Module 4

What to learn

Form
Reactive Form
Form Group
Form Control

Practice Exercise

Practice 1 (65cdfdab7ba36e06448762a8)
Create A Reactive Form and try to update partially some field using the patch value

User Form: control id-> #userForm
First Name: control id-> #firstName
Email: control id-> #email
Submit button: control id-> #submit
Patch value button: control id-> #patch-value
User List: control id-> #user-data

*Patch value should be = "updatedemail@gmail.com"

Practice 2 (6744120f9a32ad5122546106)

Basic Input Field Validation

Use Case: Implement a reactive form with an email field.

Functional Flow: The email field should accept only valid email addresses.
UI: Label: *Email*, Input ID: email-input, Error Message: *Invalid email address*, Error Control ID: email-error.

Practice 3 (6744120f9a32ad5122546107)
Basic Input Field Validation

Use Case: Add a required full name field with a minimum of 3 characters.

Functional Flow: Display an error when the input is empty or less than 3 characters.
UI: Label: *Full Name*, Input ID: fullname-input, Error Message: *Full name must be at least 3 characters*, Error Control ID: fullname-error.

Practice 4 (6744120f9a32ad5122546108)

Use Case: Include a numeric-only phone number field.

Functional Flow: Allow only numbers with 10 digits.
UI: Label: *Phone Number*, Input ID: phone-input, Error Message: *Phone number must be 10 digits*, Error Control ID: phone-error.

Practice 5 (6744120f9a32ad5122546109)

Use Case: Add a password field with a minimum of 8 characters, including a number and a special character.

Functional Flow: Validate format and show specific messages for missing criteria.
UI: Label: *Password*, Input ID: password-input, Error Message: *Password must contain at least 8 characters, a number, and a special character*, Error Control ID: password-error.

Practice 6 (6744120f9a32ad512254610a)

Use Case: Create a confirm password field that matches the password field.

Functional Flow: Validate password match.

UI: Label: *Confirm Password*, Input ID: confirm-password-input, Error Message: *Passwords do not match*, Error Control ID: confirm-password-error.

Practice 7 (67441336b7decf5124191945)

Use Case: Add a date picker for date of birth with validation for age (minimum 18 years).

Functional Flow: Validate that the date entered corresponds to an age of 18 or older.

UI: Label: *Date of Birth*, Input ID: dob-input, Error Message: *You must be at least 18 years old*, Error Control ID: dob-error.

Practice 8 (67441336b7decf5124191946)

Use Case: Include a postal code field with a regex pattern.

Functional Flow: Accept formats like 12345 or 12345-6789.

UI: Label: *Postal Code*, Input ID: postal-input, Error Message: *Invalid postal code format*, Error Control ID: postal-error.

Practice 9 (67441336b7decf5124191947)

Use Case: Add a conditional validation: If a checkbox is checked, a related input field becomes required.

Functional Flow: Example: If "Subscribe to newsletter" is checked, email must be filled.

UI: Checkbox ID: subscribe-checkbox, Related Input ID: email-input, Error Message: *Email is required for subscription*, Error Control ID: subscribe-error.

Practice 10 (67441336b7decf5124191948)

Use Case: Create a field for usernames that cannot contain "admin".

Functional Flow: Implement custom validation logic to reject "admin" as a username.

UI: Label: *Username*, Input ID: username-input, Error Message: *Username cannot contain "admin"*, Error Control ID: username-error.

Practice 11 (67441336b7decf5124191949)

Use Case: Validate a field for a specific range (e.g., age between 1 and 100).

Functional Flow: Allow only values within the range.

UI: Label: *Age*, Input ID: age-input, Error Message: *Age must be between 1 and 100*, Error Control ID: age-error.

Practice 12 (67441336b7decf512419194a)

Use Case: Create an address form group with required fields (Street, City, ZIP).

Functional Flow: Ensure all fields are validated as a group.

UI: Inputs: street-input, city-input, zip-input, Error Message: *All address fields are required*, Error Control ID: address-error.

Practice 13 (674417e04df0ff511ced9e3a)

Use Case: Implement validation to ensure at least one field in a group (e.g., phone or email) is filled.

Functional Flow: Display an error if both fields are empty.

UI: Group ID: contact-group, Error Message: *At least one contact method is required*, Error Control ID: contact-error.

Practice 14 (674417e04df0ff511ced9e3b)

Use Case: Add a dropdown with dependent validation for another field.

Functional Flow: If "Other" is selected in the dropdown, an input field becomes required.

UI: Dropdown ID: category-select, Dependent Input ID: other-input, Error Message: *Please specify "Other" value*, Error Control ID: other-error.

Practice 15 (674417e04df0ff511ced9e3c)

Use Case: Implement a checkbox for terms and conditions with required validation.

Functional Flow: Show error if unchecked.
UI: Checkbox ID: terms-checkbox, Error Message: *You must agree to the terms and conditions*, Error Control ID: terms-error.

Practice 16 (674417e04df0ff511ced9e3d)

Use Case: Validate a field for only uppercase input.
Functional Flow: Allow only uppercase characters.
UI: Label: *Code*, Input ID: code-input, Error Message: *Only uppercase letters are allowed*, Error Control ID: code-error.

Practice 17 (674417e04df0ff511ced9e3e)

Use Case: Add a file upload field that accepts only specific file types (e.g., .pdf or .docx).

Functional Flow: Show an error for unsupported types.
UI: Input ID: file-upload-input, Error Message: *Only PDF and DOCX files are allowed*, Error Control ID: file-error.

Practice 18 (674417e04df0ff511ced9e3f)

Use Case: Implement live validation feedback (e.g., as the user types).

Functional Flow: Display errors instantly for invalid inputs.
UI: Input ID: live-feedback-input, Error Message: *Invalid value*, Error Control ID: live-error.

Practice 19 (674417e04df0ff511ced9e40)

Use Case: Validate a multi-field relationship (e.g., end date must be after start date).

Functional Flow: Compare two date fields and validate accordingly.
UI: Start Date ID: start-date-input, End Date ID: end-date-input, Error Message: *End date must be after start date*, Error Control ID: date-error.

Practice 20

Dynamic Contact Information (Phone Numbers)

Business Logic:

The user can add multiple phone numbers.
At least one phone number must be entered, and it must be a valid 10-digit number.
Each phone number should be validated as the user types.

Functional Flow:

The user can click the "Add Phone Number" button to add a new phone number field.
The "Remove" button should remove the corresponding phone number.
Validation should trigger for each phone number when the user tries to submit the form.

UI and Control IDs:

Label: Contact Phone Numbers
Field ID: phone-array
Input ID: phone-input-{i} (where {i} is the dynamic index)
Error Message: "Phone number must be 10 digits"
Error Control ID: phone-error-{i}
Add Button ID: add-phone-button
Remove Button ID: remove-phone-button-{i}
Submit Button ID: submit-button

Validation:

Rule: At least one phone number is required.
Pattern: Only 10-digit numeric values are allowed.

Practice 21

Question 2: Dynamic Address List (Multiple Addresses)

Business Logic:

The user can add multiple addresses (Home, Office, etc.).
Each address should have a street, city, and zip code field.
Zip code must follow the pattern of 5 digits or 5-4 digits (e.g., 12345 or 12345-6789).

Functional Flow:

The user can click the "Add Address" button to add a new address group.
The "Remove" button should remove the corresponding address group.
The form should validate that all address fields (street, city, zip code) are filled before submitting.

UI and Control IDs:

Label: Addresses
Field ID: address-array
Street Input ID: street-input-{i}
City Input ID: city-input-{i}
Zip Input ID: zip-input-{i}
Error Message: "Zip code must be in the format 12345 or 12345-6789"
Error Control ID: zip-error-{i}
Add Button ID: add-address-button
Remove Button ID: remove-address-button-{i}
Submit Button ID: submit-button

Validation:

Rule: Zip code must be valid.
Pattern: Zip code must match ^\d{5}(-\d{4})?$.

Practice 22

Dynamic Skill Set (Multiple Skills)

Business Logic:

The user can add multiple skills.
Each skill should have a name (required) and a skill level (1-5).
Skill level must be a numeric value between 1 and 5.

Functional Flow:

The user can click the "Add Skill" button to add a new skill.
The "Remove" button should remove the corresponding skill.
The form should ensure that all skill levels are valid when submitting.

UI and Control IDs:

Label: Skills
Field ID: skills-array
Skill Name Input ID: skill-name-input-{i}
Skill Level Input ID: skill-level-input-{i}
Error Message: "Skill level must be between 1 and 5"
Error Control ID: skill-level-error-{i}
Add Button ID: add-skill-button
Remove Button ID: remove-skill-button-{i}
Submit Button ID: submit-button

Validation:

Rule: Skill level must be between 1 and 5.
Range: Skill level should be a number between 1 and 5.

Practice 23

Dynamic Payment Methods (Multiple Cards)

Business Logic:

The user can add multiple credit card details.
Each card should have a card number (16 digits), expiry date, and CVV.
The card number should be validated for the correct length and format (16 digits).
The expiry date should not be in the past.

Functional Flow:

The user can click the "Add Card" button to add a new card.
The "Remove" button should remove the corresponding card.
The form should validate each card's information (card number, expiry date, CVV) when the user tries to submit.

UI and Control IDs:

Label: Credit Cards
Field ID: cards-array
Card Number Input ID: card-number-input-{i}
Expiry Date Input ID: expiry-date-input-{i}
CVV Input ID: cvv-input-{i}
Error Message: "Invalid card number (16 digits required)" or "Expiry date must be in the future"
Error Control ID: card-number-error-{i}, expiry-date-error-{i}, cvv-error-{i}
Add Button ID: add-card-button
Remove Button ID: remove-card-button-{i}
Submit Button ID: submit-button

Validation:

Card Number: Must be exactly 16 digits.
Expiry Date: Must be a valid date in the future.
CVV: Must be 3 digits.

Practice 24

Dynamic Emergency Contacts (Multiple Contacts)

Business Logic:

The user can add multiple emergency contacts.
Each contact must have a name, phone number, and relationship.
The phone number should be validated as a 10-digit number.
The relationship field should have a set of predefined options (e.g., "Mother", "Father", "Friend").

Functional Flow:

The user can click the "Add Contact" button to add a new contact.
The "Remove" button should remove the corresponding contact.
The form should validate that all fields (name, phone, relationship) are filled before submission.

UI and Control IDs:

Label: Emergency Contacts
Field ID: emergency-contacts-array
Name Input ID: contact-name-input-{i}
Phone Input ID: contact-phone-input-{i}
Relationship Input ID: contact-relationship-input-{i}
Error Message: "Phone number must be 10 digits"
Error Control ID: contact-phone-error-{i}
Add Button ID: add-contact-button
Remove Button ID: remove-contact-button-{i}

Submit Button ID: submit-button

Validation:

Phone Number: Must be 10 digits.
Relationship: Must be a valid selection from the predefined list.

Assignment Exercise

Assignment 1 (65cdfdab7ba36e06448762a6)
1. Create a Reactive form with validations which are provided Below

Student Form: control id -> #StudentForm,
Username: control id -> #email,
Password: control id -> #password,
Date of Birth: control id -> #dob,
Phone: control id -> #phone,
Address: control id -> #address

student List: control id -> #studentTable

submit: button with type submit

validations :

1. Required validations -> {control name} is required.
   Example: Address is required.

2. Other validations ->
   -Username must be at least 3 characters.,
   -Invalid email format.,
   -Password must be at least 8 characters.,
   -Date of Birth is required.,
   -Phone must be a 10-digit number.

After submitting details Forms info should be added in StudentList Array and displayed in the Table Form

Assignment 2 (65cdfdab7ba36e06448762a7)

CRUD Operations for Managing Employee Records

Objective: Create a form for adding and editing employee records with full CRUD functionality.

You are tasked with building a simple Employee Management system that allows you to manage employee records effectively. The system should provide the following functionalities:

Add Employee: Use a Reactive Form to add new employees, including their name, email, and department.
Edit Employee: When an employee record is clicked, the form should be populated with the selected employee's details using patchValue.
Delete Employee: Ability to remove an employee from the list.
Display Employees: Show the list of employees in a tabular format.

Your Task

Create the Reactive Form with the following fields:

Name: Control ID - employeeName
Email: Control ID - employeeEmail
Department: Control ID - employeeDepartment

Implement the following:

Use patchValue to populate the form when editing an employee.
Implement logic to add, edit, and delete employee records.
Display the list of employees in a table format with actions:
Edit Button Control ID - editEmployeeButton-<id> (replace <id> with the actual employee ID)
Delete Button Control ID - deleteEmployeeButton-<id> (replace <id> with the actual employee ID)
Dynamic ID for each Employee Record: Each record should have a unique identifier in the format employee-<id>.

UI Control IDs

Field	Control ID	Error Control ID	Validation Rule	Er
Employee Name	employeeName	err-employee-name	Required	"Er
Employee Email	employeeEmail	err-employee-email	Required, Email format	"V
Employee Department	employeeDepartment	err-employee-department	Required	"D
Add Employee Button	addEmployeeButton		N/A	N/
Success Message	success-add-employee		N/A	"Er
Update Success Message	success-update-employee		N/A	"Er
Delete Success Message	success-delete-employee		N/A	"Er
Employee Record ID	employee-<id>		N/A	N/

Functional Flow

Adding Employee:

Input values for Employee Name, Email, and Department in the respective fields.
Click the "Add" button identified by addEmployeeButton.
Validate all fields before submission. If validation passes, add the employee record and display the success message in success-add-employee. If validation fails, display error messages using their respective Error Control IDs.

Editing Employee:

Click on an employee entry to edit.
Use patchValue to populate the fields with the selected employee's details.
Make any necessary edits and click the "Update" button identified by updateEmployeeButton-<id> (where <id> represents the actual employee ID).
Validate inputs and update the employee record, displaying the success message in success-update-employee.

Deleting Employee:

Click the "Delete" button identified by deleteEmployeeButton-<id> (where <id> represents the actual employee ID) next to an employee record.
After deleting, display the success message in success-delete-employee.

Display Employee List:

Render the list of employees in a table format where each employee row has a unique identifier such as employee-<id> (replace <id> with the actual employee ID).

Validation Elements

Validate that each field is filled out before submission.
Display error messages in the specified Error Control IDs (e.g., err-employee-name, err-employee-email, err-employee-department) depending on the specific field that fails validation.

Business Logic

Employee IDs should be auto-incremented starting from 1.
Email should follow proper email formatting for successful validation.
Ensure that deletion requires confirmation to prevent accidental removals.

Selectors for Test Cases

The following control IDs are necessary for automated test case generation:

Control ID	Purpose
employeeName	Input for employee name.
employeeEmail	Input for employee email.
employeeDepartment	Input for employee department.
addEmployeeButton	Button to add a new employee.
success-add-employee	Used for success message after adding employee.
success-update-employee	Used for success message after updating employee.
success-delete-employee	Used for success message after deleting employee.
err-employee-name	Shows error for employee name input.
err-employee-email	Shows error for employee email input.
err-employee-department	Shows error for department selection.
employee-<id>	Unique ID for each employee record in the list.
editEmployeeButton-<id>	Dynamic edit button ID for each employee.
updateEmployeeButton-<id>	Button that updates the information of employee.
deleteEmployeeButton-<id>	Dynamic delete button ID for each employee.

By following these detailed requirements, you will ensure the successful implementation and testing of the Employee Management System using Angular.

Assignment 3

CRUD Operations for Managing Products

Objective: Implement a product management system where users can add, edit, and delete products.

Question:
You need to implement a **Product Management** system where users can:

Add a new product with name, price, description.
Edit a product's details by selecting it from the list, which will populate the form using patchValue.
Delete a product from the list.
Display products in a table.
Your task is to:
Build a form for adding products with the following fields:
- Product Name (productName)
- Price (productPrice)
- Description (productDescription)

Use patchValue to pre-fill the form when editing a product.

Implement add, edit, and delete actions for products.
Display the list of products in a table with columns: Product Name, Price, Description, and Actions (Edit, Delete).

UI Control IDs:

Form Inputs:
- Product Name: productName
- Price: productPrice
- Description: productDescription

Table Columns:
- Product Name: productNameColumn
- Price: productPriceColumn
- Description: productDescriptionColumn
- Edit Button: editProductButton
- Delete Button: deleteProductButton

Assignment 4

CRUD Operations for Managing Customer Feedback

Objective: Build a feedback form that allows users to submit, edit, and delete customer feedback.

Question:
Create a **Customer Feedback Management** system where users can:

Submit a new feedback entry with customer name, feedback message, and rating (1-5).
Edit a feedback entry by selecting it from the list and using patchValue to load its details into the form.
Delete a feedback entry.
Display feedback entries in a table.
Your task is to:
Implement a form with the following fields:
- Customer Name (customerName)
- Feedback Message (feedbackMessage)
- Rating (feedbackRating)

Use patchValue to load the selected feedback's details when editing.
Implement actions for submitting, editing, and deleting feedback.
Display the list of feedback in a table with columns: Customer Name, Message, Rating, and Actions (Edit, Delete).

UI Control IDs:

Form Inputs:
- Customer Name: customerName
- Feedback Message: feedbackMessage
- Rating: feedbackRating

Table Columns:
- Customer Name: feedbackCustomerNameColumn
- Feedback Message: feedbackMessageColumn
- Rating: feedbackRatingColumn
- Edit Button: editFeedbackButton
- Delete Button: deleteFeedbackButton

Assignment 5

Student Record Management with Reactive Form

Objective: Build a student record management system where users can add, edit, and delete student records.

Question:
You need to implement a **Student Management** system with the following features:

- Add new student records with name, email, course, and grade.
- Edit an existing student record by selecting it from the list. When selected, the student's data should be patched into the form using patchValue.
- Delete a student record.
- Display the students in a table format.
- Your task is to:
- Create a Reactive Form with the following fields:
  - Name (studentName)
  - Email (studentEmail)
  - Course (studentCourse)
  - Grade (studentGrade)
- Use patchValue to populate the form when editing.
- Implement add, edit, and delete functionality for student records.
- Display the student records in a table with columns: Name, Email, Course, Grade, and Actions (Edit, Delete).

UI Control IDs:

- Form Inputs:
  - Name: studentName
  - Email: studentEmail
  - Course: studentCourse
  - Grade: studentGrade
- Table Columns:
  - Name: studentNameColumn
  - Email: studentEmailColumn
  - Course: studentCourseColumn
  - Grade: studentGradeColumn
  - Edit Button: editStudentButton
  - Delete Button: deleteStudentButton

Assignment 6

CRUD Operations for Task Management System

Objective: Implement a simple task management system where users can add, edit, and delete tasks.

Question:
Create a **Task Management** system where users can:

- Add tasks with a title, description, and due date.
- Edit a task by selecting it from the list, which will populate the form using patchValue.
- Delete a task from the list.
- Display tasks in a table format.
- Your task is to:
- Build a form to create tasks with the following fields:
  - Title (taskTitle)
  - Description (taskDescription)
  - Due Date (taskDueDate)
- Use patchValue to auto-fill the form with the selected task's details when editing.
- Implement actions for adding, editing, and deleting tasks.
- Display tasks in a table with columns: Title, Description, Due Date, and Actions (Edit, Delete).

UI Control IDs:

Form Inputs:

- Title: taskTitle
- Description: taskDescription
- Due Date: taskDueDate

Table Columns:

- Title: taskTitleColumn
- Description: taskDescriptionColumn
- Due Date: taskDueDateColumn
- Edit Button: editTaskButton
- Delete Button: deleteTaskButton

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