System Requirements Specification

Index

For

My Time Away

Version 1.0

TABLE OF CONTENTS

MY TIME AWAY

System Requirements Specification

You need to consume APIs exposed by Backend application in Angular to make application work as FULLSTACK

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

My Time Away is a FullStack Application with a backend implemented using Spring Boot with a MySQL database and a frontend developed using Angular. It enables users to manage various aspects and streamline the process of managing employee leaves within an organization. The system facilitates efficient management of employee leave requests and approvals, ensuring proper allocation of resources and smooth workflow management.

Following is the requirement specifications:

	My Time Away
Modules	
1	MyTimeAway
Event Module Functionalities	
1	Create an Leave
2	Update the existing Leave details
3	Get the Leave by Id
4	Get all Leaves
5	Delete a Leave
6	Search for Leave by Id, Name,and Total Days
7	Cancel a Leave by Id
8	Approve a Leave by Id
9	Reject Leave request by Id

2 Assumptions, Dependencies, Risks / Constraints

2.1 Leave Constraints

- When fetching a Leave by ID, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave with Id - " + id + " not found!".
- When updating a Leave, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave with Id - " + id + " not found!".
- When removing a Leave, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave application not found".
- When canceling a Leave, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave with Id - " + id + " not found!".
- When approving a Leave, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave with Id - " + id + " not found!".
- When rejecting a Leave, if the leave ID does not exist, the operation should throw ApplicationNotFoundException with message "Leave with Id - " + id + " not found!".

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in dto classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 Business Validations

- Employee id should not be null.
- Employee name should not be null.
- Employee phones should not be null.
- Employee email should not be null.
- Manager email should not be null.
- FromDate should not be null.
- ToDate should not be null.
- TotalDay should not be null and must be positive.
- Reason should not be null.

4 Database Operations

4.1 EmployeeLeave Entity

- **EmployeeLeave** class should be bound to a table named **`employee_leave`**.
- `id` should be the **primary key**, generated with **IDENTITY**, mapped to column **`id`**.
- `employeeId` should be **non-nullable**, mapped to column **`employee_id`**.
- `employeeName` should be **non-nullable**, mapped to column **`employee_name`**.
- `employeePhone` should be **non-nullable**, mapped to column **`employee_phone`**.
- `employeeEmail` should be **non-nullable**, mapped to column **`employee_email`**.
- `managerEmail` should be **non-nullable**, mapped to column **`manager_email`**.
- `fromDate` should be **non-nullable**, mapped to column **`from_date`**, and should use the annotation **Temporal** with **TemporalType.DATE**.
- `toDate` should be **non-nullable**, mapped to column **`to_date`**, and should use the annotation **Temporal** with **TemporalType.DATE**.
- `totalDays` should be **non-nullable**, mapped to column **`total_days`**, and should have the validation annotation **`Positive`** with the message: **"Total days must be a positive value"**.
- `reason` should be **non-nullable**, mapped to column **`reason`**.
- `isProcessed` should be mapped to column **`is_processed`**.
- `status` should be mapped to column **`status`**

5 Rest Endpoints

Rest End-points to be exposed in the controller along with method details for the same to be created. Please note, that these all are required to be implemented.

5.1 EmployeeLeave Controller

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/leaves<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>ResponseEntity<List<EmployeeLeaveDTO>></td></tr></table></td><td>Fetches all the leaves</td></tr>
<tr><td>2. /api/leaves<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>EmployeeLeaveDTO</td></tr><tr><td>Return</td><td>ResponseEntity<EmployeeLeaveDTO></td></tr></table></td><td>Add a new leave</td></tr>
<tr><td>3. /api/leaves/{id}</td><td>Delete leaves with given leave id</td></tr>
</table>

Http Method	DELETE
Parameter 1	Long (id)
Return	-

4. /api/leaves/{id}

Fetches the leave with the given id

Http Method	GET
Parameter 1	Long (id)
Return	ResponseEntity<EmployeeLeaveDTO>

5. /api/leaves/{id}

Updates existing leave

Http Method	PUT
Parameter 1	Long (id)
Parameter 2	EmployeeLeaveDTO
Return	ResponseEntity<EmployeeLeaveDTO>

6. /api/leaves/search?employeeId={id}&employeeName={name}&totalDays={totalDays}

Fetches the leave with the given id, name, total days

Http Method	GET
Parameter 1	String (id)
Parameter 2	String (name)
Parameter 3	Int (totaldays)
Return	ResponseEntity<List<EmployeeLeaveDTO>>

7. /api/leaves/{id}/cancel

Cancel the leave request

Http Method	PUT
Parameter 1	Long (id)
Return	EmployeeLeaveDTO

8. /api/leaves/{id}/approve

Approve the leave request

Http Method	PUT
Parameter 1	Long (id)
Return	EmployeeLeaveDTO

9. /api/leaves/{id}/reject

Reject the leave request

Http Method	PUT
Parameter 1	Long (id)

Return	EmployeeLeaveDTO	

6 TEMPLATE CODE STRUCTURE

6.1 PACKAGE: COM.MYTIMEAWAY

Resources

MyTimeAwayApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

6.2 PACKAGE: COM.MYTIMEAWAY.REPOSITORY

Resources

Class/Interface	Description	Status
EmployeeLeaveRepository (interface)	• Repository interface exposing CRUD functionality for EmployeeLeave Entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.

6.3 PACKAGE: COM.MYTIMEAWAY.SERVICE

Resources

Class/Interface	Description	Status
EmployeeLeaveService (interface)	• Interface to expose method signatures for employee leave related functionality. • Do not modify, add or delete any method.	Already implemented.

6.4 PACKAGE: COM.MYTIMEAWAY.SERVICE.IMPL

Resources

Class/Interface	Description	Status
EmployeeLeaveServiceImpl (class)	• Implements EmployeeLeaveService. • Contains template method implementation. • Need to provide implementation for employee leave related functionalities. • Do not modify, add or delete any method signature	To be implemented.

6.5 PACKAGE: COM.MYTIMEAWAY.CONTROLLER

Resources

Class/Interface	Description	Status
EmployeeLeaveController (Class)	• Controller class to expose all rest-endpoints for employee leave related activities. • May also contain local exception handler methods	To be implemented

6.6 PACKAGE: COM.MYTIMEAWAY.DTO

Resources

Class/Interface	Description	Status
EmployeeLeaveDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.

6.7 PACKAGE: COM.MYTIMEAWAY.ENTITY

Resources

Class/Interface	Description	Status
EmployeeLeave (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **employeeId** as primary key. • Map this class with an **employeeleave table**. • Generate the **employeeId** using the IDENTITY strategy	Partially implemented.

6.8 PACKAGE: COM.MYTIMEAWAY.EXCEPTION

Resources

Class/Interface	Description	Status
ApplicationNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the leave info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.
CustomExceptionHandler (Class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class.	Already implemented.

	• Use this as a reference for creating exception handlers for other custom exception classes.	

7 METHOD DESCRIPTIONS

1. EmployeeLeaveServiceImpl Class - Method Descriptions

Method	Task	Implementation Details
`getAllLeaves()`	Fetch all leave records	- Uses `leaveRepository.findAll()` to retrieve all leave entities - Maps each entity to DTO using `convertToDTO` - Returns the list of mapped DTOs
`getLeaveById()`	Fetch a leave by ID	- Uses `leaveRepository.findById(id)` - Throws `ApplicationNotFoundException` if a leave is not found with the message: Leave with Id - {id} not found! - Maps entity to DTO using `modelMapper` - Returns the mapped DTO
`createLeave()`	Create a new leave entry	- Converts DTO to entity using `convertToEntity` - Saves entity using `leaveRepository.save(leave)` - Returns the saved entity as DTO
`updateLeaveById()`	Update existing leave	- Retrieves leave using `leaveRepository.findById(id)` - Throws `ApplicationNotFoundException` if not found with message: Leave with Id - {id} not found! - Updates fields using `modelMapper.map` - Saves updated entity and returns as DTO
`deleteLeaveById()`	Delete leave by ID	- Checks if leave exists using `leaveRepository.findById(id)` - If found, deletes the entity using `leaveRepository.delete()` - If not found, throws `ApplicationNotFoundException` with message: Leave application not found - Returns boolean true if successful
`searchLeaves()`	Search leaves by parameters	- Retrieves all leave records - Applies filtering based on employeeId, name, and totalDays - Maps filtered results to DTOs using `convertToDTO` - Returns filtered list of DTOs

`cancelLeaveRequest()`	Cancel a leave request	- Calls `changeLeaveStatus(id, "CANCELED")` - Updates status to `CANCELED` and `isProcessed` to true - Saves and returns updated DTO
`approveLeaveRequest()`	Approve a leave request	- Calls `changeLeaveStatus(id, "APPROVED")` - Updates status to `APPROVED` and `isProcessed` to true - Saves and returns updated DTO
`rejectLeaveRequest()`	Reject a leave request	- Calls `changeLeaveStatus(id, "REJECTED")` - Updates status to `REJECTED` and `isProcessed` to true - Saves and returns updated DTO
`changeLeaveStatus()`	Helper to change leave status	- Finds leave by ID or throws `ApplicationNotFoundException` with message: Leave with Id - {id} not found! - Updates `status` and `isProcessed` fields - Saves and returns updated DTO
`convertToDTO()`	Convert Entity to DTO	- Uses `BeanUtils.copyProperties` to map properties from entity to DTO - Returns the mapped DTO
`convertToEntity()`	Convert DTO to Entity	- Uses `BeanUtils.copyProperties` to map properties from DTO to entity - Returns the mapped entity

2. EmployeeLeaveController Class - Method Descriptions

Method	**Task**	**Implementation Details**
`getAllLeaves`	Fetch all leave applications	- Calls `leaveService.getAllLeaves()` to fetch all leave records - Wraps result in `ResponseEntity` with `HttpStatus.OK` - Returns list of leave DTOs
`getLeaveById`	Fetch leave by ID	- Calls `leaveService.getLeaveById(id)` to fetch a leave record by its ID - Checks if the leave object is not null - Returns `ResponseEntity` with `HttpStatus.OK` if found, otherwise `HttpStatus.NOT_FOUND`
`createLeave`	Create new leave application	- Accepts a `EmployeeLeaveDTO` from request body - Calls `leaveService.createLeave(leaveDTO)` to save the new leave - Returns the created leave DTO wrapped in `ResponseEntity` with `HttpStatus.CREATED`

`updateLeaveById`	Update existing leave application	- Accepts a leave ID and a DTO from request body - Calls `leaveService.updateLeaveById(id, leaveDTO)` to update the record - Returns updated DTO with `HttpStatus.OK` if successful, else `HttpStatus.NOT_FOUND`
`deleteLeaveById`	Delete leave application by ID	- Calls `leaveService.deleteLeaveById(id)` to delete the leave entry - Returns `ResponseEntity` with `HttpStatus.NO_CONTENT`
`searchLeaves`	Search leaves by filters	- Accepts optional query params: `employeeId`, `employeeName`, and `totalDays` - Calls `leaveService.searchLeaves(...)` to filter based on the criteria - Returns filtered list of leave DTOs with `HttpStatus.OK`
`cancelLeaveRequest`	Cancel leave request	- Calls `leaveService.cancelLeaveRequest(id)` to set status to CANCELED - Returns the updated DTO with `HttpStatus.OK` if successful, else `HttpStatus.NOT_FOUND`
`approveLeaveRequest`	Approve leave request	- Calls `leaveService.approveLeaveRequest(id)` to set status to APPROVED - Returns the updated DTO with `HttpStatus.OK` if successful, else `HttpStatus.NOT_FOUND`
`rejectLeaveRequest`	Reject leave request	- Calls `leaveService.rejectLeaveRequest(id)` to set status to REJECTED - Returns the updated DTO with `HttpStatus.OK` if successful, else `HttpStatus.NOT_FOUND`

8 CONSIDERATIONS

A. There is no roles in this application
B. You can perform the following possible action

EmployeeLeave

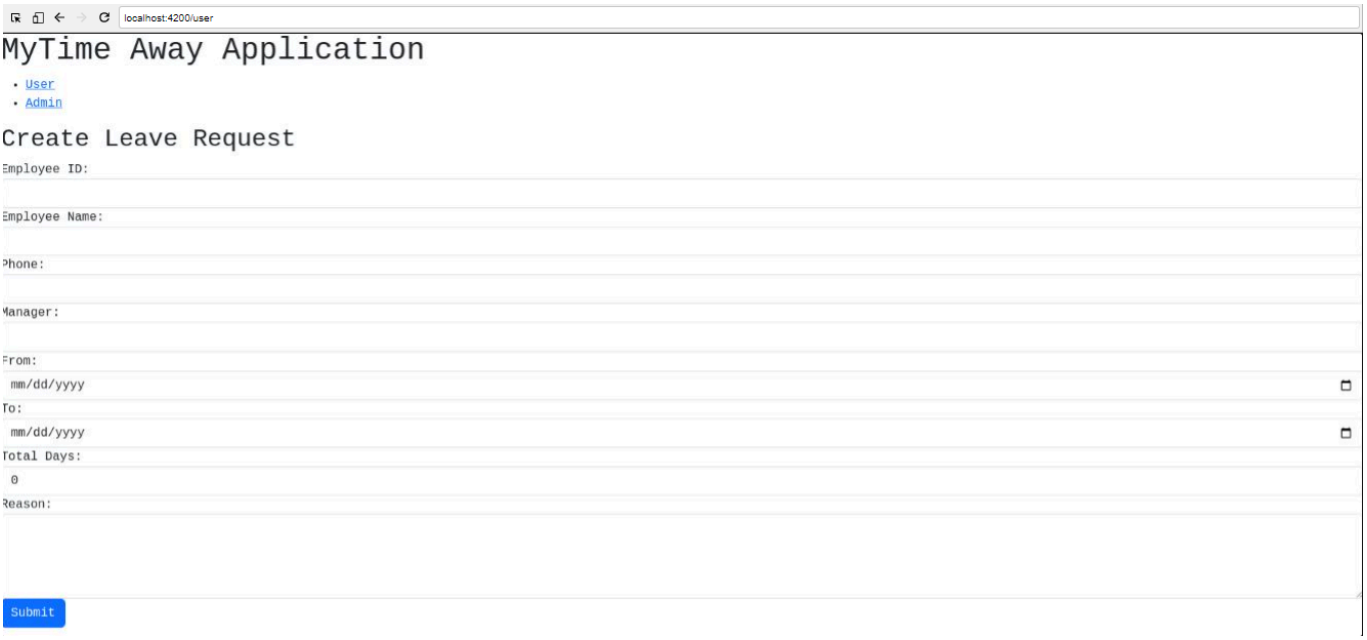
1 PROBLEM STATEMENT

My Time Away is SPA (Single Page Application), it enables users to manage various aspects and streamline the process of managing employee leaves within an organization like it allows to add leave, update leave, delete leave, get leave by id, get all leave, search leave by id, name and total days, cancel leave request, approve leave request and reject leave request.

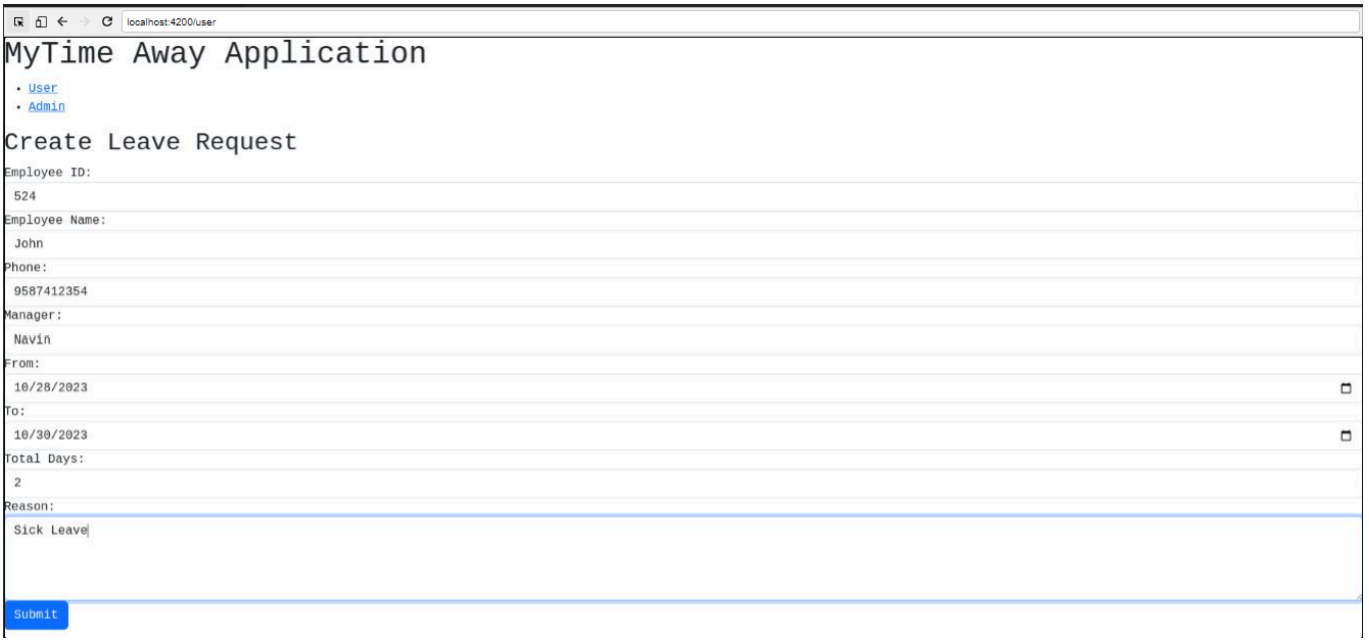
2 PROPOSED MY TIME AWAY WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

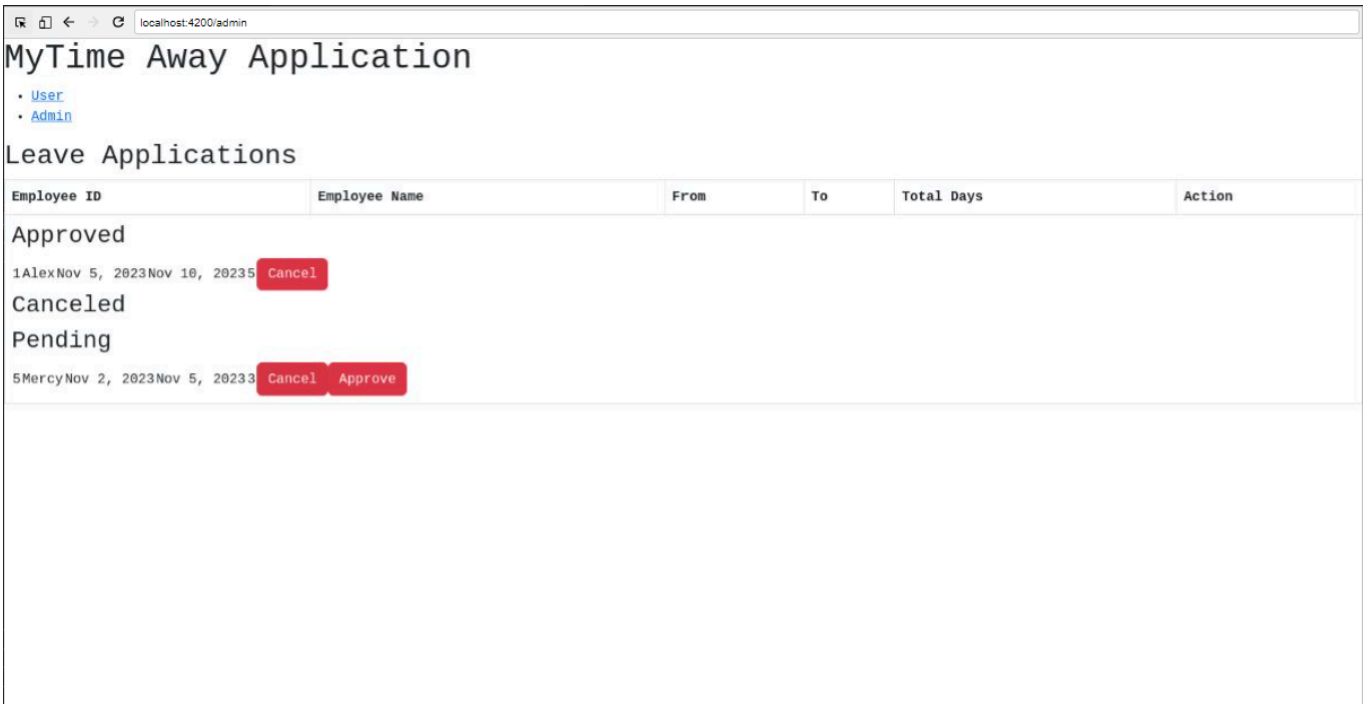
2.1 HOME PAGE



2.2 USER PAGE



2.3 ADMIN PAGE



3 BUSINESS-REQUIREMENT:

As an application developer, develop the Social Networking App (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the Home page as the default page. **App Shell — `app.component.html`** **HTML Structure** ● **Top-level `<h1>` heading:** Text must be **"MyTime Away Application."** ● **Header with navigation (`<header><nav><ul>`):** ○ Two list items with router links: ■ **User** → navigates to `/user` ■ **Admin** → navigates to `/admin` ● **Router outlet:** Place a single `<router-outlet>` after the header to render routed components. **Dynamic Behavior** ● Clicking **User** loads the **UserComponent** view inside the outlet. ● Clicking **Admin** loads the **AdminComponent** view inside the outlet. ● Heading and header remain static while routed content changes. **Routing — `app-routing.module.ts`**

		## Purpose Define top-level routes and fallbacks for the app. ## Routes - `/user` → **UserComponent** - `/admin` → **AdminComponent** - **Default redirect:** empty path (`''`) redirects to `/user` - **Wildcard redirect:** any unknown path (`**`) redirects to `/user`. ## Functions & Responsibilities - **Configure root RouterModule:** provide the routes above. - **Export RouterModule:** make routing available to the app module. ## Root Component — `app.component.ts` ## Purpose - Acts as the application shell. - Supplies the title and hosts the header + `router-outlet`. ## State - **title:** set to **"MyTime Away Application."** ## Responsibilities - Bind the title to the heading in the template. - Provide a static frame around routed content (no extra logic required).

US_01	User Page	As a user I should be able to see the homepage and perform all operations:
		
		## Acceptance criteria:
		
		## User Component — `user.component.html`
		
		### HTML Structure
		
		- **Heading**
		  - Use an **h2** heading with text: **Create Leave Request**.
		
		- **Form Container**
		  - A single **form** element configured to submit via a component method.
		  - Add a **template reference** for validation state (e.g., `#leaveForm="ngForm"`).
		  - Mark the form as **novalidate** to rely on Angular validations.
		
		- **Form Fields (each wrapped in a form-group)**
		  - **Employee ID**
		  - Text input bound to the model for employee ID.
		  - **Employee Name**
		  - Text input bound to the model for employee name.
		  - **Phone**
		  - Text input bound to the model for employee phone.
		  - **Required**.
		  - **Validation message** displayed only when the control is invalid and touched/dirty:
		  - “Phone is required.”
		  - **Manager**
		  - Text input bound to the model for manager email.
		  - **Required**.
		  - **Validation message** displayed when invalid and touched/dirty:
		  - “Manager is required.”
		  - **From**
		  - Date input bound to the model for start date.
		  - **Required**.
		  - **Validation message** displayed when invalid and touched/dirty:
		  - “From date is required.”

- **To**
  - Date input bound to the model for end date.
  - **Required**.
  - **Validation message** displayed when invalid and touched/dirty:
    - "To date is required."
- **Total Days**
  - Number input bound to the model for total leave days.
  - **Required**.
  - **Validation message** displayed when invalid and touched/dirty:
    - "Total Days is required."
- **Reason**
  - Textarea bound to the model for leave reason.
  - **Required**.
  - **Validation message** displayed when invalid and touched/dirty:
    - "Reason is required."

- **Submit Button**
  - A single **submit** button to trigger form submission.

User Component — `user.component.ts`

Purpose

Handle **creation of a leave request**: build reactive form controls, hold the leave model used by the template form, submit the request to the backend, and navigate on success.

State

- **leaveForm**: Reactive FormGroup with controls for all required fields (employeeName, employeePhone, managerEmail, fromDate, toDate, totalDays, reason).
- **leave**: Object holding the current leave request data (id, employeeId, employeeName, employeePhone, managerEmail, fromDate, toDate, totalDays, reason, status, employeeEmail, isProcessed).

Functions & Responsibilities

- **constructor(fb, leaveService, router)**
  - Injects dependencies.
  - Calls the form initializer.

		• **createForm()**
		  ○ Builds the **reactive form**:
		  ■ Adds controls for: employeeName, employeePhone, managerEmail, fromDate, toDate, totalDays, reason.
		  ■ Applies **required** validators to each of the controls above.
		  ■ Seeds default values where provided (e.g., reason from the `leave` object).
		
		• **submitForm()**
		  ○ Sends the current `leave` object to the backend via the service.
		  ○ **On success**: navigates to the **Admin** route (`/admin`).
		  ○ **On error**: logs the error to the console with message:
		  ■ "Error creating leave request:" followed by the error.
		
		## Dynamic Behavior
		
		• The template-driven inputs (`ngModel`) update the `leave` object as the user types.
		• The reactive `leaveForm` mirrors validation requirements; error messages in the HTML are shown for required fields when touched/dirty and invalid.
		• Submitting the form triggers the create call:
		  ○ Success → route changes to Admin page.
		  ○ Failure → error is printed in the console .
		
		**** Kindly refer to the screenshots for any clarifications. ******

US_02	Admin Page	As a admin I should be able to see the admin and perform all operations: ### Acceptance criteria: ## Admin Component — `admin.component.html` ## HTML Structure • **Page Heading** ○ Use an **h2** heading with the text: **Leave Applications**. • **Outer Table (container)**

		○ A **bordered table** with a single header row containing these column headings (in this exact order): ■ Employee ID ■ Employee Name ■ From ■ To ■ Total Days ■ Action ○ The **body** contains one full-width row (`colspan="6"`) that groups three subsections: **Approved**, **Canceled**, and **Pending**. Each subsection has its own title and nested table. ● **Approved Section** ○ **Subheading:** use **h3** with the text **Approved**. ○ A **nested table** to list only applications whose status is **APPROVED**. ○ Each data row displays: ■ Employee ID ■ Employee Name ■ From (formatted as a date) ■ To (formatted as a date) ■ Total Days ■ Action column with a **Cancel** button. ○ Rows for approved items should visually indicate approved status. ● **Canceled Section** ○ **Subheading:** use **h3** with the text **Canceled**. ○ A **nested table** to list only applications whose status is **REJECTED**. ○ Each data row displays: ■ Employee ID ■ Employee Name ■ From (formatted as a date) ■ To (formatted as a date) ■ Total Days ■ Action column. ○ Rows for canceled items should visually indicate canceled/rejected status. ● **Pending Section** ○ **Subheading:** use **h3** with the text **Pending**. ○ A **nested table** to list applications whose status is **empty** (pending). ○ Each data row displays: ■ Employee ID ■ Employee Name ■ From (formatted as a date) ■ To (formatted as a date)

- Total Days
- Action column with two buttons:
  - **Cancel**
  - **Approve**

Dynamic Behavior

- The three subsections filter the same underlying collection into **Approved**, **Canceled**, and **Pending** views.
- Clicking **Cancel** in Approved or Pending should trigger the cancel action for that request.
- Clicking **Approve** in Pending should trigger the approve action for that request.
- After any action, the list should refresh to reflect the latest statuses.

Admin Component — `admin.component.ts`

Purpose

Display all leave applications grouped by status and allow the admin to **approve** or **cancel** requests. All actions refresh the list.

State

- **leaveApplications**: array holding all leave requests fetched from the backend.

Lifecycle

- **ngOnInit()**
  - Loads the current list of leave requests immediately when the component initializes.

Functions & Responsibilities

1. **fetchLeaveApplications()**
   - Retrieves all leave requests from the backend.
   - On success: updates `leaveApplications` with the latest data.
   - On error: logs the message **"Error fetching leave applications:"** followed by the error.

2. **cancelLeave(application)**
   - Sends a request to cancel the specified leave

		(identified by its id). ○ On success: calls `fetchLeaveApplications()` to refresh the list. ○ On error: logs the message **"Error canceling leave application:"** followed by the error. 3. **approveLeave(application)** ○ Sends a request to approve the specified leave (identified by its id). ○ On success: calls `fetchLeaveApplications()` to refresh the list. ○ On error: logs the message **"Error approving leave application:"** followed by the error. ## Dynamic Behavior ● Initial load: `ngOnInit()` triggers `fetchLeaveApplications()` to populate the table. ● Actions: ○ **Approve** (visible only in Pending): updates status on the server, then refreshes the UI. ○ **Cancel** (visible in Approved and Pending): updates status on the server, then refreshes the UI. ● Grouping: ○ **Approved**: items with status **APPROVED**. ○ **Canceled**: items with status **REJECTED**. ○ **Pending**: items with an **empty** status string. **** Kindly refer to the screenshots for any clarifications. ****
		## Leave Service — `leave.service.ts` ### Purpose Provide a single place to call the backend API for **leave applications** at `http://127.0.0.1:8081/mytimeaway/api/leaves` — including CRUD, search, and status actions (approve/cancel/reject). ### Base ● **Base URL:** `http://127.0.0.1:8081/mytimeaway/api/leaves`

		• **Transport:** Angular `HttpClient`
		
		Functions & Responsibilities
		
		`getAllLeaves()`
		
		• Sends a **GET** request to retrieve all leave applications.
		• Used in the **Admin component** to display every leave request on load or refresh.
		• **Returns:** The HTTP response containing an **array of all leave objects**.
		
		`getLeaveById(id)`
		
		• Sends a **GET** request to fetch a **single leave** by its unique ID.
		• Used when viewing or editing the details of one leave application.
		• **Returns:** The HTTP response containing the **leave object** matching the ID.
		
		`createLeave(leave)`
		
		• Sends a **POST** request to create a **new leave** with the provided data.
		• Called when a user submits the **"Create Leave Request" form**.
		• **Returns:** The HTTP response containing the **newly created leave object**.
		
		`updateLeaveById(id, leave)`
		
		• Sends a **PUT** request to update an **existing leave** with the given ID.
		• Called when editing a leave request that is already stored.
		• **Returns:** The HTTP response containing the **updated leave object**.
		
		`deleteLeaveById(id)`
		
		• Sends a **DELETE** request to remove a leave application by ID.
		• Used for deleting obsolete or unwanted leave records.
		• **Returns:** The HTTP response with **void**.
		
		`searchLeaves(employeeId?, employeeName?, totalDays = 0)`
		
		• Sends a **GET** request with **query parameters** to filter leaves by:
		  ○ `employeeId`
		  ○ `employeeName`

		○ `totalDays` (included only if > 0) ● Used when admins need to **search/filter leave records**. ● **Returns:** The HTTP response containing an **array of filtered leave objects**. **`cancelLeaveRequest(id)`** ● Sends a **PUT** request to change a leave status to **Canceled**. ● Called when an admin clicks **Cancel** for a leave request. ● **Returns:** The HTTP response containing the **updated leave object** with canceled status. **`approveLeaveRequest(id)`** ● Sends a **PUT** request to change a leave status to **Approved**. ● Called when an admin approves a **Pending** leave. ● **Returns:** The HTTP response containing the **updated leave object** with approved status. **`rejectLeaveRequest(id)`** ● Sends a **PUT** request to change a leave status to **Rejected**. ● Used when an admin explicitly **rejects** a leave request. ● **Returns:** The HTTP response containing the **updated leave object** with rejected status. ## Dynamic Behavior ● Admin views typically call **`getAllLeaves()`** on init and after **approve/cancel/reject** to refresh the lists. ● **Search** adds only the provided filters as query params; missing filters are **not** sent. ● Status endpoints (**approve/cancel/reject**) expect **no body** beyond `{}` and return the **updated** record so the UI can reflect the new status immediately. **** Kindly refer to the screenshots for any clarifications. ****

4 EXECUTION STEPS TO FOLLOW FOR BACKEND

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **cd into your backend project folder**
4. **To build your project use command:**

   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

   java -jar <your application jar file name>
6. **This editor Auto Saves the code.**
7. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
8. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**
9. **To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
10. **Default credentials for MySQL:**
    a. **Username: root**
    b. **Password: pass@word1**
11. **To login to mysql instance: Open new terminal and use following command:**
    a. **sudo systemctl enable mysql**
    b. **sudo systemctl start mysql**
    c. **mysql -u root -p**

    The last command will ask for password which is 'pass@word1'
12. **Mandatory: Before final submission run the following command:**

    mvn test

1 EXECUTION STEPS TO FOLLOW FOR FRONTEND

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

3. **This is a web-based application, to run the application on a browser, use the internal browser in the environment.**
4. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min**
   b. **npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min**
   c. **npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**