Angular Project

Flight Plan Application

Paranthaman S

1948TM0494

Project Specifications to be covered

- 1. Components
- 2. Parent Child components
- 3. Structural and attribute directives
- 4. @Input
- 5. Basic Routing
- 6. Services
- 7. Observables
- 8. API Calls
- 9. Template and Reactive Forms
- 10. Pipes

Introduction

- The **Flight Planning System** is an Angular-based project that demonstrates key Angular concepts through a dynamic and interactive interface. This project covers **components**, including **parent-child communication**, to structure the UI efficiently. It utilizes **structural and attribute directives** to control DOM behavior and styling. Data is passed between components using **@Input**, ensuring seamless interaction. **Basic routing** is implemented to navigate between different views, while **services** and **observables** are used to manage and share data efficiently. The project also integrates **API calls** to fetch real-time data and utilizes **template-driven and reactive forms** for user input handling. Additionally, **pipes** are employed to format and transform data dynamically. Through this project, we aim to build a functional and scalable flight planning application that effectively demonstrates these Angular features.

Project Structure

- /src
- /app
- /book-tickets
- /available-tickets
- /home
- /menu-bar
- /models
- /offers
- /pipes
- /services
- /assets

Parent & Child Components

In my project, the **BookTickets** component serves as the **parent component**, where I allow users to select their departure and destination locations along with the travel date. Once the user provides this information and submits the form, I pass the selected details to the **AvailableTickets** component, which is my **child component**. The **AvailableTickets** component receives the data using **Input properties** and dynamically fetches and displays the relevant flights based on the user's selection. This setup helps me keep the responsibilities clear—**BookTickets** handles user input, while **AvailableTickets** focuses on showing the filtered flight options. Details of the flights are stored in an **api.json** file.

Routing, Services & API Calls

In my project, I have implemented **Basic Routing** to navigate between different pages, ensuring a smooth user experience using the Angular **RouterModule**. I use **Services** to centralize business logic, such as handling flight data and bookings, making my application more modular and reusable. To manage asynchronous data efficiently, I rely on **Observables**, which allow me to fetch and update data reactively, ensuring my components update dynamically when new data is available. Additionally, I perform **API Calls** using Angular's **HttpClientModule** to fetch flight details and offers from my api.json file, ensuring real-time data rendering in my application. This combination makes my project well-structured, efficient, and scalable. 🚀

JSON File

In my project, I use an api.json file to store flight and offer details, which I serve using **json-server**. This allows me to simulate a backend and fetch data dynamically. The **Flights** data is accessible at “http://localhost:4600/Flights”, where I store all available flight details, including destinations, prices, and images. Similarly, the **Offers** data is available at “http://localhost:4600/Offers”, containing various promotions and discounts. I use **API Calls** with Angular’s “HttpClientModule” to fetch this data, ensuring that my application always displays the latest flight availability and offers efficiently.

Related Command - json-server --port 4600 src/assets/api.json

Navigation

- Home Page
- │
- ├──> Book Tickets Page (Parent)
- │ ├──> Available Tickets Page (Child - Displays filtered flights)
- │
- ├──> Offers Page (Displays available offers)
- └──> Menu Bar (Accessible from all pages for easy navigation)