EVENT MANAGEMENT APP

1. Import the necessary modules and components in the respective files.

2. Create a `SigninComponent` with the following functionalities: - Initialize `username`, `password`, and `msg` variables. - Implement the `LoginCheck` method to handle the form submission. - Inside `LoginCheck`: - Check if the form is valid. - Retrieve registration data from local storage. - Validate the entered username and password. - If the login is successful: - Set the logged-in user in local storage. - Set a success message. - Redirect to the home page after a short delay. - If the login fails, display an error message.

3. Create a `DashboardComponent` with the following functionalities: - Initialize `loggedInUser`, `showResetForm`, `newPassword`, and `message` variables. - Implement the `resetPassword` method to handle the password reset form submission. - Implement the `findById` method to handle searching by ID.

4. Configure routing in the Angular app to navigate between the `SigninComponent` and

`DashboardComponent`.

5. Create HTML templates for the `SigninComponent` and `DashboardComponent` with the provided

code snippets.

6. Style the components as desired.

7. Update the `dashboard.component.ts` file: - Import the `EmployeeService` from `'src/app/employee.service'`. - Add the `EmployeeService` to the constructor. - Initialize the `loggedInUser` and `showResetForm` variables. - Implement the `ngOnInit` method

to retrieve the logged-in user from localStorage. - Implement the `findById` method to fetch employees by ID or all employees. - Implement the `resetPassword` method to update the password for the logged-in user and display a

success message.

8. Create a component inside `src/app/homepage/dashboard/` using the command `ng g c add

employee`.

9. Update the `add-employee.component.html` file: - Add a navigation bar. - Implement the form for adding employees with validation.

10. Create a component inside `src/app/homepage/dashboard/` using the command `ng g c employee

list`.

11. Update the `employee-list.component.html` file: - Add a navigation bar. - Implement the employee list table with search functionality.

12. Update the `employee-list.component.ts` file: - Import the `EmployeeService` from `'src/app/employee.service'`. - Add the `EmployeeService` to the constructor. - Implement the `ngOnInit` method to fetch all employees. - Implement the `findById` method to fetch employees by ID or all employees.

13. Create a component inside `src/app/homepage/dashboard/` using the command `ng g c update`.

14. Update the `update.component.html` file: - Add a navigation bar. - Implement the update form.

15. Update the `update.component.ts` file: -

Import the `ActivatedRoute` and `EmployeeService` from `'@angular/router'` and

`'src/app/employee.service'`, respectively. - Add the `ActivatedRoute` and `EmployeeService` to the constructor. - Implement the `ngOnInit` method to retrieve the employee by ID. - Implement the `updateNow` method to update the employee data.

16. Create a component inside `src/app/homepage/dashboard/` using the command `ng g c view

details`.

17. Update the `view-details.component.html` file: - Add a navigation bar. - Implement the employee details view with options to update and delete.

18. Update the `view-details.component.ts` file: - Import the `ActivatedRoute`, `EmployeeService`, `Router`, and `ChangeDetectorRef` from

`'@angular/router'` and `'src/app/employee.service'`, respectively. - Add the `ActivatedRoute`, `EmployeeService`, `Router`, and `ChangeDetectorRef` to the

constructor. - Implement the `ngOnInit` method to retrieve the employee by ID. - Implement the `updateNow` method to update the employee data. - Implement the `deleteEmployee` method to delete the employee data.

Note: Make sure to update the necessary imports and service method calls according to your

implementation and requirements.

Note: Make sure to import and declare the components in the appropriate module files and add the

required routes to the routing configuration.