📘 Framework Structure & Usage Guide

This framework is built on **Playwright + Keyword-Driven POM architecture**.
The goal is to **separate locators, page logic, test logic, and execution control** so tests are scalable and easy to maintain.

1️⃣ Locators Layer

All UI locators are stored separately in the **Locators** folder.

📁 Folder

```
Locators/
│
├── AdminPageLocators.js
├── MainMenuLocators.js
├── Index.js
```

📄 Example: `AdminPageLocators.js`

```
export const AdminPageLocators = {
    systemUsersHeader: "//h5[text()='System Users']"
};
```

📄 Export locators via `Locators/Index.js`

```
export { AdminPageLocators } from './AdminPageLocators.js';
export { MainMenuLocators } from './MainMenuLocators.js';
```

✅ Why

- Single source of truth for selectors
- No hard-coded locators in tests or pages
- Easy updates when UI changes

2️⃣ Tests Folder Structure

The **Tests** folder contains three main sub-folders:

```
tests/
│
├── Pages       → Page Object Model (business actions)
├── Specs       → Test cases
├── Collections → Test execution grouping
```

3️⃣ Pages Layer (POM – Page Object Model)

Page files contain **page-specific actions** using **WebActions keywords**.

📁 Folder

```
tests/Pages/
│
├── AdminPage.js
├── LoginPage.js
├── Index.js
```

📄 Example: `AdminPage.js`

```
import { FailureHandling } from '../../Templates/FailureHandling.js';
import * as Locators from '../../Locators/Index.js';

export default class AdminPage {
    constructor(web) {
        this.web = web;
        this.menuLocators = Locators.MainMenuLocators;
        this.adminLocators = Locators.AdminPageLocators;
    }

    async enterAdminPage() {
        await this.web.click(this.menuLocators.admin);

        await this.web.verifyElementPresent(
            this.adminLocators.systemUsersHeader,
            FailureHandling.STOP_ON_FAILURE
        );
```

```
    }
}
```

📄 Export pages via `Pages/Index.js`

```
export { default as AdminPage } from './AdminPage.js';
export { default as LoginPage } from './LoginPage.js';
```

✅ Rules for Pages

- No assertions in test files
- No direct Playwright calls (`page.click`)
- Only use `web.*` keywords
- One page file per UI page

4️⃣ Specs Layer (Test Cases)

Test files are **very thin** and readable.
They only **call page methods**, not locators or Playwright APIs.

📁 Folder

```
tests/Specs/
│
├── Login/
│   └── Login.spec.js
│
├── Admin/
│   └── AdminPage.spec.js
```

📄 Example: `AdminPage.spec.js`

```
import { test } from '../../../Templates/webFixture.js';
import * as pages from '../../Pages/Index.js';

test('Enter Admin Page', async ({ web }) => {
```

```
    const loginPage = new pages.LoginPage(web);
    const adminPage = new pages.AdminPage(web);

    await loginPage.navigateToLogin(globalVariable.baseUrl);

    await loginPage.login(
        globalVariable.username,
        globalVariable.password
    );

    await loginPage.verifyDashboardVisible();
    await adminPage.enterAdminPage();
});
```

✅ **Why this approach**

- Tests are business readable
- Easy debugging
- Minimal duplication
- Clear flow

5️⃣ Collections Layer (Test Execution Control)

Collections define **which test folders should run together**.

📁 File

`tests/Collections/test-collection.config.js`

📄 Example

```
export const testCollections = {
    smoke: [
        'tests/Specs/Login'
    ],
    regression: [
        'tests/Specs/Login',
        'tests/Specs/Admin'
```

```
    ]
};
```

✅ Once a feature is completed, **add its Specs folder to a collection**.

6️⃣ Running Test Collections

Tests are executed using **Node + Playwright** via `run-collection.js`.

📌 Example Commands

```
ENV=qa node run-collection.js regression
ENV=qa node run-collection.js regression serial
ENV=qa node run-collection.js regression parallel
ENV=qa node run-collection.js regression headed
ENV=qa node run-collection.js regression headed serial
ENV=qa node run-collection.js regression headed parallel
```

🔹 Parameters Explained

Parameter	Meaning
`ENV=qa`	Environment (qa / uat / prod)
`regression`	Test collection name
`headed`	Runs browser in UI mode
`serial`	Runs tests sequentially
`parallel`	Runs tests in parallel

7️⃣ Templates Folder (Global Configuration)

📁 Folder

```
Templates/
│
├── webFixture.js
```

```
├── FailureHandling.js
├── GlobalVariables.js
```

📄 Global Variables

`GlobalVariables.js` is used to store variables accessible across all tests.

```
global.globalVariable = {
    baseUrl: process.env.BASE_URL,
    username: process.env.USERNAME,
    password: process.env.PASSWORD
};
```

8️⃣ Mandatory Imports

✅ In `.spec.js` files

```
import { test } from '../../../Templates/webFixture.js';
import * as pages from '../../Pages/Index.js';
```

✅ In Page `.js` files (POM)

```
import { FailureHandling } from '../../Templates/FailureHandling.js';
import * as Locators from '../../Locators/Index.js';
```

9️⃣ Overall Flow Summary

```
Spec (.spec.js)
   ↓
Page Object (.js)
   ↓
WebActions Keywords
   ↓
Playwright Engine
```
