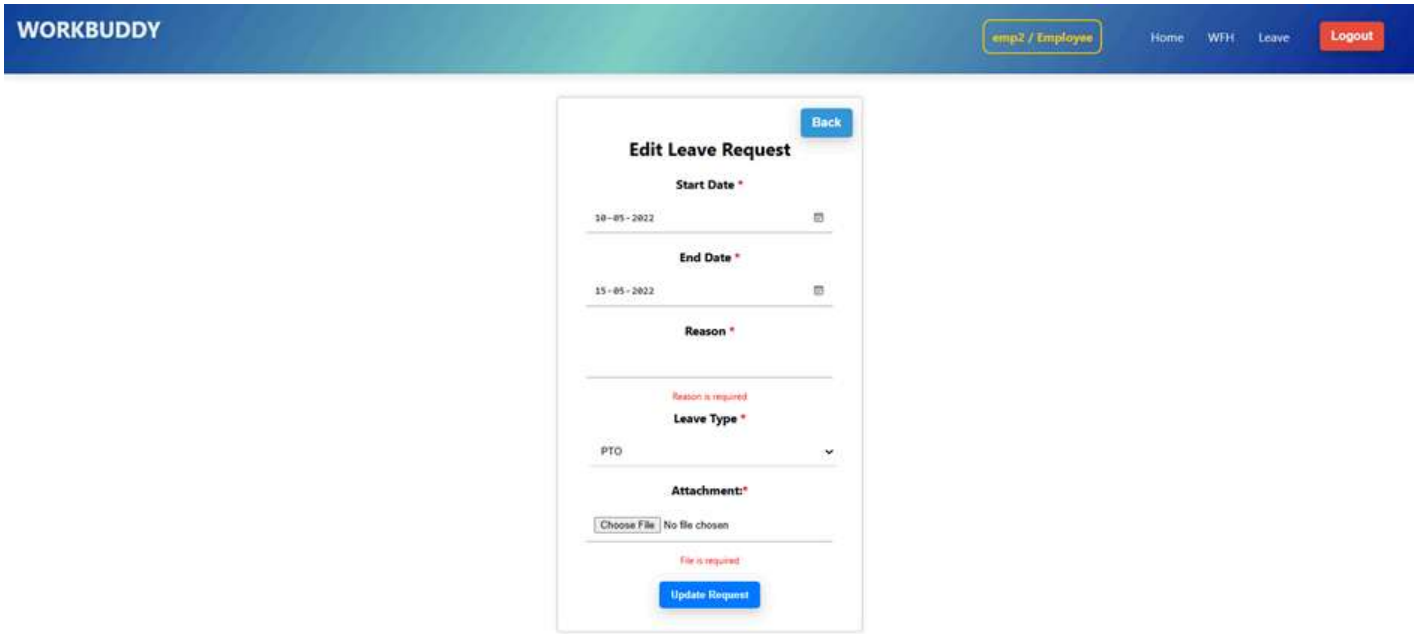


Clicking the "Edit" button will navigate to the leaveForm component, displaying the edit form prepopulated with the data of the selected leave request.

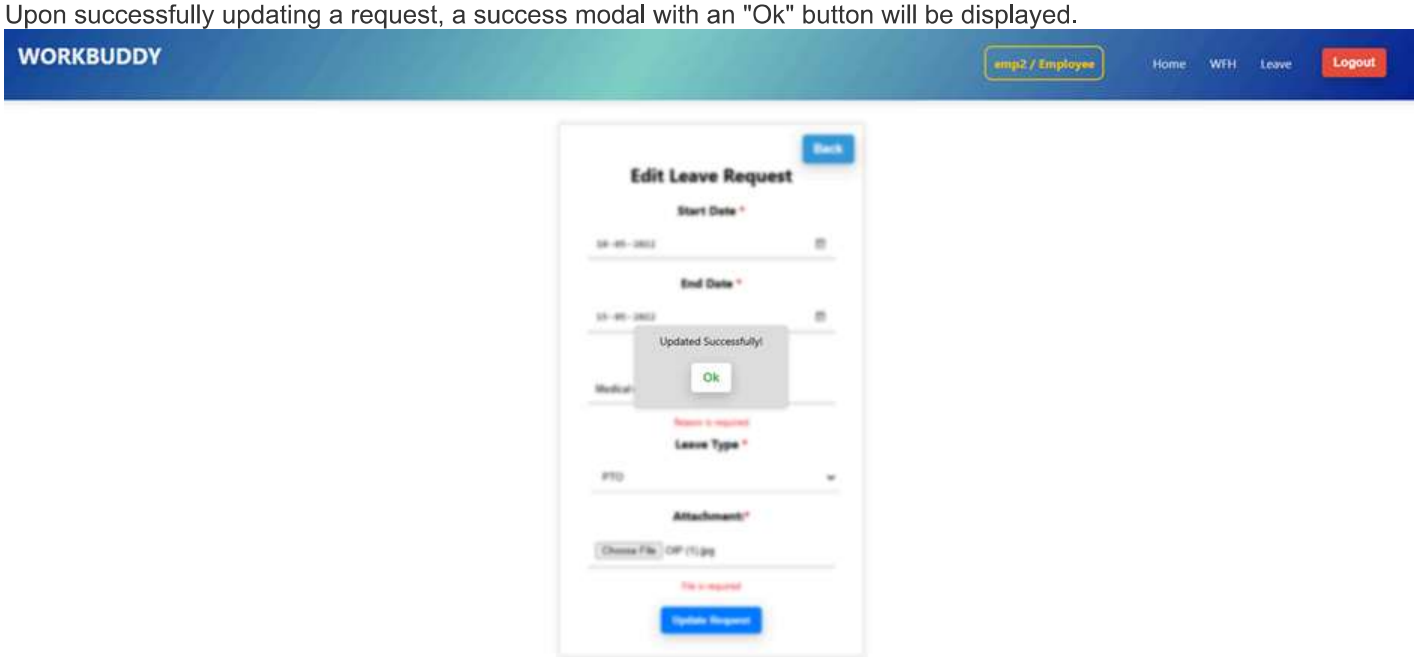


Form includes “Back” button , will navigate to previous page.

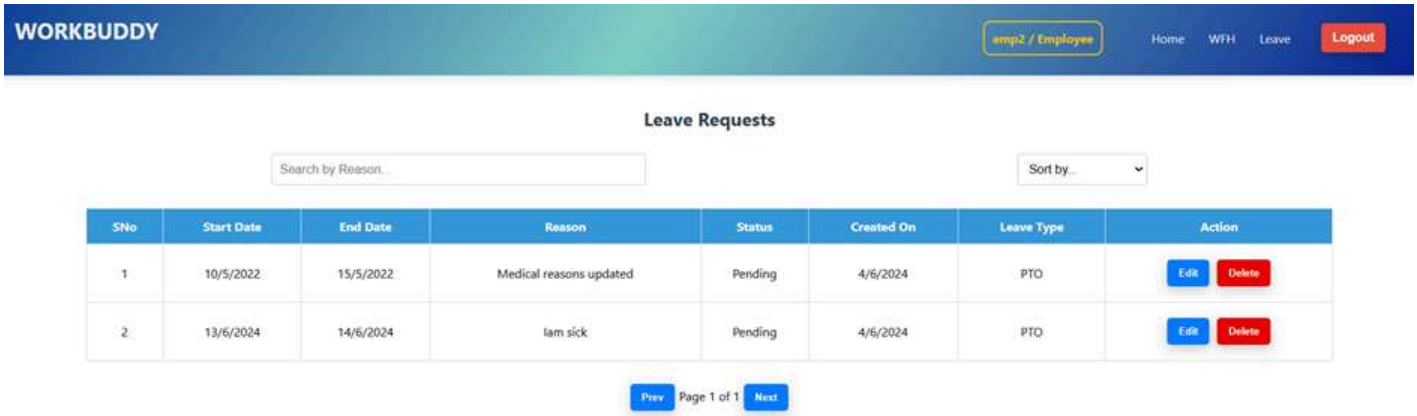
Validation message is displayed on clicking “Update Request” with empty field.



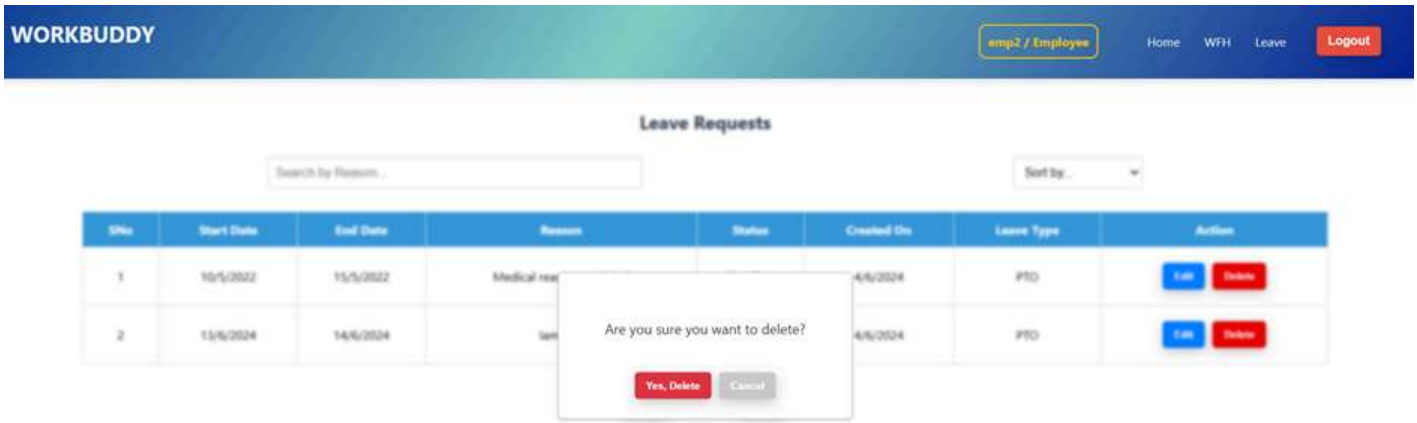
Upon successfully updating a request, a success modal with an "Ok" button will be displayed.



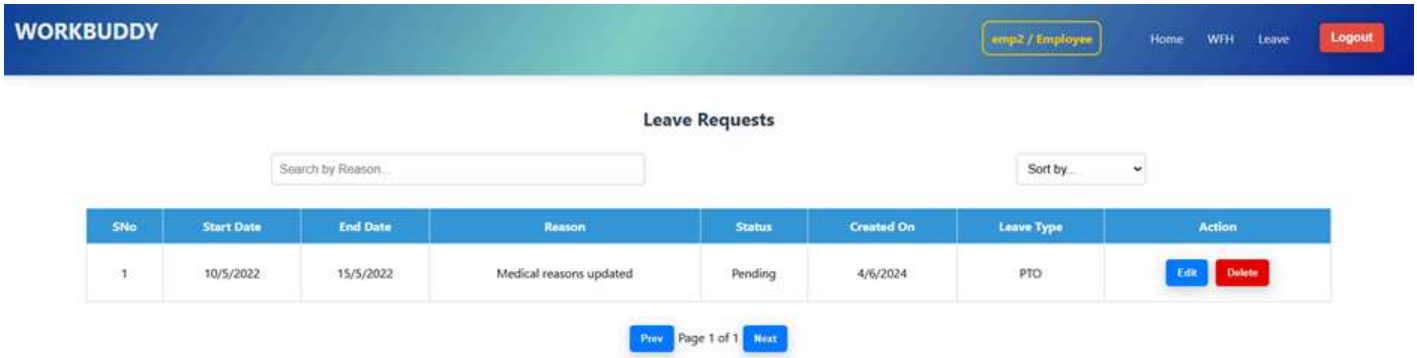
Clicking on “Ok” button will navigate to “viewLeave” component.



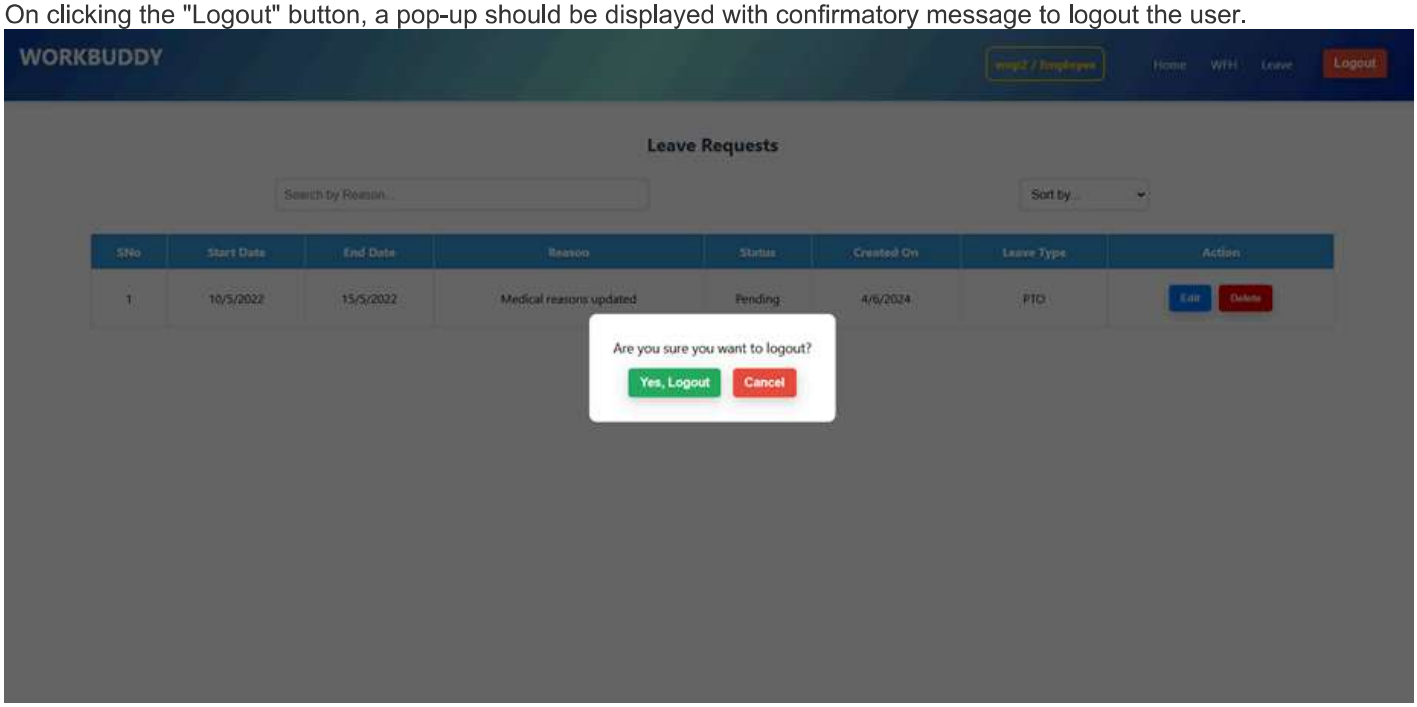
The Edit and Delete buttons can only be accessed if the status is in the Pending state.
When clicking the "Delete" button, a confirmation modal will be displayed.



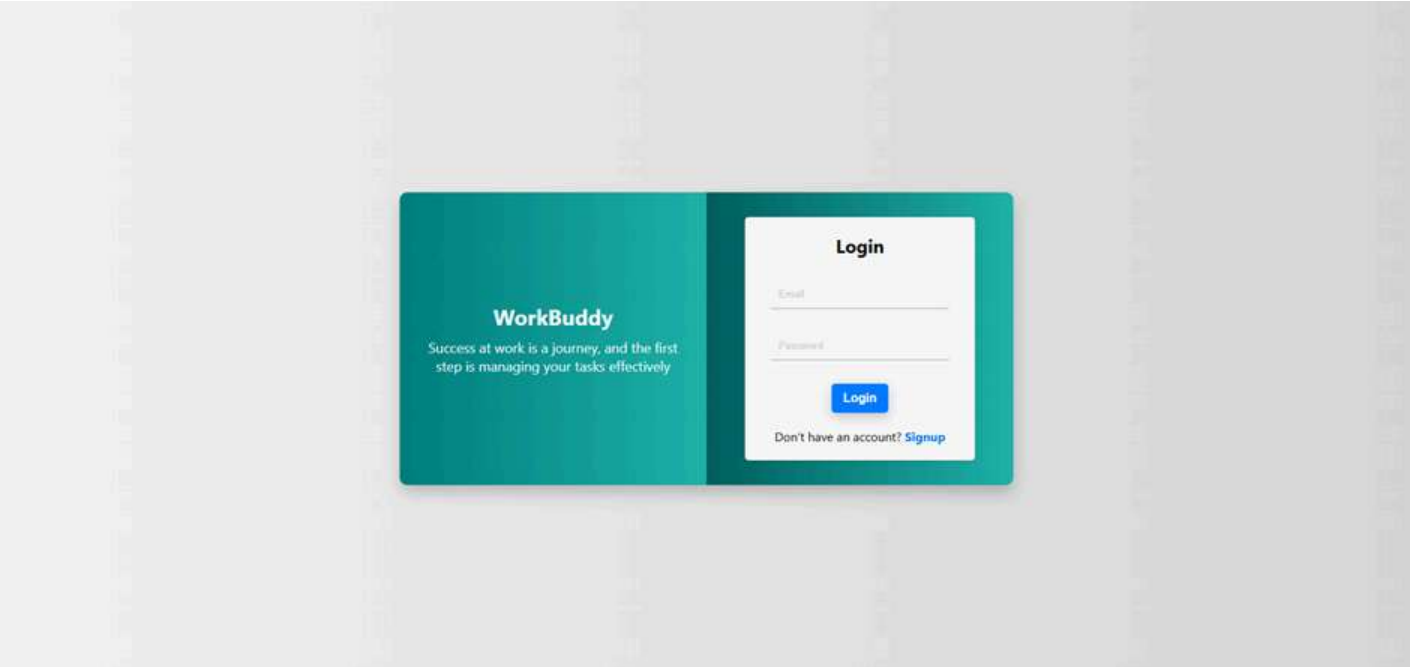
Clicking "Yes, Delete" will remove the selected request and refresh the displayed data.
Clicking “Cancel” will close the popup window.



On clicking the "Logout" button, a pop-up should be displayed with confirmatory message to logout the user.



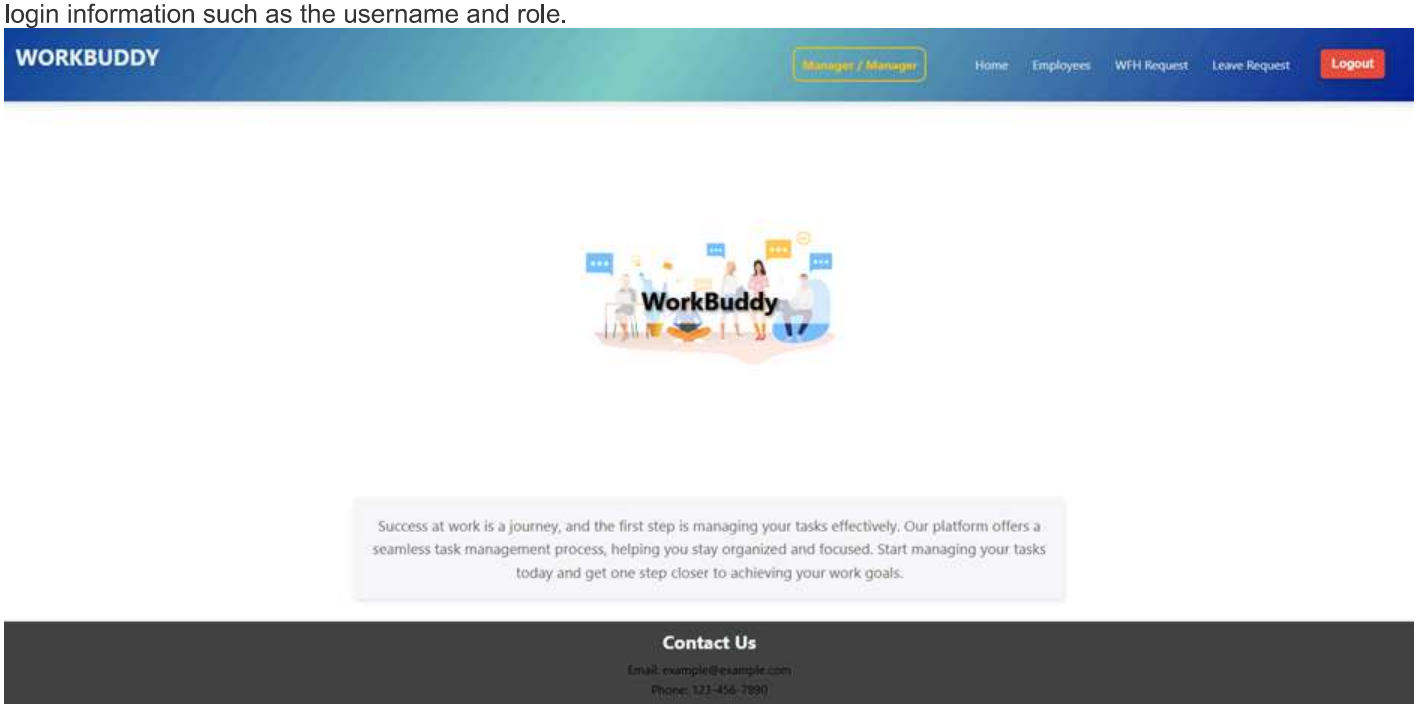
OnClicking "Yes, Logout" will navigate to the login component.



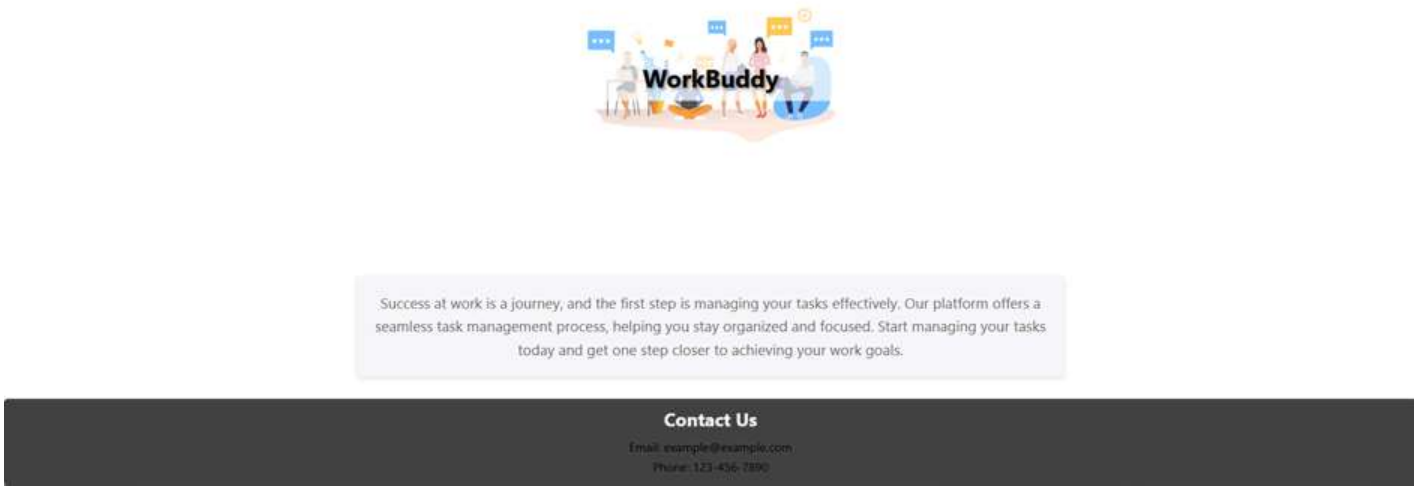
Manager Side:

managerNavbar component:

Upon successful login, if the user is an Employee, the (employeeNavbar component) will be displayed. If the user is Manager, the (managerNavbar component) will be displayed. Additionally, the role-based navigation bar will also display login information such as the username and role.



homePage Component: This page is used to display the information about workbuddy application. On clicking the ‘Home’ tab, user can view the information about the application.(workbuddycoverimage.png provided in assets/images folder). Implement “WorkBuddy”, “Success at work is a journey” and “Contact Us” as a text content in the home-page.

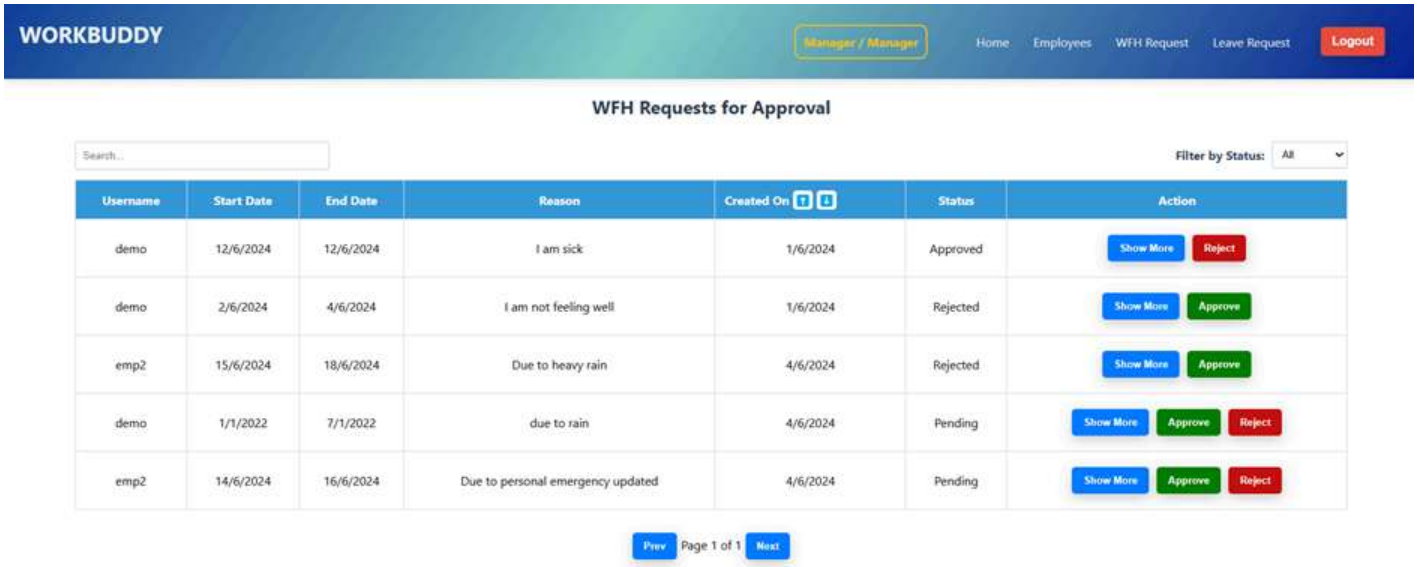


Employee Directory
On Clicking over the "Employees" item in the navbar, will navigate to “employeeList” Component

WFH Requests Approval:

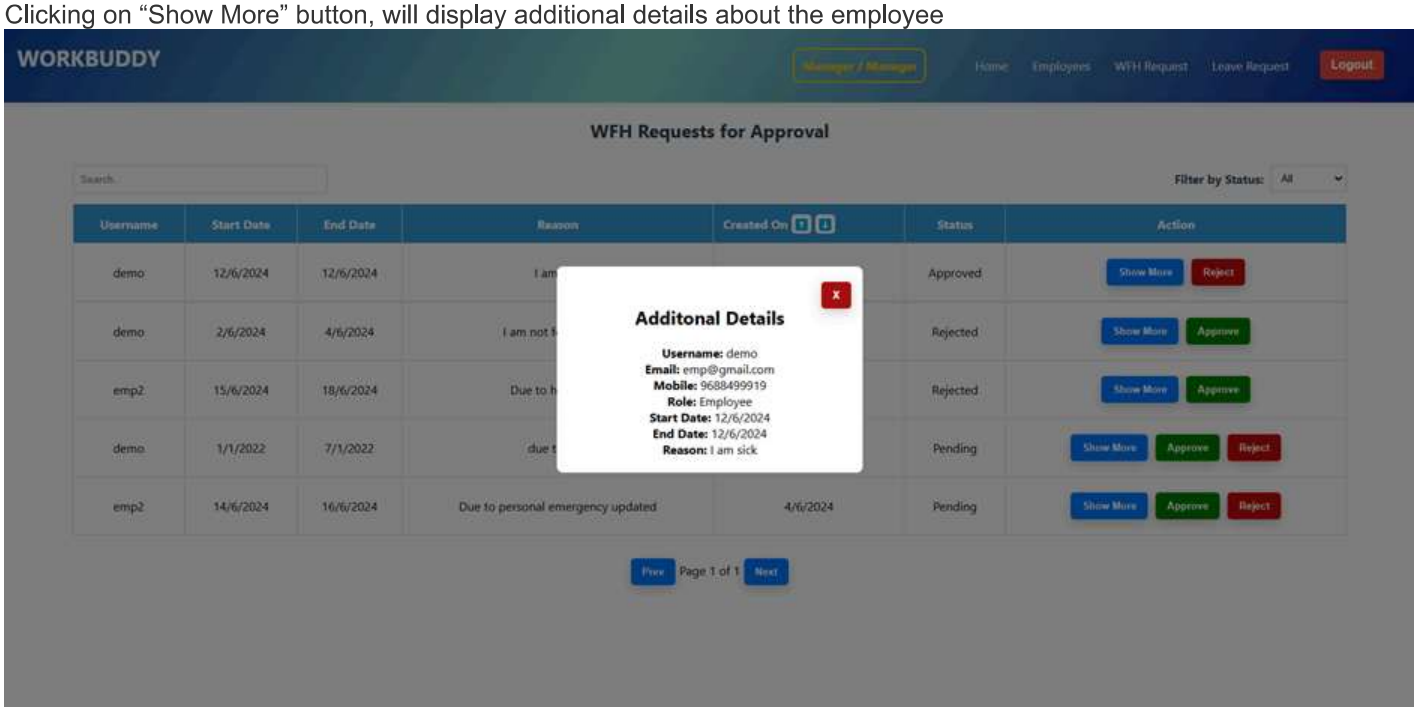
By clicking the “WFH Request” in ManagerNavbar will navigate to the “wfhRequests” component. Additionally, features containing search based on reason, filter based on status and pagination
Use text content “WFH Requests for Approval”, input element with placeholder “Search” should be present.

Implement a table to represent the requests.



Clicking "Approve" or "Reject" will update the status to "Approved" or "Rejected" respectively, based on the selection.

Clicking on “Show More” button, will display additional details about the employee

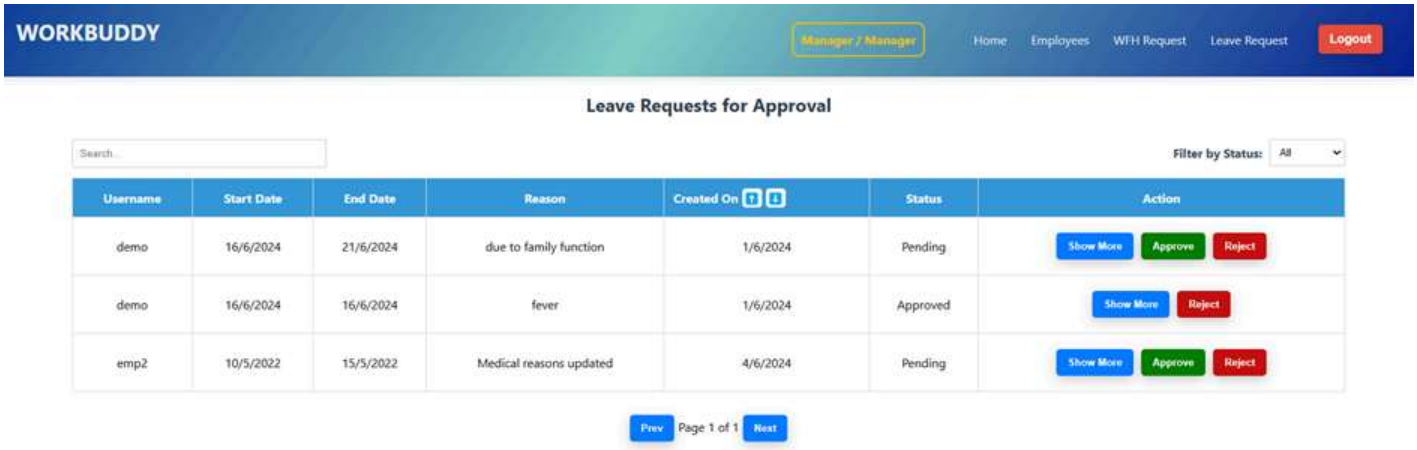


Clicking on “X” will close the popup window.

Leave Requests Approval:

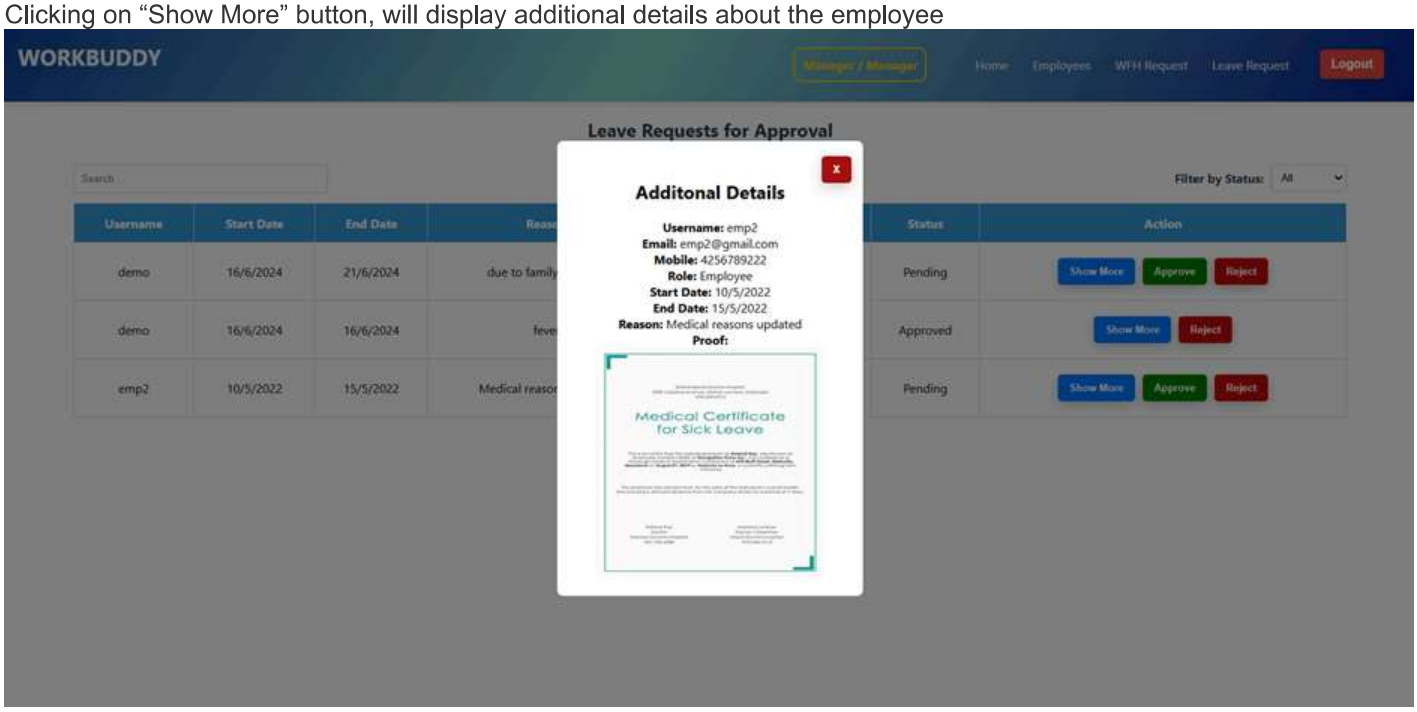
By clicking the “Leave Request” in ManagerNavbar will navigate to the “leaveRequests” component. Additionally, features containing search based on reason, filter based on status and pagination

Implement the text content “Leave Requests for Approval” for the heading and use “table” to represent the request.



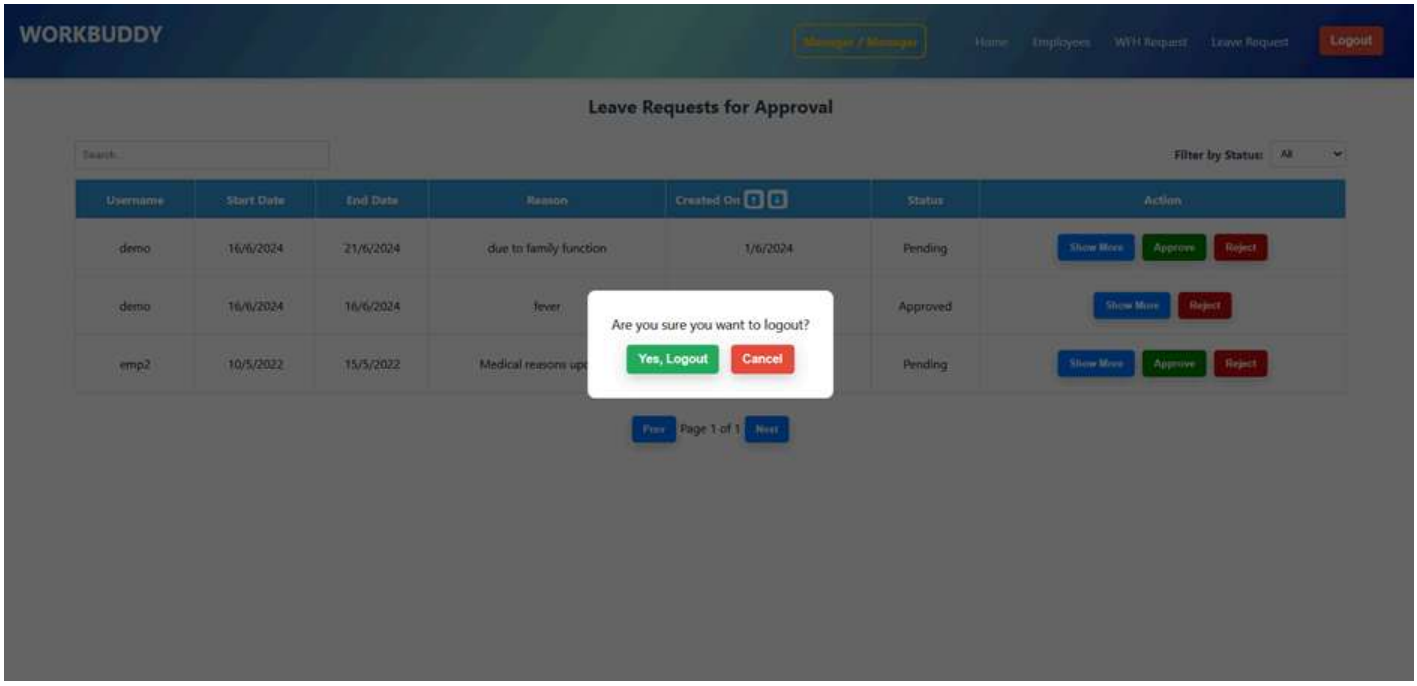
Clicking "Approve" or "Reject" will update the status to "Approved" or "Rejected" respectively, based on the selection.

Clicking on “Show More” button, will display additional details about the employee

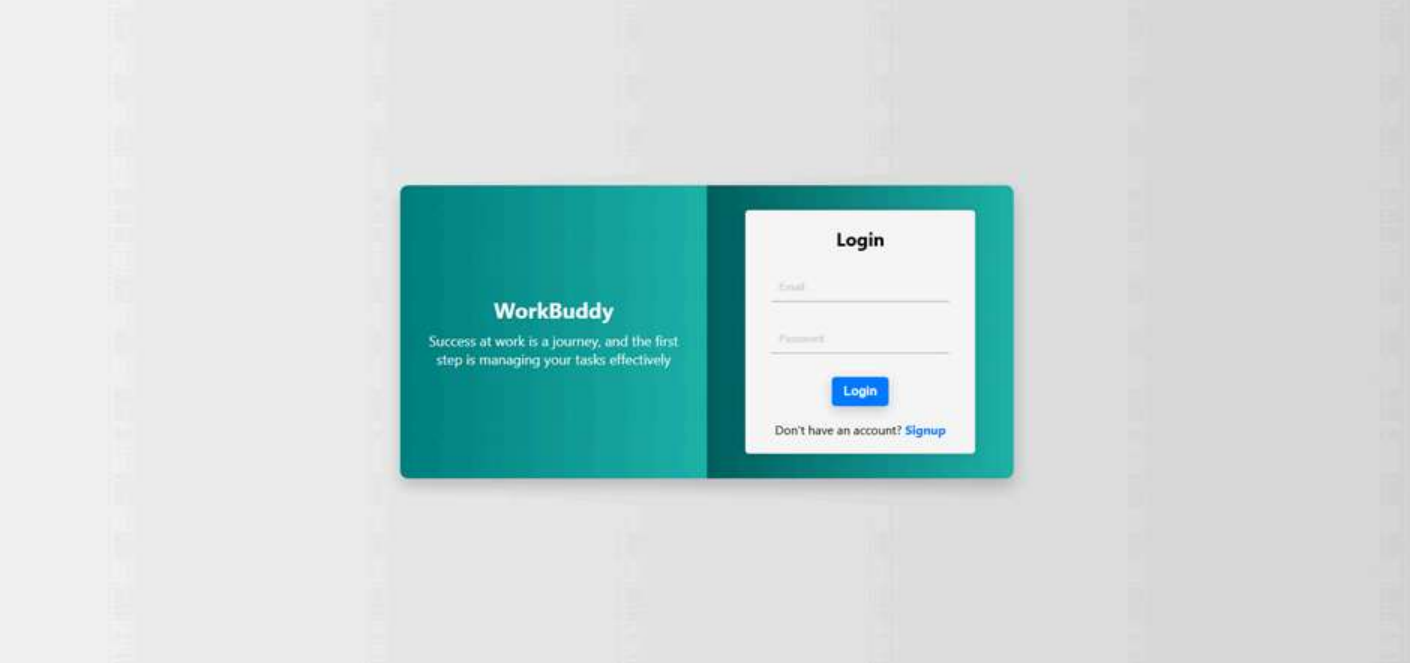


Clicking on “X” will close the popup window.

On clicking the "Logout" button, a pop-up should be displayed with confirmatory message to logout the user.

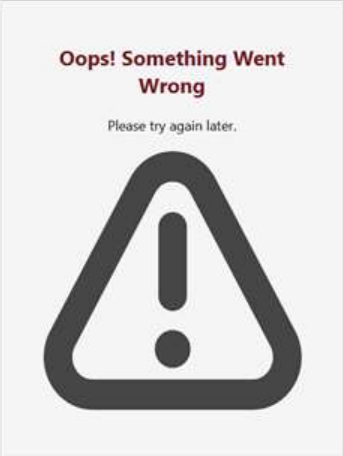


OnClicking "Yes, Logout" will navigate to the login component.



OnClicking “Cancel” will close the modal window displayed.

errorPage Component:
This page displays with “Oops! Something Went Wrong” message with the image (alert.png) provided in assets/images folder with the text content “Please try again later.”



Platform Instructions:

Frontend:
Step 1:
Use **"cd angularapp**" command to go inside the angularapp folder.
Install Node Modules **- "npm install"**

Step 2:
Write the code inside the src folder.
Create the necessary services by using the command **- "npx ng generate service <service-name>"**
Create the necessary components by using the command **- "npx ng generate component <component-name>"**

Step 3:
Command to run the project **"npm start**"

Backend:
Step 1:
Open the terminal and enter the following command to use Node version 14:
nvm use 14

Step 2:
Use the command cd nodeapp to navigate into the nodeapp directory, then install the required Node modules
cd nodeapp
npm install

Step 3:
Run the project with the following command:
npm start

Note:
- Click PORT 8081 to view the result / output
- To run the test cases, click the run test case button.
- If any error persists while running the app, delete the node modules and reinstall them