Module 7

What to learn

Dependency Injection
Creating Service
Hierarchical DI in Angular
Injection service into another service
Registering Value Data Service
Using Observable to pass values

Practice Exercise

Practice 1
Do the hands on from the following url https://angular.io/guide/dependency-injection https://angular.io/guide/dependency-injection-providers
Practice 2

User Authentication System

Use Case: Build a user authentication system where a **LoginService** handles login, and a **UserService** manages the user data after login.

Business Logic:

Implement a login form using Reactive Forms. Upon successful login, the **LoginService** will fetch user credentials and pass them to the **UserService** to fetch and store the user's profile.
If login fails, show an error message.

UI:

Login form with fields: **Username**, **Password**
Upon successful login, display the user name and email from the **UserService**.

Control IDs:

Login Form:

Username: loginUsername
Password: loginPassword
Submit Button: loginSubmit

User Profile:

User Name: userNameDisplay
User Email: userEmailDisplay

Practice 3

Product Catalog with Search Functionality

Use Case: You need to display a list of products. Create a **ProductService** that fetches the product list and a **SearchService** to filter products based on the search term.

Business Logic:

Implement a form to search for products by name.
Use **SearchService** to filter products based on user input.
The **ProductService** will fetch products when the component is loaded, and **SearchService** will filter them dynamically.

UI:

Product list displayed in a table.
Search bar at the top to filter products by name.

Control IDs:

Search Form:

Search Input: searchProduct
Search Button: searchButton

Product Table:

Product Name: productNameColumn
Product Price: productPriceColumn
Edit Button: editProductButton
Delete Button: deleteProductButton

Practice 4

Customer Feedback Form

Use Case: Build a customer feedback form where the **FeedbackService** saves and fetches customer feedback.

Business Logic:

Upon form submission, save the feedback using the **FeedbackService**.
The service should save feedback data and also fetch all submitted feedback to display in a table.

UI:

A form for customer name, feedback, and rating (1-5).
Display all feedback in a table with the ability to edit or delete.

Control IDs:

Feedback Form:

Customer Name: customerName
Feedback Message: feedbackMessage
Rating: feedbackRating

Submit Button: submitFeedbackButton

Feedback Table:

Customer Name: feedbackCustomerNameColumn

Feedback Message: feedbackMessageColumn

Rating: feedbackRatingColumn

Edit Button: editFeedbackButton

Delete Button: deleteFeedbackButton

Practice 5

Order Management System

Use Case: Manage orders for an e-commerce site. **OrderService** manages the order details and **PaymentService** handles payment processing.

Business Logic:

On order submission, use the **OrderService** to save the order, and **PaymentService** to handle payment status.

Display a list of orders with their status (pending, processed, completed).

UI:

Form to place an order (order details, payment status).

Display all orders in a table with their status.

Control IDs:

Order Form:

Order Item: orderItem

Quantity: orderQuantity

Payment Status: paymentStatus

Submit Button: placeOrderButton

Orders Table:

Order ID: orderIdColumn

Order Item: orderItemColumn

Quantity: orderQuantityColumn

Payment Status: orderPaymentStatusColumn

Edit Button: editOrderButton

Delete Button: deleteOrderButton

Practice 6

Employee Time Tracking

Use Case: Track employee working hours. Use **TimeTrackingService** to log time-in and time-out for employees.

Business Logic:

Create a form that records the **time-in** and **time-out** for employees.

TimeTrackingService will store the logs and calculate total hours worked.
Display all logs in a table with employee name and total hours worked.

UI:

A time-in/time-out form.
A table that displays employee hours worked.

Control IDs:

Time Tracking Form:
Employee Name: employeeName
Time In: timeIn
Time Out: timeOut
Submit Button: submitTimeLogButton

Time Logs Table:
Employee Name: employeeNameColumn
Time In: timeInColumn
Time Out: timeOutColumn
Total Hours Worked: totalHoursWorkedColumn

Practice 7

Observables: User Dashboard with Real-time Data

Use Case: You are building a user dashboard that fetches data (e.g., user activities, notifications) in real-time using Observables.

Business Logic:

Create a **UserService** that returns user activity data and notifications using an **Observable**.
The component will subscribe to the Observable and update the dashboard when new data arrives.
Implement a **refresh button** to manually trigger the refresh of data.

UI:

A **User Dashboard** with sections for user activity and notifications.
Refresh Button to fetch new data.

Control IDs:

Dashboard Sections:
Activity List: activityListSection
Notifications List: notificationsListSection
Refresh Button: refreshButton

Data Display:
Activity Item: activityItem

Notification Item: notificationItem

Practice 8

Observables: Live Search for Products

Use Case: Implement a live search for products that filters results as the user types in the search box, using an **Observable** for real-time data.

Business Logic:

Create a **ProductService** that returns an Observable of product data.

The component subscribes to the search Observable, triggering the **ProductService** each time the user types a new search term.

Display the filtered results dynamically.

UI:

Search Box to input the search term.

Product List that updates in real-time as the search term is entered.

Control IDs:

Search Box:

Search Input: searchInput

Search Button: searchButton

Product List:

Product Name: productNameColumn

Product Price: productPriceColumn

Add to Cart Button: addToCartButton

Practice 9

Observables: Fetching User Details with RxJS Operators

Use Case: Fetch user details from a server using **Observables** and **RxJS operators** like **map** and **catchError** to process data and handle errors.

Business Logic:

Create a **UserService** that returns user data from an API endpoint.

Use RxJS operators like **map** to transform data and **catchError** to handle errors gracefully.

Display the user data or an error message in the component.

UI:

Display the user's name, email, and other details if the fetch is successful.

Show an error message if the request fails.

Control IDs:

User Data:

User Name: userName
User Email: userEmail
User Address: userAddress

Error Message:

Error Message: errorMessage

Practice 10

Observables: Stock Price Tracker

Use Case: Implement a stock price tracker that updates the price in real-time using **Observables**.

Business Logic:

Create a **StockService** that emits stock price updates at regular intervals (using **setInterval** or a similar approach) as an **Observable**.
Display the current price of a stock, and allow the user to subscribe to the Observable to receive price updates.

UI:

Stock Price Display with current price.
Subscribe Button to start receiving updates.
Unsubscribe Button to stop receiving updates.

Control IDs:

Stock Price Display:

Stock Price: stockPriceDisplay

Subscription Buttons:

Subscribe Button: subscribeButton
Unsubscribe Button: unsubscribeButton

Practice 11

Observables: Real-Time Chat Application

Use Case: Implement a real-time chat application where messages are sent and received using **Observables**.

Business Logic:

Create a **ChatService** that emits new messages via an **Observable** whenever a new message is sent.
Use **RxJS operators** to filter, map, or combine messages.
The component subscribes to the Observable and updates the chat window with new messages.

UI:

Chat Input Box to type messages.
Message List that updates in real-time.

Send Button to send messages.

Control IDs:

Chat Window:

Message Input: messageInput
Send Button: sendMessageButton
Message List: messageList
Individual Message: messageItem

Practice 12

Hierarchical DI: Managing User Preferences Across Multiple Components

Use Case: You need to manage user preferences (theme, language) across multiple components using Angular's **Hierarchical DI**. A shared service should be injected into different parts of the application but should maintain separate states based on the component hierarchy.

Business Logic:

Create a service that manages user preferences like theme and language.
In the root component, the service should hold the default preferences.
For a child component, inject the same service but allow it to override the preferences for that component only.

UI:

Main App Component that displays the current theme and language.
Child Component where users can change their theme and language independently.

Control IDs:

Main App:

Current Theme: currentTheme
Current Language: currentLanguage

Child Component:

Theme Dropdown: themeDropdown
Language Dropdown: languageDropdown

Assignment Exercise

Assignment 1

Create StudentService which will contains operation for crud operation using student type Array.

Assignment 2

Create a Log service which will be injected by StudentService on every crud operation will console list message in console. Inject student service Student Component and StudentList Component

Online Reference

No online Reference

Introduction

the basics

course project-basics

debugging

components & databinding deep dive

course project – components & databinding

directives deep dive

Using Services & Dependency Injection

Course Project – Services & Dependency Injection

Changing Pages with Routing

Course Project – Routing

Handling Forms in Angular Apps

Course Project-Forms

Using Pipes to Transform Output

Making Http Requests