**Pages/Components**

-   Eliminate deep nesting

-   use `signal` type for all reactive properties.

-   use `ChangeDetectionStrategy.OnPush` in all components for best performance.

-   Maximum file size 200 lines.

-   Implement new reusable components for each usable and separable unit

-   Handle expected errors all the time

-   all API calling buttons MUST be disabled on click and show loading.

-   Handle UI page/component responsiveness to all target devices

**Logic**

-   Write simple, clean and expressive lines of code

-   Avoid `if` nesting

-   Never use `switch` or `ternary` nesting

-   Avoid long `ternaries`: extract values to `variables`

-   Never use `any` datatype

-   Write simple functions and let be pure as possible

-   Eliminate code nesting

-   avoid using magic numbers, extract them to variables.

-   constant values like `strings`, `numbers` should be defined in a central place.

-   **Naming**:

    -   constants -> ALLCAPS

    -   variables -> camelCase

    -   files/folders -> kebab-case

    -   Stick to established conventions

    -   Avoid ambiguous and misleading names

    -   Try to document less known variables, tell what is going on in a short comment

    -   NEVER USE TWO WAYS TO NAME AN OBJECT

    -   Remove buzz names: ~~`PaginationHelper`~~ => `Pagination`

    -   `functions` usually are clear verbs to do a specific action

    -   `variables` are short, mnemonic and meaningful nouns

    -   `classes` are singular descriptive nouns, avoid acronyms unless it's a convention.

**Design Patterns**

-   **Solid Principles**

    -   **Single Responsibility**: every class must have a single, focused responsibility, a single reason to change.

    -   **Open/Closed**: components must be open for extension but closed for modification.

    -   **Liskov Substitution**: you must be able to replace a superclass object with a subclass object without affecting the correctness of the program.

    -   **Interface Segregation**: a client should not be forced to depend on methods it does not use.

    -   **Dependency Inversion**: high-level modules should not depend on low-level modules. Both should depend on abstractions.

-   **KISS**✅

    -   Keep it stupid simple

-   **DRY**✅

    -   Don't repeat yourself

-   **Behavioral**

    -   **Memento**

    -   **State**✅

    -   **Iterator**✅

    -   **Strategy**✅

    -   **Chain of Responsibility**

    -   **Template Method**✅

    -   **Command**

    -   **Mediator**

    -   **Observer**

    -   **Visitor**

-   **Structural**

    -   **Composite**

    -   **Adapter**

    -   **Decorator**

    -   **Facade**✅

    -   **Flyweight**

    -   **Brdige**

    -   **Proxy**

-   **Creational**✅

    -   **Prototype**✅

    -   **Singleton**✅

    -   **Factory Method**✅

    -   **Abstract Factory**✅

    -   **Builder**✅