UNIVERSITY OF Baguio
SCHOOL OF INFORMATION TECHNOLOGY
General Luna Road, Baguio City
Telefax No.: (074) 442-3071 Website: www.ubaguio.edu E-mail Address: ub@ubaguio.edu
APPDEV1
INTRODUCTION TO APPLICATIONS DEVELOPMENT

Name:	1. SALANGO, TYRONE 2. IDALA, ASHLEY	Date 03-04-2025	Section: IDB2
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: ProfileComponent }
  ]},
  { 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:**
    - Functional navigation between Home, About, and Contact components.
    - Correct URL updates for each route.
2. **Child Routes:**
    - Nested routing within the About component.
    - Correct URL updates for nested routes.
3. **Lazy Loading Modules:**
    - Lazy-loaded Admin module.
    - Improved initial load time by loading the Admin module only when needed.
4. **Route Guards:**
    - Protected Admin route with guard logic.

Reflection Questions:

5. **Technical Challenges:**
    - What specific technical challenges did you encounter while implementing the routing methods, and how did you resolve them?
    - Were there any errors or bugs that were particularly difficult to debug? How did you approach solving them?
    - How do child routes and lazy loading improve the structure and performance of an Angular application?
    - What is the purpose of route guards, and how do they enhance the security of an application?
    - How did collaborating with your peers help you understand Angular routing better?

6. **Learning Process:**
    - How did this hands-on activity help you understand Angular routing better compared to just reading or watching tutorials?
    - What new concepts or techniques did you learn during this activity that you were not familiar with before?
7. **Collaboration:**
    - How did working in a group influence your understanding of Angular routing? Did you find it helpful to discuss and solve problems together?
    - What strategies did your group use to ensure effective collaboration and communication?
8. **Application Design:**
    - How did you decide on the structure and design of your application? What factors influenced your design choices?
    - How do you think the routing methods you implemented will impact the user experience of your application?
9. **Best Practices:**
    - What best practices did you follow while setting up the routes and organizing your code?
    - How did you ensure that your code is maintainable and scalable?
10. **Real-World Applications:**
    - Can you think of real-world applications or websites that use similar routing methods? How do they benefit from these methods?
    - How would you apply what you learned in this activity to a real-world project or job?

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 correctly implemented 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.
Route Guards 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.
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 are clear and demonstrate a good 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	Participates in group discussions and	Participates in group discussions but with limited contributions.	Rarely participates in group discussions and contributes minimally.

	contributes meaningfully to the project.	contributes to the project.		

Additional Criteria:

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

Reflection Questions:

1. **Technical Challenges:**
    - What specific technical challenges did you encounter while implementing the routing methods, and how did you resolve them?
        - **RouterLink Not Working:** Initially, navigation links weren't clickable due to missing imports.
          Solution: Imported RouterLink and RouterLinkActive in app.component.ts.
        - **Navbar Alignment Issues:** The navbar shifted slightly when switching between pages due to the scrollbar appearing only on some pages.
          Solution: Forced overflow-y: scroll; in CSS to maintain a consistent layout.
        - **Standalone Component Conflicts:** AdminComponent was a standalone component, which caused issues when adding it to admin.module.ts.
          Solution: Removed it from declarations and imported it instead.
        - **Child Routes Didn't Work at First:** Incorrect paths prevented navigation to subcomponents like /about/profile.
          Solution: Carefully reviewed and corrected the routing setup in app-routing.module.ts.
        - **Merge Conflicts Caused Navigation Issues:** Conflicting changes in routing files led to broken navigation.
          Solution: We manually reviewed, tested, and merged changes to ensure proper functionality.
        - **Lazy-Loaded Modules Not Working:** Some modules failed to load due to incorrect import paths and missing loadChildren configurations.
          Solution: Ensured correct module structure and proper imports in app-routing.module.ts.

    - Were there any errors or bugs that were particularly difficult to debug? How did you approach solving them?
        - **Admin Authentication Issues:** window.prompt() was blocked in some cases, preventing users from entering /admin.
          Solution: Used setTimeout() to delay execution, preventing security restrictions.
        - **Navigation to /admin Only Worked Sometimes:** After entering the admin page once, re-accessing it redirected straight to /home.
          Solution: Used router.events.subscribe() to reset authentication on navigation.

    - How do child routes and lazy loading improve the structure and performance of an Angular application?

      Child routes and lazy loading both played an important role in improving the structure and performance of our application. Child routes helped organize related components under a single path, making the application more modular and easier to maintain. Instead of defining separate routes for each component, we were able to nest them logically, such as */about/profile*. This improved readability and simplified navigation. Meanwhile, lazy loading helped optimize performance by ensuring that only the necessary modules were loaded when needed, rather than loading everything at once. This reduced the initial bundle size and made the application faster and more efficient, especially for users who did not need access to certain modules immediately.

    - What is the purpose of route guards, and how do they enhance the security of an application?

      Route guards are essential for enhancing security and controlling access within an application. In our case, implementing an AdminGuard ensured that only authenticated users could access the admin panel. This prevented unauthorized users from manually entering */admin* in the URL and gaining access. Additionally, route guards allowed us to redirect unauthorized users to */home*, providing a seamless and controlled user experience while ensuring that sensitive areas of the application remained protected. Without route guards, users could potentially access restricted areas without authentication, compromising security.

    - How did collaborating with your peers help you understand Angular routing better?

      Collaborating with team members was highly beneficial in understanding Angular routing. Debugging and discussing routing issues together allowed us to identify solutions much faster than working alone. When one person encountered an issue, others could provide insights based on their own experiences, leading to better problem-solving. Additionally, sharing different approaches to

implementing features such as lazy loading and route guards helped broaden our understanding of best practices. Through collaboration, we were able to refine our routing implementation, ensuring that it was both efficient and maintainable.

2. **Learning Process:**
    a. How did this hands-on activity help you understand Angular routing better compared to just reading or watching tutorials?

        While tutorials provide a solid foundation by explaining concepts, actually implementing Angular routing and facing real issues gave us a much deeper understanding. When working hands-on, we encountered unexpected errors that forced us to analyze documentation, experiment with different fixes, and apply problem-solving strategies. Unlike passive learning from videos or articles, debugging issues such as broken routes, lazy loading failures, and authentication problems required critical thinking and persistence. Through trial and error, we gained a more practical grasp of how routing functions in a real-world application.

    b. What new concepts or techniques did you learn during this activity that you were not familiar with before?

        During this activity, we learned several key concepts that we were not entirely familiar with before. One major takeaway was how lazy loading improves performance by reducing the initial load time, ensuring that modules are only loaded when needed. We also gained a better understanding of route guards and their role in restricting access to certain parts of the application, improving security. Additionally, we learned how module imports impact routing functionality—incorrect imports often led to broken routes, which we resolved by carefully structuring our project. Another interesting discovery was how CSS properties like *overflow-y: scroll;* can fix layout inconsistencies caused by scrollbars appearing dynamically, which affected navbar alignment across different pages.

3. **Collaboration:**
    a. How did working in a group influence your understanding of Angular routing? Did you find it helpful to discuss and solve problems together?

        Working in a group significantly improved our troubleshooting process and overall efficiency. Instead of spending hours debugging issues alone, we were able to discuss problems, compare different approaches, and collectively find the best solutions. Collaboration allowed us to distribute tasks effectively, ensuring that different aspects of routing—such as child routes, lazy loading, and guards—were implemented correctly. Additionally, reviewing each other's code helped us learn alternative ways to solve routing challenges, reinforcing best practices and improving our understanding of Angular's routing system.

    b. What strategies did your group use to ensure effective collaboration and communication?

        To ensure smooth teamwork, we adopted several collaboration strategies. We conducted regular code reviews to verify that our routing structures were correctly implemented and to catch any mistakes early. We also used clear commit messages to track changes in routing files, making it easier to debug issues when something broke. Pair programming was another helpful strategy, especially when debugging navigation problems—we worked in pairs to analyze errors and come up with efficient solutions. Additionally, we tested our application across multiple devices and browsers to confirm that our routing worked consistently, ensuring a better user experience.

4. **Application Design:**
    a. How did you decide on the structure and design of your application? What factors influenced your design choices?

        Our application was designed for modularity, scalability, and performance. We used feature modules to keep components and routing organized, while lazy loading reduced the initial load time by loading modules only when needed. Route guards were implemented to secure restricted areas like the admin panel.

        Since we love cats, we made them the theme of our website, influencing both visuals and content structure. We focused on making navigation intuitive and seamless, ensuring a smooth user experience. Our design choices were also inspired by real-world applications, such as e-commerce sites and admin dashboards, which use similar routing patterns for efficiency and security.

b. How do you think the routing methods you implemented will impact the user experience of your application?

Properly implemented routing greatly improved the overall user experience of our application. Users could navigate through different sections smoothly without unnecessary delays. Lazy loading ensured that pages loaded quickly by only fetching resources when needed, rather than forcing users to wait for all modules to load upfront. Additionally, using route guards enhanced security and prevented broken navigation paths, ensuring that restricted pages were only accessible to authorized users. These routing methods contributed to a more polished and professional user experience, making our application both faster and safer.

5. **Best Practices:**
   a. What best practices did you follow while setting up the routes and organizing your code?

   To ensure our application was built following best practices, we focused on keeping our routing setup clean and well-structured. We avoided hardcoding navigation paths in multiple places, instead relying on Angular's built-in RouterModule and dynamic navigation methods. We also followed Angular's style guide, which emphasizes modularization and reusability. Additionally, lazy loading was implemented to reduce load times, ensuring that users only downloaded the resources they actually needed. Route guards were used to enhance security, making sure that sensitive pages like the admin panel remained protected from unauthorized access.

   b. How did you ensure that your code is maintainable and scalable?

   To make our codebase easy to maintain and scalable for future development, we structured our app using feature modules. This approach allowed us to keep related components, services, and routes grouped together, making modifications easier. We also ensured that components were modular and reusable, reducing redundancy and improving code organization. Following a consistent folder structure further contributed to maintainability, making it easy for team members to navigate the project and understand how different parts of the application were connected.

6. **Real-World Applications:**
   a. Can you think of real-world applications or websites that use similar routing methods? How do they benefit from these methods?

   Many real-world applications utilize similar routing techniques to enhance performance and security. E-commerce platforms like Amazon leverage lazy loading to load product pages only when a user navigates to them, improving page speed and reducing unnecessary resource usage. Admin dashboards, such as those used in content management systems (CMS) or enterprise applications, commonly implement route guards to restrict access based on user roles. Social media platforms like Facebook use child routes to organize sections within a user's profile (e.g., /profile/settings). These routing methods are widely used across modern web applications to create efficient, secure, and scalable navigation systems.

   b. How would you apply what you learned in this activity to a real-world project or job?

   The concepts and techniques we learned during this activity will be highly valuable in real-world projects. Moving forward, we can build more secure applications by incorporating route guards to control access to specific features. Additionally, using lazy loading will improve performance in applications that handle large amounts of data, ensuring that users only load the content they need. Finally, proper route organization and modularization will help us design applications that are scalable, easy to maintain, and adaptable for future development, whether in a personal project or a professional setting.