Below is an overview of project structures and usage for various automation frameworks spanning web, mobile, and API testing with different languages and tools. Customise these templates based on your project's needs.

Final Note

1. Selenium + BDD (Cucumber) + TestNG + POM + Java (Web Automation)

Project Structure:

```
SeleniumBDD_Web/
├── src
│   ├── main
│   │   └── java
│   │       └── pageObjects
│   │           ├── LoginPage.java
│   │           └── HomePage.java
│   └── test
│       └── java
│           ├── stepDefinitions
│           │   ├── Hooks.java
│           │   └── LoginSteps.java
│           ├── runners
│           │   └── TestRunner.java
│           └── features
│               └── login.feature
├── pom.xml
└── testng.xml
```


Usage:

- **Page Objects:** Encapsulate UI elements and actions.
- **Step Definitions:** Map Gherkin steps to Selenium methods.
- **Test Runner & TestNG:** Execute tests and generate reports.
- **BDD Feature Files:** Write human-readable test scenarios.

---

2. Appium + BDD (Cucumber) + TestNG + POM + Java (Mobile Automation)

Project Structure:

```
AppiumBDD_Mobile/
├── src
│   ├── main
│   │   └── java
│   │       ├── pageObjects
│   │       │   ├── LoginPage.java
│   │       │   └── HomePage.java
│   │       └── utils
│   │           └── AppiumUtils.java
│   └── test
│       └── java
│           ├── stepDefinitions
│           │   ├── Hooks.java
│           │   └── LoginSteps.java
│           ├── runners
│           │   └── TestRunner.java
│           └── features
│               └── login.feature
├── pom.xml
└── testng.xml
```


Usage:

- **Page Objects:** Represent mobile screens with Appium locators.
- **Utils:** Configure Appium driver capabilities.
- **Step Definitions & Feature Files:** Define mobile test scenarios.
- **TestNG:** Runs tests on mobile devices/emulators.

---

3. Rest Assured + BDD (Cucumber) + TestNG + POM + Java (API Automation)

Project Structure:

```
RestAssuredBDD_API/
├── src
│   └── test
│       └── java
│           ├── stepDefinitions
│           │   ├── Hooks.java
│           │   └── ApiSteps.java
│           ├── runners
│           │   └── TestRunner.java
│           ├── features
│           │   └── apiTesting.feature
│           └── utils
│               └── ApiUtils.java
├── pom.xml
└── testng.xml
```


Usage:

- **API Utils:** Helper methods to build and send API requests using Rest Assured.
- **Step Definitions:** Implement BDD steps that interact with APIs.
- **Feature Files:** Define API scenarios (GET, POST, PUT, DELETE).
- **TestNG & Cucumber:** Manage test execution and reporting.

---

4. Selenium + BDD (Behave) + Pytest + POM + Python (Web Automation)

Project Structure:

```
PytestBDD_Web/
├── pages
│   ├── base_page.py
│   └── login_page.py
├── features
│   ├── login.feature
│   └── steps
│       ├── __init__.py
│       └── login_steps.py
├── tests
│   └── test_runner.py   # Could simply run behave from command line
├── conftest.py        # Pytest fixtures if needed
├── pytest.ini
└── requirements.txt
```


Usage:

- **Pages:** Define POM classes for web pages.
- **Feature Files & Steps:** Write BDD scenarios in Gherkin and implement them using Behave.
- **Pytest:** Can be used alongside Behave for fixtures and parallel execution.
- **Configuration:** Use pytest.ini and conftest.py to manage settings and fixtures.

---

5. Appium + BDD (Behave) + Pytest + POM + Python (Mobile Automation)

Project Structure:

```
PytestBDD_Mobile/
├── pages
│   ├── base_page.py
│   └── login_page.py
├── features
│   ├── login.feature
│   └── steps
│       ├── __init__.py
│       └── login_steps.py
├── tests
│   └── test_runner.py
├── utils
│   └── appium_utils.py
├── conftest.py
├── pytest.ini
└── requirements.txt
```


Usage:

- **Pages:** Define mobile page objects with Appium locators.
- **Utils:** Configure Appium driver and capabilities.
- **Feature Files & Steps:** Write mobile test scenarios in Gherkin.
- **Pytest Fixtures:** Set up and tear down mobile sessions.
- **Run Tests:** Execute Behave scenarios using Pytest command-line.

---

6. Rest + BDD (Behave) + Pytest + POM + Python (API Automation)

Project Structure:

```
PytestBDD_API/
├── features
│   ├── api.feature
│   └── steps
│       ├── __init__.py
│       └── api_steps.py
├── tests
│   └── test_runner.py
├── utils
│   └── api_utils.py
├── conftest.py
├── pytest.ini
└── requirements.txt
```


Usage:

- **Utils:** Implement functions to send API requests (using requests library).
- **Feature Files:** Define API test scenarios in Gherkin.
- **Step Definitions:** Implement steps to call APIs and validate responses.
- **Pytest Fixtures:** Manage API session or test data.
- **Run Tests:** Use pytest or behave to execute tests.

---

7. Cypress + BDD + POM (Web Automation with JavaScript)

Project Structure:

```
cypress-bdd-web/
├── cypress
│   ├── integration
│   │   └── features
│   │       └── login.feature
│   ├── support
│   │   ├── pageObjects
│   │   │   └── loginPage.js
│   │   └── step_definitions
│   │       └── loginSteps.js
├── package.json
├── cypress.json
└── README.md
```


Usage:

- **Feature Files:** Write BDD scenarios in Gherkin.
- **Step Definitions:** Map Gherkin steps to Cypress commands.
- **Page Objects:** Create reusable components for UI interactions.
- **Run Tests:** Execute using Cypress Test Runner or via CLI.

---

8. Cypress + BDD + POM + API (API Automation with Cypress and JavaScript)

Project Structure:

```
cypress-bdd-api/
├── cypress
│   ├── integration
│   │   ├── features
│   │   │   ├── login.feature      # Web feature scenarios
│   │   │   └── api.feature        # API test scenarios
│   ├── support
│   │   ├── pageObjects
│   │   │   └── loginPage.js       # POM file for web interactions
│   │   └── step_definitions
│   │       ├── loginSteps.js      # Step definitions for web tests
│   │       └── apiSteps.js        # Step definitions for API tests (using Cypress commands and cy.request())
├── package.json
├── cypress.json                   # Global configuration for Cypress
└── README.md
```


Usage:

- **Web & API Tests:** Write separate feature files for UI and API.
- **Step Definitions:** Implement actions using Cypress for UI and HTTP calls for API tests.
- **Page Objects:** Reuse selectors and actions across tests.
- **Run Tests:** Cypress Test Runner for UI; use plugins or custom commands for API testing.

---

9. WebdriverIO + BDD + Selenium + Appium (Web and Mobile with JavaScript)

Project Structure:

```
webdriverio-bdd/
├── test
│   ├── features
│   │   └── login.feature
│   ├── stepDefinitions
│   │   └── login.steps.js
│   ├── pageobjects
│   │   ├── login.page.js
│   │   └── home.page.js
├── wdio.conf.js
├── package.json
└── README.md
```

Usage:

- **Web & Mobile Automation:** Use Selenium for web and Appium for mobile.
- **BDD with Cucumber:** Write features in Gherkin.
- **Step Definitions:** Implement steps using WebdriverIO commands.
- **Page Objects:** Centralize element selectors and actions.
- **Configuration:** Use wdio.conf.js to define capabilities and test suites.
- **Run Tests:** Use WebdriverIO CLI to execute tests.

10. Playwright + BDD (Cucumber/Pytest-BDD) – Unified for Web, Mobile, API, Accessibility, Load, & Performance

Project Structure:

```
playwright-bdd/
├── src
│   ├── test
│   │   ├── features
│   │   │   ├── web.feature        # Web UI scenarios
│   │   │   ├── mobile.feature     # Mobile emulation scenarios
│   │   │   └── api.feature        # API testing scenarios
│   │   ├── stepDefinitions
│   │   │   ├── webSteps.ts        # Steps for web testing using Playwright commands
│   │   │   ├── mobileSteps.ts     # Steps for mobile testing (using viewport emulation or real device capabilities)
│   │   │   └── apiSteps.ts        # Steps for API testing using Playwright's request API
│   │   └── pages
│   │       ├── webPage.ts         # Page Object Model for web pages
│   │       ├── mobilePage.ts      # POM for mobile pages (or responsive views)
│   │       └── apiHelper.ts       # Utility functions for API requests and validations
├── playwright.config.ts           # Global configuration for Playwright tests (multiple projects, timeouts, reporters, etc.)
├── package.json
└── tsconfig.json
```


Usage:

- **Unified Framework:** One framework to test Web UI, Mobile Web (using mobile emulation), API endpoints, accessibility, load, and performance.
- **Feature Files:** Separate features for web, mobile, and API.
- **Step Definitions:** Implement tests in TypeScript.

- **Page Objects:** Maintainable POM for different testing domains.
- **Playwright Config:** Define projects, browsers, and parallelism.
- **Run Tests:** Use npx playwright test to execute all tests.
- **CI/CD Integration:** Integrate with GitHub Actions, Jenkins, etc.

---

11. Spring Boot + Java + Selenium + POM – Web Automation with Spring Boot Java

Project Structure:

```
springboot-selenium/
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com/example/pages
│   │   │       ├── BasePage.java     # Common Selenium functions
│   │   │       └── LoginPage.java    # Page Object for Login page
│   │   └── resources
│   │       └── application.properties # Spring Boot configuration
│   ├── test
│   │   └── java
│   │       └── com/example/tests
│   │           ├── LoginTest.java    # Selenium test using TestNG/JUnit
│   │           └── TestRunner.java   # Test suite configuration
├── pom.xml                           # Maven configuration and dependencies (Spring Boot, Selenium, TestNG/JUnit, etc.)
└── README.md
```

Usage:

- **Spring Boot Application:** Acts as the backend or serves the web app.
- **Page Objects:** Encapsulate UI interactions.
- **Test Classes:** Use Selenium WebDriver with JUnit/TestNG.
- **Maven (pom.xml):** Manage dependencies and build lifecycle.
- **Test Runner:** Execute tests against the deployed Spring Boot application.

Final Note

Each framework is tailored to a specific testing domain (web, mobile, API) and uses a combination of BDD, POM, and test runners (TestNG/Pytest) to enhance maintainability, readability, and scalability.

These project structures serve as starting points. Customize them further based on:

- **Team preferences**
- **Project complexity**
- **Tool integrations (CI/CD, reporting, etc.)**

Happy Learning! 🌸
