Name:	1. Calderon Jay Mark	Date: 03/06/2025	Section: IDB2
	2. De Castro Ralph Vincent		
Self-Guided Laboratory Activity: Building an Angular App with Routing and GitHub Collaboration		**Activity 4b**	Score:

This activity is designed to be self-paced and encourages students to learn through hands-on experience and collaboration.

Objective:

- Understand and implement different Angular v.19 routing methods.
- Collaborate to build a simple Angular app with multiple routes.
- Use GitHub for version control and collaboration.
- Submit the completed project to a GitHub repository with descriptive commit messages.

Materials Needed:

- Computers with Angular CLI installed
- Code editor (e.g., Visual Studio Code)
- Internet access for documentation and resources
- GitHub accounts for each student

Pre-Lab Preparation:

- Group by pairs.
- Ensure all students have Node.js, Angular CLI, and Git installed on their computers.
- One of the students in the pair will create a GitHub repository for the project and invite his/her partner to be a collaborator.
- Choose a routing method to create, merge your work, commit and push to GitHub.

Instructions:

1. **Create a GitHub Repository and Add Collaborators:**
   - One student creates a new repository on GitHub:
     1. Go to GitHub and log in.
     2. Click on the "+" icon in the top right corner and select "New repository".
     3. Name the repository (lastname1-lastname2-angular-routing-lab), add a description, and click "Create repository".
   - Add the other student as a collaborator:
     1. Go to the repository page.
     2. Click on "Settings" > "Collaborators".
     3. Add the GitHub username of the other student and click "Add collaborator".

2. **Create a new Angular project.**

```
ng new angular-routing-lab
cd angular-routing-lab
```

3. **Initialize Git:**
   - In the project directory, initialize Git and connect to your GitHub repository:

```
git init
git remote add origin <repository-url>
```

4. **Creating Components (10 minutes):**
   - Create three components: Home, About, and Contact. Use the following commands:
     `ng generate component home`
     `ng generate component about`

     `ng generate component contact`

5. **Setting Up Routes (15 minutes):**
   - Open the `app-routing.module.ts` file and define the routes for the components:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from
'./contact/contact.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent ];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

6. **Adding Navigation Links (10 minutes):**
   - In the `app.component.html` file, add navigation links to the components:

```
<nav>
  <a routerLink="/home">Home</a>
  <a routerLink="/about">About</a>
  <a routerLink="/contact">Contact</a>
</nav>
<router-outlet></router-outlet>
```

7. **Implementing Child Routes (20 minutes):**
   - Create a new component for a child route, e.g., `Profile` under `About`:

```
ng generate component about/profile
```

   - Update the app-routing.module.ts to include child routes:

```
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent, children: [
    { path: 'profile', component: ProfileCompogtnent }
  ]},
  { path: 'contact', component: ContactComponent }
];
```

8. **Lazy Loading Modules (30 minutes):**
   - Create a new module for lazy loading, e.g., `Admin`:

```
ng generate module admin --route admin --module app.module
```

   - This command automatically sets up lazy loading for the `Admin` module.

9. **Route Guards (30 minutes):**
   - Create a route guard to protect the `Admin` route:

```
ng generate guard admin/admin
```

   - Implement the guard logic in `admin.guard.ts`:

```
 import { Injectable } from '@angular/core';
 import { CanActivate, ActivatedRouteSnapshot,
RouterStateSnapshot, UrlTree } from '@angular/router';
 import { Observable } from 'rxjs';

 @Injectable({
   providedIn: 'root'
 })
 export class AdminGuard implements CanActivate {
   canActivate(
     next: ActivatedRouteSnapshot,
    Tree> | Promise<boolean | UrlTree> | boolean | UrlTree {
     // Add your authentication logic here
     return true; // Change this to actual authentication
   check }
 }
Update the app-routing.module.ts to use the guard:
 const routes: Routes = [
   { path: '', redirectTo: '/home', pathMatch: 'full' },
   { path: 'home', component: HomeComponent },
   { path: 'about', component: AboutComponent, children: [
     { pathprofile', component: ProfileComponent }
   ]},
   { path: 'contact', component: ContactComponent },
   { path: 'admin', loadChildren: () =>
import('./admin/admin.module').then(m => m.AdminModule),
canActivate: [AdminGuard] }
 ];
```

10. **Testing the Application (20 minutes):**
   - Run the application using the following command:

```
ng serve
```

   - Open a web browser and navigate to `http://localhost:4200`. Test the navigation links and routes to ensure everything works correctly.

11. **Submitting to GitHub (15 minutes):**
   - Initialize a Git repository in the project directory:

```
 git init
 git add .
 git commit -m "Initial commit"
Push the project to a GitHub repository:
 git remote add origin <your-github-repo-url>
 git push -u origin master
```

Learning Outcomes:

- Students will understand and implement different Angular v.19 routing methods.
- Students will collaborate to build a functional Angular app with multiple routes.
- Students will submit their completed project to a GitHub repository.

Here are the expected outputs for each routing method in the activity:

1. Basic Routing

Expected Output:

- The application should navigate between the Home, About, and Contact components using the defined routes.
- The navigation links in the app.component.html should work correctly, allowing users to switch between the different components.
- The URL should update accordingly when navigating to /home, /about, and /contact.

2. Child Routes
Expected Output:

- The About component should have a nested route for the Profile component.
- Navigating to /about/profile should display the Profile component within the About component. ●
The URL should update to reflect the nested route structure.

3. Lazy Loading Modules
Expected Output:

- The Admin module should be lazy-loaded when navigating to the /admin route.
- The Admin component should be displayed when navigating to /admin.
- The application should only load the Admin module when the /admin route is accessed, improving the initial load time of the application.

4. Route Guards
Expected Output:

- The Admin route should be protected by the AdminGuard.
- The guard logic should determine whether the user can access the /admin route.
- If the guard condition is not met, the user should be redirected or prevented from accessing the Admin component.

Summary of Expected Outputs:

1. **Basic Routing:**
   - o Functional navigation between Home, About, and Contact components.
   - o Correct URL updates for each route.
2. **Child Routes:**
   - o Nested routing within the About component.
   - o Correct URL updates for nested routes.
3. **Lazy Loading Modules:**
   - o Lazy-loaded Admin module.
   - o Improved initial load time by loading the Admin module only when needed.
4. **Route Guards:**
   - o Protected Admin route with guard logic.

Reflection Questions:

1. **Technical Challenges:**
   - o What specific technical challenges did you encounter while implementing the routing methods, and how did you resolve them?
     - The linking of the project and merging
   - o Were there any errors or bugs that were particularly difficult to debug? How did you approach solving them?
     - the router link, because it must be connected to the project
   - o How do child routes and lazy loading improve the structure and performance of an Angular application?

     - to make the project much better
   - o What is the purpose of route guards, and how do they enhance the security of an application?
     - we installed admin guard so our project is secured
   - o How did collaborating with your peers help you understand Angular routing better?
     - yes, but sometimes were confused
2. **Learning Process:**
   - o How did this hands-on activity help you understand Angular routing better compared to just reading or watching tutorials?
     - because you are doing it physically, and troubleshooting any errors so it much more helpful o What new concepts or techniques did you learn during this activity that you were not familiar with before?
     - the routing link
3. **Collaboration:**
   - o How did working in a group influence your understanding of Angular routing? Did you find it helpful to discuss and solve problems together?
     - yes it is helpful
   - o What strategies did your group use to ensure effective collaboration and communication?
     - we must communicate from time to time
4. **Application Design:**
   - o How did you decide on the structure and design of your application? What factors influenced your design choices?
     - it is influenced by our past projects
   - o How do you think the routing methods you implemented will impact the user experience of your

application?
  - because they can click the 4 buttons above that can guide them to new text

5. **Best Practices:**
   - o What best practices did you follow while setting up the routes and organizing your code?
     - communicating and in order coding
   - o How did you ensure that your code is maintainable and scalable?
     - we only did the basic ones
6. **Real-World Applications:**
   - o Can you think of real-world applications or websites that use similar routing methods? How do they benefit from these methods?
     - they will benefit from these methods because they can implement this too
   - o How would you apply what you learned in this activity to a real-world project or job?
     - I apply what i learned in this project like communicating from time to time and using scalable codes

Grading Rubric

Criteria	**Excellent (4)**	**Good (3)**	**Satisfactory (2)**	**Needs Improvement (1)**
Basic Routing Implementation	All routes (Home, About, Contact) are correctly implemented and functional.	Most routes are correctly implemented and functional, with minor issues.	Some routes are correctly implemented, but there are noticeable issues.	Routes are incorrectly implemented or non-functional.
Child Routes Implementation	Child routes are correctlyimplemented and functional, with clear nested navigation.	Child routes are mostly correct, with minor issues in navigation.	Child routes are partially correct, but there are noticeable issues.	Child routes are incorrectly implemented or non-functional.
Lazy Loading Implementation	Lazy loading is correctly implemented for the Admin module, improving load time.	Lazy loading is mostly correct, with minor issues in implementation.	Lazy loading is partially correct, but there are noticeable issues.	Lazy loading is incorrectly implemented or non-functional.

Implementation
Route guards are correctly implemented, effectively protecting routes.
Route guards are mostly correct, with minor issues in protection logic.
Route guards are partially correct, but there are noticeable

issues.
Route guards are incorrectly implemented or non-functional.

Route Guards

Code Quality	Code is well-organized, clear, and follows best practices.	Code is mostly well-organized and clear, with minor issues.	Code is somewhat organized, but there are noticeable issues.	Code is poorly organized and unclear.
GitHub Collaboration	Regular commits with descriptive messages, active collaboration.	Regular commits with some descriptive messages, moderate collaboration.	Infrequent commits, minimal collaboration.	Few or no commits, lack of collaboration.
Project Functionality	All routes and components function as expected.	Most routes and components function as expected.	Some routes and components function as expected.	Many routes and components do not function as expected.
Reflection Responses	Reflection responses are insightful, well-written, and demonstrate a deep understanding.	Reflection responses areclear and demonstrate agood understanding.	Reflection responses are somewhat clear but lack depth.	Reflection responses are unclear or demonstrate a lack of understanding.
Collaboration and Participation	Actively participates in group discussions and contributes meaningfully to the project.	Participates in group discussions and contributes to the project.	Participates in group discussions but with limited contributions.	Rarely participates in group discussions and contributes minimally.

Additional Criteria:

1. **Error Handling and Debugging:**
   - o **Excellent (4):** Effectively handles errors and debugging, ensuring the application runs smoothly.
   - o **Good (3):** Handles most errors and debugging, with minor issues.
   - o **Satisfactory (2):** Handles some errors and debugging, but there are noticeable issues. o **Needs Improvement (1):** Fails to handle errors and debugging, resulting in a non-functional application.
2. **Documentation and Comments:**
   - o **Excellent (4):** Code is well-documented with clear comments explaining the logic.
   - o **Good (3):** Code is mostly documented, with some comments explaining the logic.
   - o **Satisfactory (2):** Code has limited documentation and comments.
   - o **Needs Improvement (1):** Code lacks documentation and comments.

Scoring:

- **Excellent:** 36-40 points
- **Good:** 28-35 points
- **Satisfactory:** 20-27 points
- **Needs Improvement:** 12-19 points

Answer Sheets: