

# Angular Technical Test

GitHub Repository: <https://github.com/duvan-23/frontend-code-challenge>

This document provides a detailed overview of the Angular-based technical test project. It explains the structure, logic, and implementation decisions taken to meet the test requirements. The application demonstrates core Angular concepts such as reactive forms, component communication, signals, file handling, and client-side CSV parsing, while adhering to clean code principles and a modern, responsive UI using Tailwind CSS.

## Project Structure

The project follows a modular and scalable structure that separates concerns across feature and utility layers, supporting maintainability, reusability, and testability. Pages like *home* and *summary* handle specific business logic, while shared components (e.g., *DynamicForm*, *DynamicCsvTable*) are reusable across the app. Clear folders for models, pipes, and directives keep logic organized and easy to manage. The structure follows Angular best practices, making it easy to extend features without impacting core functionality. Environment files also help isolate sensitive data like encryption keys, improving security and configuration.

```
src/
├── app/
│   ├── core/           # Core module
│   ├── layout/         # Layout module
│   ├── pages/          # Pages module
│   ├── shared/         # Shared module
│   │   ├── components/ # Components
│   │   ├── directives/ # Directives
│   │   ├── models/     # Models
│   │   ├── pipes/      # Pipes
│   │   ├── services/   # Services
│   │   └── utils/      # Utilities
└── environments/
    ├── environment.ts
    └── environment.prod.ts
```

## Route Guards

The application uses two `CanActivate` guards — `sessionFormGuard` and `summaryGuard` — to manage access based on session state and enforce proper navigation flow:

- **summaryGuard**: Applied to the `/summary` route. It allows access **only if** valid form data exists in `sessionStorage`. If the session is missing or invalid, the user is redirected back to the home page (`/`), preventing direct access to the summary without submitting the form first.
- **sessionFormGuard**: Applied to the root route (`/`). If valid form data already exists in `sessionStorage`, the user is **redirected to `/summary`** instead of being shown the form again. This avoids overwriting existing data and improves the user experience.

These guards provide consistent user flow and ensure that access to pages is controlled based on session state.

## Layout & Routing Structure

The application uses a `Layout` component to provide a consistent UI structure across routes.

Routing is configured to nest the `Home` and `Summary` pages under the `Layout` component, ensuring the layout is shared across routes:

```
export const routes: Routes = [
  {
    path: '',
    component: Layout,
    children: [
      { path: '', component: Home, canActivate: [sessionFormGuard] },
      { path: 'summary', component: Summary, canActivate: [summaryGuard] },
    ],
  },
  { path: '**', redirectTo: '' },
];
```

- The **wildcard route `**`** acts as a fallback: if the user navigates to an undefined route, they are automatically redirected to the Home page (`/`). This improves user experience and prevents broken navigation.

## Home Page & DynamicForm Component

The **Home** page serves as the main entry point for the application and includes a shared, reusable component called DynamicForm. This shared component is responsible for building a reactive form dynamically based on a configuration input.

---

### *Input: Dynamic Field Configuration*

The DynamicForm component receives its structure through an @Input() fields, which is an array of objects following the DynamicField interface:

```
export interface DynamicField {  
  name: string;  
  label: string;  
  type: DynamicFieldType;  
  options?: { label: string; value: string }[];  
  validators?: ValidatorFn[];  
  required?: boolean;  
  patternHint?: string;  
  fileAccept?: string;  
  autocomplete?: string;  
  defaultValue: any;  
}
```

Supported field types (DynamicFieldType) include:

```
'type' = 'text' | 'email' | 'password' | 'select' | 'file';
```

This flexibility allows the form to be built dynamically with custom labels, validation rules, default values, input types, and accepted file formats.

### *Output: Form Submission*

The DynamicForm emits a value through an @Output() formSubmitted event once the form passes validation and is submitted.

---

### *Validation & Error Handling*

- Uses Angular **FormBuilder** to create reactive controls.
- Built-in support for **validators** like required, minLength, pattern, email, etc.

- **Real-time validation** using `form.valueChanges` with `debounceTime` to avoid immediate validation while typing.
  - Displays individual **inline error messages** under each input.
  - Shows a **cumulative summary of all form errors** at the top when attempting to submit an invalid form.
- 

### *Secure Submission Logic*

Upon successful submission:

- The password is encrypted (if a password is present) using a utility (`Crypto.encrypt`) for added security.
  - The resulting data is stored temporarily in `sessionStorage`.
  - Navigation is then triggered to the summary view.
- 

### *Clear Button Behavior*

- The **Clear** button is only active if the form has been modified.
  - When clicked, it shows a confirmation dialog using **SweetAlert (Swal)** to prevent accidental resets.
  - If confirmed, it resets the form to its default values and clears validation states, including uploaded file metadata.
- 

### *File Upload Support*

- Supports file input types via the `type: 'file'` field option.
  - You can pass accepted file types via the `fileAccept` property (e.g., `'csv'`).
  - Uploaded file data is handled and previewed in later steps (e.g., the Summary page).
- 

This modular, declarative, and extensible form design helps maintain clean code, enhances reusability, and satisfies all business requirements from the technical test prompt.

## Summary Page & DynamicCsvTable Component

After submitting the form on the Home page, users are navigated to the **Summary** page. This view presents a structured summary of all submitted data, including parsed CSV content, and offers editing and logout functionality.

---

### *Personal and Subscription Info*

- The **first section** displays all form data **except the CSV content**.
- If a CSV file was uploaded, its **file name** is shown clearly.
- If no CSV file was included (it is optional), a fallback message appears:

*"No CSV File Loaded – Please upload a CSV file to see the data displayed in this section."*

### CSV Table View

This section uses the shared DynamicCsvTable component.

#### Component inputs:

```
@Input() dataBase64: string = '';  
@Input() title: string = '';
```

#### Functionality:

- The component decodes and parses the base64 CSV data.
- Displays a **dynamic table** with:
  - **Pagination** (10 rows per page for readability)
  - **Search input** to filter rows
  - **Horizontal scrolling** to handle many columns gracefully

If CSV data is not available, the component hides the table and shows an instructional placeholder.

### *Editing Form Data via Dialog*

Next to the page title, there's a **pencil icon button** that opens a MatDialog containing the same DynamicForm component used on the Home page.

#### Key features of the dialog:

- Pre-fills the form using the previously saved data (via defaultValue)
- Accepts an edit flag to toggle button behavior (e.g., shows **Revert** instead of **Clear**)
- On submit:

- Updates the session-stored data
- Parses and displays any newly uploaded CSV
- Retains error validation and debounce behavior from the Home form

### *Logout Option*

In the Summary page header, a **Logout button** is available.

- Clicking it clears the session storage
- Redirects the user to the Home page to start fresh

---

This setup ensures a seamless and reactive user experience, keeping code DRY by reusing shared components (DynamicForm, DynamicCsvTable) while handling complex state like editing, persistence, and conditional CSV rendering in a clean, modular way.

### Custom Directive: HighlightInvalid

To improve user feedback and form usability, the application includes a custom directive called HighlightInvalid, which visually highlights invalid form fields in real-time.

---

#### *How It Works*

- The directive listens to **form control status changes**.
- When an input becomes **invalid** (e.g., fails validation) and is either **touched or dirty**, it:
  - Applies a red border to the input field.
  - Removes any box shadow to ensure the error state is clearly visible.
- Once the input becomes **valid**, the highlighting styles are removed.

### Custom Pipe: formErrorMessage

This application uses a custom pipe called formErrorMessage to dynamically generate human-readable error messages for form fields in the UI.

#### *How It Works*

- It accepts a **form control**, an optional **field name**, a **custom message** for pattern validation, and optionally a set of **override errors**.
- Based on the validation errors (e.g., required, email, minlength, pattern), it returns a friendly and specific error message for each case.
- If no known error is matched, a generic invalid message is returned.

## Shared Service: SessionStorage

This service provides a typed and safe abstraction for interacting with the browser's sessionStorage. It is used throughout the application to **persist form data temporarily** between navigation events (e.g., from the home page to the summary view).

```
setItem<T>(key: string, value: T): void {
  try {
    sessionStorage.setItem(key, JSON.stringify(value));
  } catch (error) {
    console.error(`Error saving to sessionStorage: ${key}`, error);
  }
}

getItem<T>(key: string): T | null {
  try {
    const item = sessionStorage.getItem(key);
    return item ? (JSON.parse(item) as T) : null;
  } catch (error) {
    console.error(`Error reading from sessionStorage: ${key}`, error);
    return null;
  }
}
```

## Utility Helpers (/shared/utls)

The utls folder contains specialized, reusable helper functions that encapsulate common logic used across the application:

- **Crypto**: Provides encryption and decryption logic to securely store sensitive data (e.g., passwords) in sessionStorage.
- **FileConvert**: Converts uploaded files to base64 format, allowing them to be stored as string values in memory or sessionStorage.
- **ParseData**: Uses papaparse to parse CSV files (from base64) directly on the frontend without backend dependencies.

These utilities keep the main component logic clean and focused, while promoting reusability and separation of concerns.

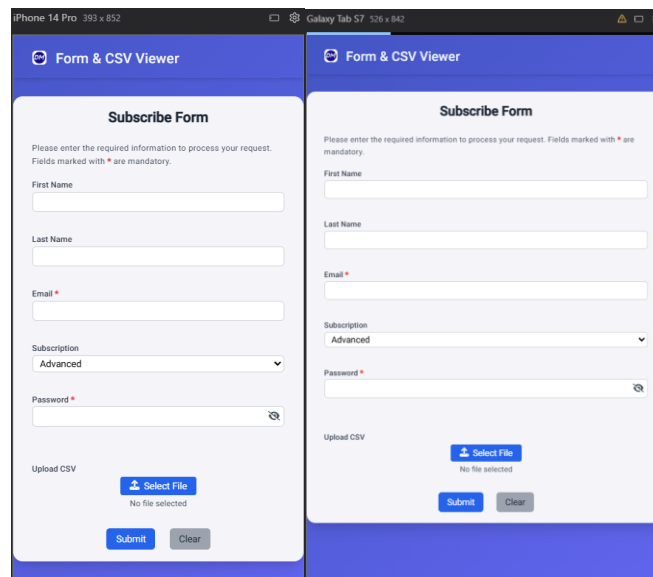
## Responsive Design

This application was built with **Tailwind CSS**, embracing a **mobile-first approach**. Utility classes like flex-wrap, min-h-screen, text-wrap, and responsive padding/margin utilities were used to ensure the UI adapts seamlessly across various screen sizes.

Below are screenshots showing how the **Home** and **Summary** pages adjust across devices:

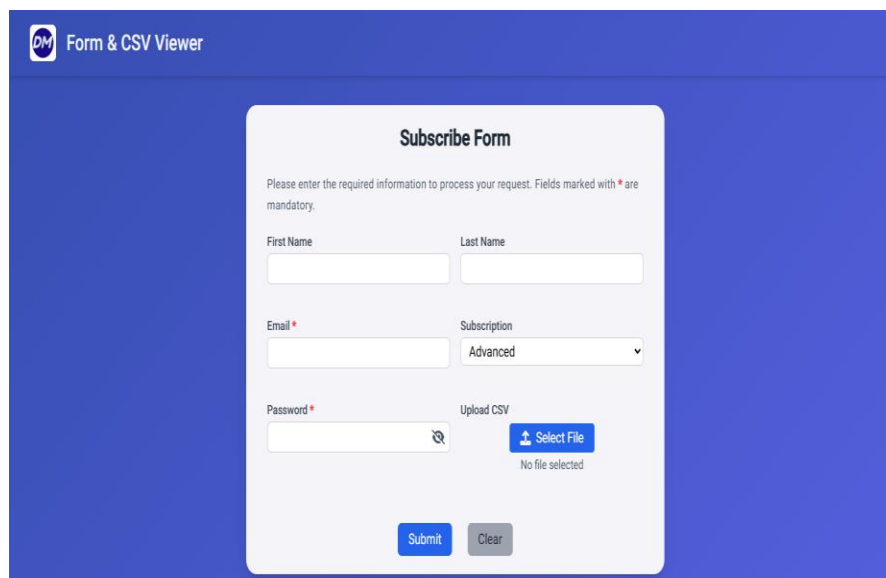
### Home Page – Responsive Views

- Mobile View - Tablet View



The image shows two side-by-side screenshots of the 'Form & CSV Viewer' application. The left screenshot is for an iPhone 14 Pro (393 x 852) and the right is for a Galaxy Tab S7 (526 x 842). Both screens display a 'Subscribe Form' with the following fields: First Name, Last Name, Email (marked with a red asterisk), Subscription (a dropdown menu set to 'Advanced'), Password (marked with a red asterisk), and an 'Upload CSV' section with a 'Select File' button. At the bottom are 'Submit' and 'Clear' buttons. The layout is a single column on both devices.

- Desktop View



The image shows a desktop view of the 'Form & CSV Viewer' application. The form is titled 'Subscribe Form' and includes the same fields as the mobile views: First Name, Last Name, Email (marked with a red asterisk), Subscription (a dropdown menu set to 'Advanced'), Password (marked with a red asterisk), and an 'Upload CSV' section with a 'Select File' button. At the bottom are 'Submit' and 'Clear' buttons. On the desktop, the form fields are arranged in a two-column grid for better readability.



## Summary Page – Responsive Views

- Mobile View - Tablet View

iPhone 14 Pro 393 x 852

Form & CSV Viewer

Submitted Information

First NameDuvan

Last NameMendivelso

Emailduva@hotmail.com

SubscriptionPro

Passworddsadsads\*Odd

File NamedataTest.csv

CSV information

Search...

name	email	age	gender	country
Alice Smith	alice@example.com	28	Female	USA
Bob Johnson	bob@example.com	35	Male	Canada
Carol White	carol@example.com	42	Female	UK
David Lee	david@example.com	30	Male	Australia
Eva Green	eva@example.com	26	Female	Germany
Frank Moore	frank@example.com	38	Male	USA
Grace Hall	grace@example.com	29	Female	Canada
Hank King	hank@example.com	33	Male	UK
Ivy Scott	ivy@example.com	31	Female	Australia
Jack Ray	jack@example.com	27	Male	Germany

PreviousPage 1 of 3Next

iPad Air 5 820 x 1180

Form & CSV Viewer

Submitted Information

First NameDuvan

Last NameMendivelso

Emailduva@hotmail.com

SubscriptionPro

Passworddsadsads\*Odd

File NamedataTest.csv

CSV information

Search...

id	name	email	age	gender	country	city	phone	address	subscription	joined_date
1	Alice Smith	alice@example.com	28	Female	USA	New York	1234567890	123 Main St	Basic	2023-01-10
2	Bob Johnson	bob@example.com	35	Male	Canada	Toronto	2345678901	456 Maple Ave	Pro	2022-11-20
3	Carol White	carol@example.com	42	Female	UK	London	3456789012	789 Oak St	Advanced	2023-03-05
4	David Lee	david@example.com	30	Male	Australia	Sydney	4567890123	321 Pine Rd	Basic	2022-12-01
5	Eva Green	eva@example.com	26	Female	Germany	Berlin	5678901234	654 Cedar Blvd	Pro	2023-02-15
6	Frank Moore	frank@example.com	38	Male	USA	Chicago	6789012345	987 Spruce Ln	Advanced	2022-10-25
7	Grace Hall	grace@example.com	29	Female	Canada	Montreal	7890123456	159 Elm Dr	Basic	2023-01-30
8	Hank King	hank@example.com	33	Male	UK	Manchester	8901234567	753 Birch Pl	Pro	2022-11-12
9	Ivy Scott	ivy@example.com	31	Female	Australia	Melbourne	9012345678	951 Palm Ct	Advanced	2023-03-22
10	Jack Ray	jack@example.com	27	Male	Germany	Hamburg	1234509876	147 Redwood Trl	Basic	2023-02-01

PreviousPage 1 of 3Next

- Desktop View

Form & CSV Viewer

Submitted Information

First NameDuvan

Last NameMendivelso

Emailduva@hotmail.com

SubscriptionPro

Passworddsadsads\*Odd

File Namecsv prueba.csv

CSV information

Search...

Name	Surname	Age	City	Address
John	Doe	25	London	5 Main Street
Jane	Doe	28	London	15 London Street

PreviousPage 1 of 1Next

## Conclusion

This Angular application demonstrates a clean and modular approach to building a fully client-side form solution with dynamic behavior, live validation, and CSV handling. It highlights:

- The use of **Angular signals**, **custom directives**, and **pipes** to manage reactivity and form state.
- **Reusability** through shared components like DynamicForm and DynamicCsvTable.
- **Responsiveness** built with Tailwind CSS to ensure accessibility across devices.
- **Good practices** for structure, session management, and maintainability.

## About the Developer

**Name:** Duvan Mendivelso

**GitHub Repository:** <https://github.com/duvan-23/frontend-code-challenge>

**Email:** duvamendi2@hotmail.com