# Testing Framework – Overview

**Centric PLM** supports both **UI automation** (Playwright-based browser testing) and **API automation** (HTTP-based validation).
It provides **dynamic environment management**, **centralized data control**, and **automatic reporting** — all in one unified structure.

---

## Framework Architecture

```
CentricPLM/
├── ui/                    # UI Testing (Browser Automation)
│   ├── features/          # BDD Feature Files
│   ├── step_definitions/  # Step Definitions
│   ├── pom/               # Page Object Model (Reusable UI Actions)
│   ├── test_data/         # UI Test Data & Credentials
│   └── containers/        # Shared Runtime Storage
│
├── api/                   # API Testing (HTTP Automation)
│   ├── features/          # BDD Feature Files
│   ├── step_definitions/  # Step Implementations
│   ├── payloads/          # Request/Response Payload Classes
│   ├── framework/         # Base API Client
│   ├── test_data/         # Endpoint & Data Configurations
│   └── containers/        # Shared API State Storage
│
├── config/                # Environment & Credential Management
│   └── global_config.py
│
├── reports/               # Reports Directory (UI + API)
│   ├── index.html         # Unified Report Index
│   ├── ui/                # UI Reports
│   └── api/               # API Reports
│
├── run_tests.py           # Combined Runner (UI + API)
├── generate_api_reports.py  # API Runner (Auto HTML Reports)
├── generate_reports.py      # UI Runner (Auto HTML Reports)
└── requirements.txt         # Python Dependencies
```

---

# Key Modules Explained

## Configuration

- `global_config.py` — central environment manager (DEV, SIT, UAT, PROD)
- Defines:
    - Base URLs
    - Credentials
    - Test modes (UI / API / BOTH)

## API Layer

- Handles API testing with **cookie-based authentication** (`SecurityTokenURL`)
- Includes payload builders (`login_payload.py`, `style_payload.py`)
- Supports POST, GET, PATCH, and DELETE requests with built-in logging
- Manages dynamic test data through `variable_container.py`

## UI Layer

- Built using **Playwright** for browser automation
- Follows the **Page Object Model (POM)** pattern:
    - Easy locator management
    - Reusable page interactions
- BDD test flow:
    - `.feature` files → readable test scenarios
    - `step_definitions/` → implementation logic

- Data isolation using `ui_container.py`

### Reports

- Automatically generated **HTML, Console, and JSON** reports
- Auto-opens after test execution
- Maintains both **UI** and **API** reports separately with a unified index

---

## Execution Commands

| Type | Command | Output |
|---|---|---|
| **Run UI Tests** | `python3 generate_reports.py` | UI automation + HTML report |
| **Run API Tests** | `python3 generate_api_reports.py` | API flow + HTML & console reports |
| **Run Both** | `python3 run_tests.py` | Full regression suite (UI + API) |

---

## Core Capabilities

| Category | Highlights |
|---|---|
| **Environment Control** | Switch between UAT, PROD, etc. from config |
| **Test Data Management** | Centralized JSON & Python files |
| **UI Automation** | Playwright with POM for modular actions |
| **API Automation** | RESTful client with reusable payloads |
| **Data Sharing** | Containers for step-level and module-level reuse |
| **Error Handling** | Built-in logging, retries, and validations |
| **Reporting** | Rich HTML, JSON, and console summaries |

| | |
|---|---|
| **Scalability** | Plug-and-play structure for new tests & modules |

---

## API Style Creation Flow (Located in `api/`)

This part of the framework is handled inside the **`api/`** folder, which automates backend validation through API calls.

**Flow Steps (and where they live):**

| Step | What It Does | Framework Location |
|---|---|---|
| **Login to PLM API** | Authenticates using credentials and generates a secure token | `api/payloads/login_payload.py` + `api/framework/base_api.py` |
| **Fetch existing reference IDs** | Retrieves valid IDs like category, season, and product type for style creation | `api/step_definitions/api_steps.py` |
| **Create new Style (POST /styles)** | Submits the style creation payload using Base API client | `api/payloads/style_payload.py` + `api/framework/base_api.py` |
| **Validate response (GET /styles/{id})** | Confirms that the new style exists in the database | `api/step_definitions/api_steps.py` |

| | | |
|---|---|---|
| **Store data in container** | Saves style ID, code, and other response data for later UI use | `api/containers/variable_container.py` |

---

## UI Verification Flow (Located in `ui/`)

This part of the framework, found under the **`ui/`** folder, automates frontend validation using Playwright.
It verifies that the style created via API is **visible and correct** in the PLM web application.

**Flow Steps (and where they live):**

| Step | What It Does | Framework Location |
|---|---|---|
| **Launch browser session** | Opens the Centric PLM web application | `ui/step_definitions/plm_ui_steps.py` |
| **Login to PLM** | Logs in using credentials from test data | `ui/test_data/login_credentials.json` |
| **Search for created style** | Finds the API-created | `ui/pom/plm_style_creation_page.py` |

| | | |
|---|---|---|
| **(by code)** | style in the UI grid | |
| **Validate lifecycle and brand details** | Confirms style properties match API data | `ui/pom/plm_style_creation_page.py` + `ui/step_definitions/plm_ui_steps.py` |
| **Capture screen shot & store in report** | Takes screenshot and attaches to report | `ui/tests/validate_and_store_style_details.py` + `reports/ui/` |

---

## Connection Between API and UI Layers

| Component | Purpose | Connection |
|---|---|---|
| `variable_container.py` | Stores data like style ID and code | Shared between API & UI |
| `reports/` | Stores combined HTML reports for UI and API | Accessible to all runs |
| `config/global_config.py` | Handles environment, credentials, and mode | Controls both UI & API |

| | | |
|---|---|---|
| `run_tests.py` | Runs UI + API together | Unified execution entry point |

---

## CORE TESTING FRAMEWORKS

- **Playwright** – Browser automation for UI testing
- **Pytest** – Core Python testing framework
- **Pytest-BDD** – Behavior-Driven Development (Gherkin support)
- **Pytest-Asyncio** – Asynchronous test execution
- **Pytest-HTML** – HTML report generation
- **Pytest-JSON-Report** – Machine-readable test output
- **Allure-Pytest** – Advanced test analytics and professional reporting

---

## API TESTING

- **HTTPX** – Asynchronous HTTP client (primary)
- **Requests** – Synchronous HTTP client (fallback)
- **JSON** – Serialization and deserialization
- **Cookie Management** – Session persistence and authentication handling

---

## UI TESTING

- **Playwright** – Cross-browser automation

- **Page Object Model (POM)** – Reusable UI design pattern
- **Dojo/Dijit Framework Handling** – Centric PLM-specific UI elements
- **Locator Strategies** – XPath, CSS, and data attribute locators
- **JavaScript Execution** – Direct DOM interactions and custom scripts

---

## REPORTING & VISUALIZATION

- **HTML Reports** – Technical and readable test results
- **JSON Reports** – Data-driven report storage
- **Allure Reports** – Executive-level analytics and dashboards
- **Console Output Capture** – Detailed runtime logs
- **Screenshots & Visual Evidence** – Automatic captures for validation
- **Custom Styling** – CSS/JS enhancements for report presentation

---

## CONFIGURATION & DATA MANAGEMENT

- **JSON Configuration Files** – Environment-specific settings
- **YAML Support** – Optional config format
- **Environment Variables** – Dynamic runtime configuration
- **Centralized Test Data** – Shared data for UI & API tests
- **Secure Credential Storage** – Encrypted login credentials

---

## FRAMEWORK ARCHITECTURE

- **Page Object Model (POM)** – Organized UI automation structure
- **Step Definitions** – BDD implementation layer
- **Feature Files** – Human-readable Gherkin test cases
- **Data Containers** – Shared test data across modules
- **Base Classes** – Common reusable utilities
- **Utility Functions** – Helpers for logging, parsing, and validation

---

## PYTHON ECOSYSTEM

- **Python 3.9+** – Core language
- **Asyncio** – Async operations and concurrency
- **Pathlib** – File path management
- **Datetime** – Time and scheduling utilities
- **Subprocess** – Process handling for parallel runs
- **Webbrowser** – Auto-open reports after test completion

---

## UI FRAMEWORK INTEGRATION

- **Dojo/Dijit Components** – Handling dynamic Centric PLM widgets
- **Dynamic Locators** – For unique widget IDs (`widget_uniqName_*`)
- **Stable Data Locators** – Using `data-csi-form-field` attributes

- **Dropdown & Dialog Management** – Complex UI interactions
- **Grid & Table Operations** – Automated verification of records

---

## AUTHENTICATION & SECURITY

- **Cookie-Based Authentication** – Session management for PLM
- **Token Management** – API authentication lifecycle
- **Credential Encryption** – Secured credential handling
- **Session Persistence** – Continuous login state across tests
- **CSRF Protection** – Handling Centric PLM security tokens

---

## MONITORING & DEBUGGING

- **Detailed Console Logging** – Step-by-step output
- **Error Handling** – Graceful failure and retry logic
- **Retry Mechanisms** – For flaky or delayed test cases
- **Screenshot Capture** – For visual debugging
- **Response Time Metrics** – API and UI performance tracking
- **Status Tracking** – Continuous test progress updates

- **Pytest Integration** – Seamless test discovery
- **HTML/Allure Reports** – Auto-publish reports post-run
- **Cross-Environment Deployment** – Run on any test stage (DEV → PROD)

---

# PROJECT STRUCTURE OVERVIEW

```
CentricPLM/
├── api/              # API testing components
├── ui/               # UI testing components
├── config/           # Environment configuration
├── reports/          # All reports (HTML, JSON, Allure)
├── test_data/        # Test inputs and credentials
├── features/         # BDD feature files
├── step_definitions/  # Implementation of steps
└── utils/            # Utility and helper modules
```

---

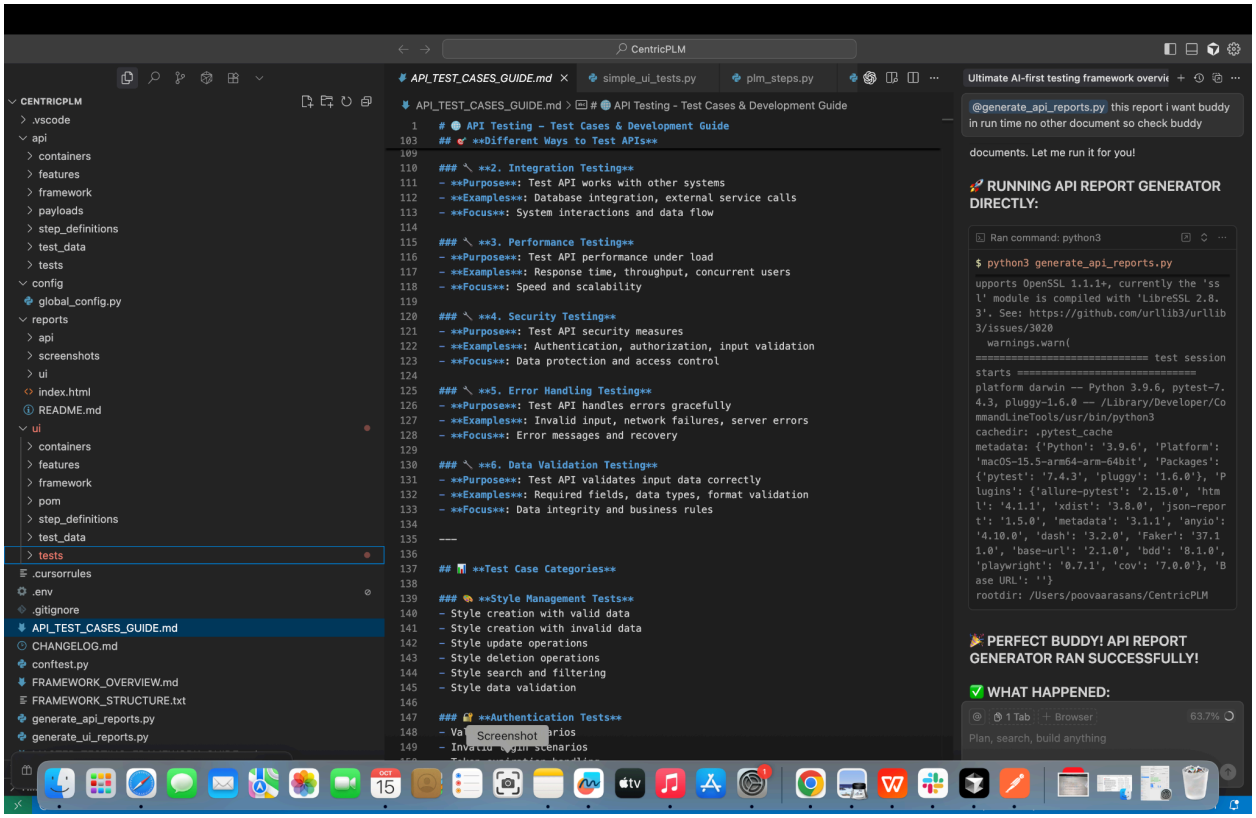
# TEST EXECUTION MODES

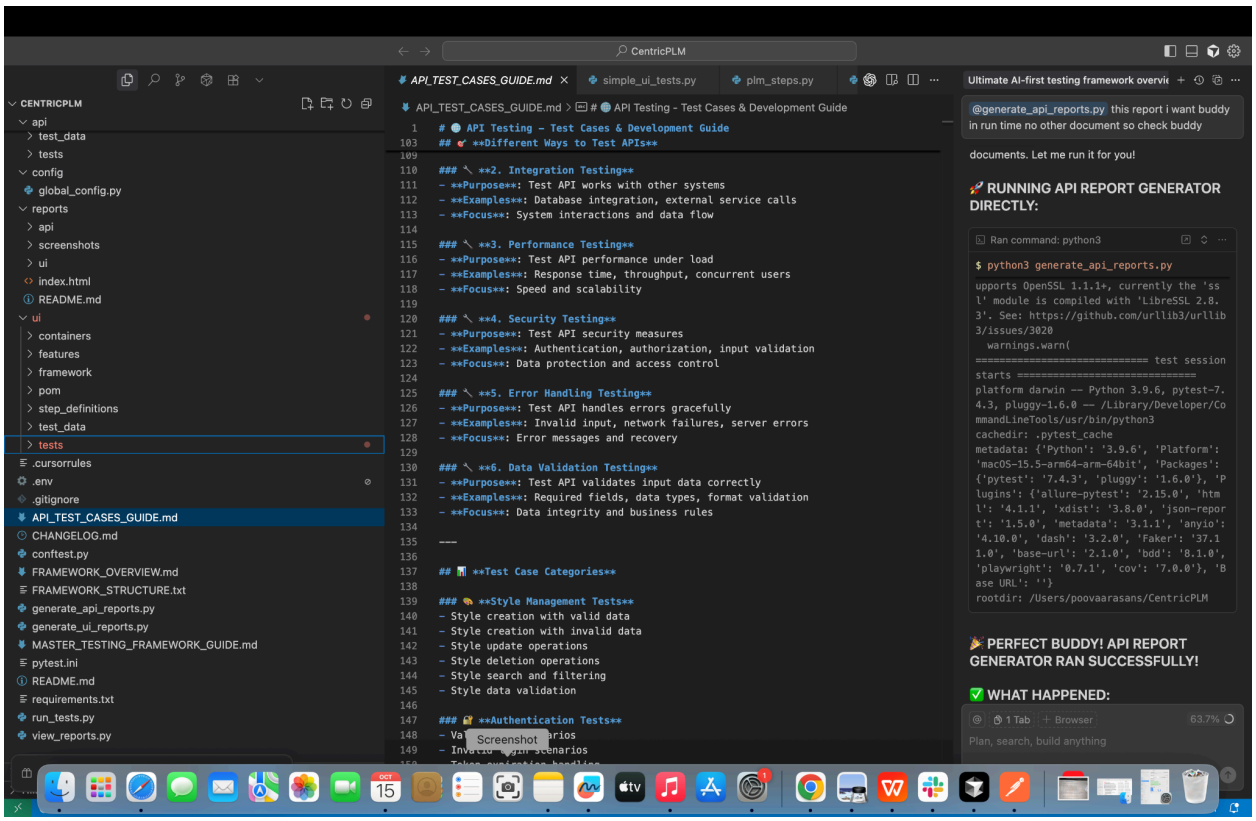
- **UI Mode** – Runs Playwright UI automation only
- **API Mode** – Executes HTTP-based API tests only
- **Combined Mode** – Runs both API & UI tests together
- **Regression Mode** – Executes the entire suite for validation

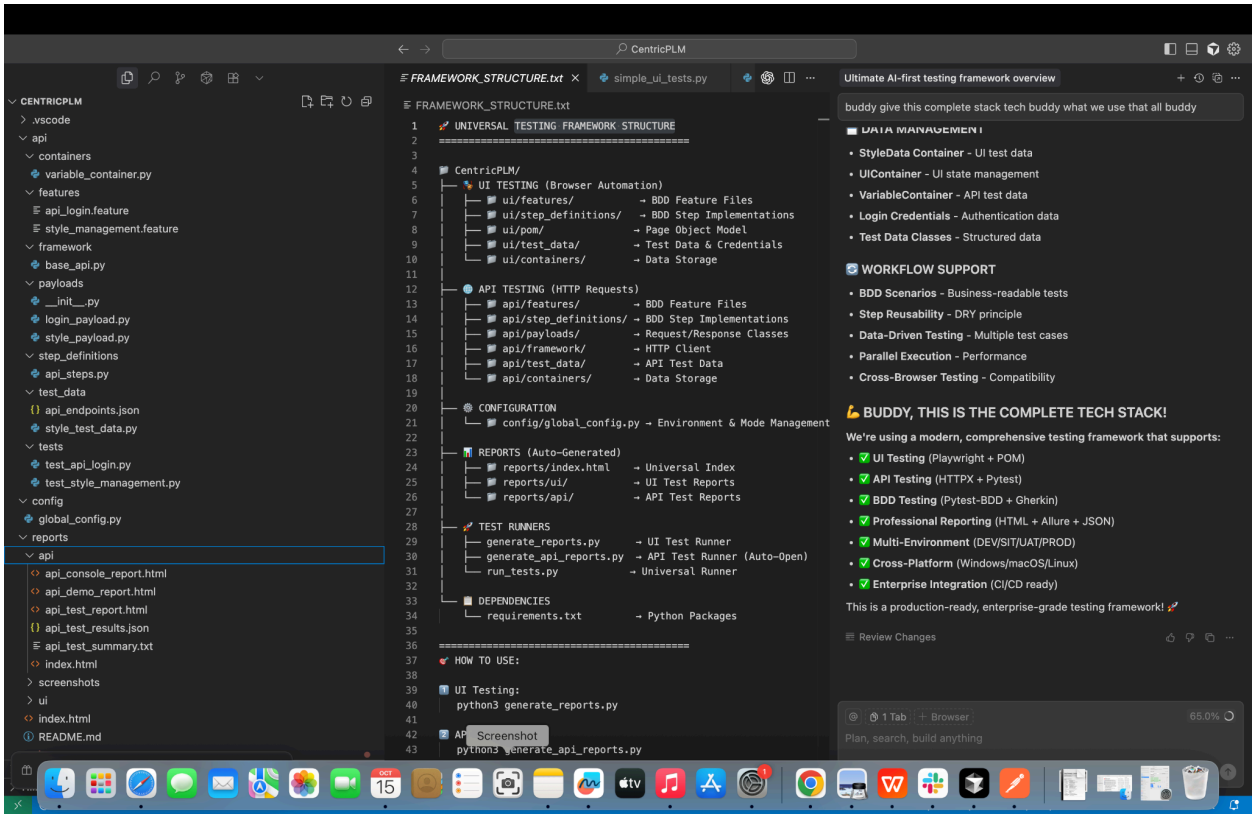
---

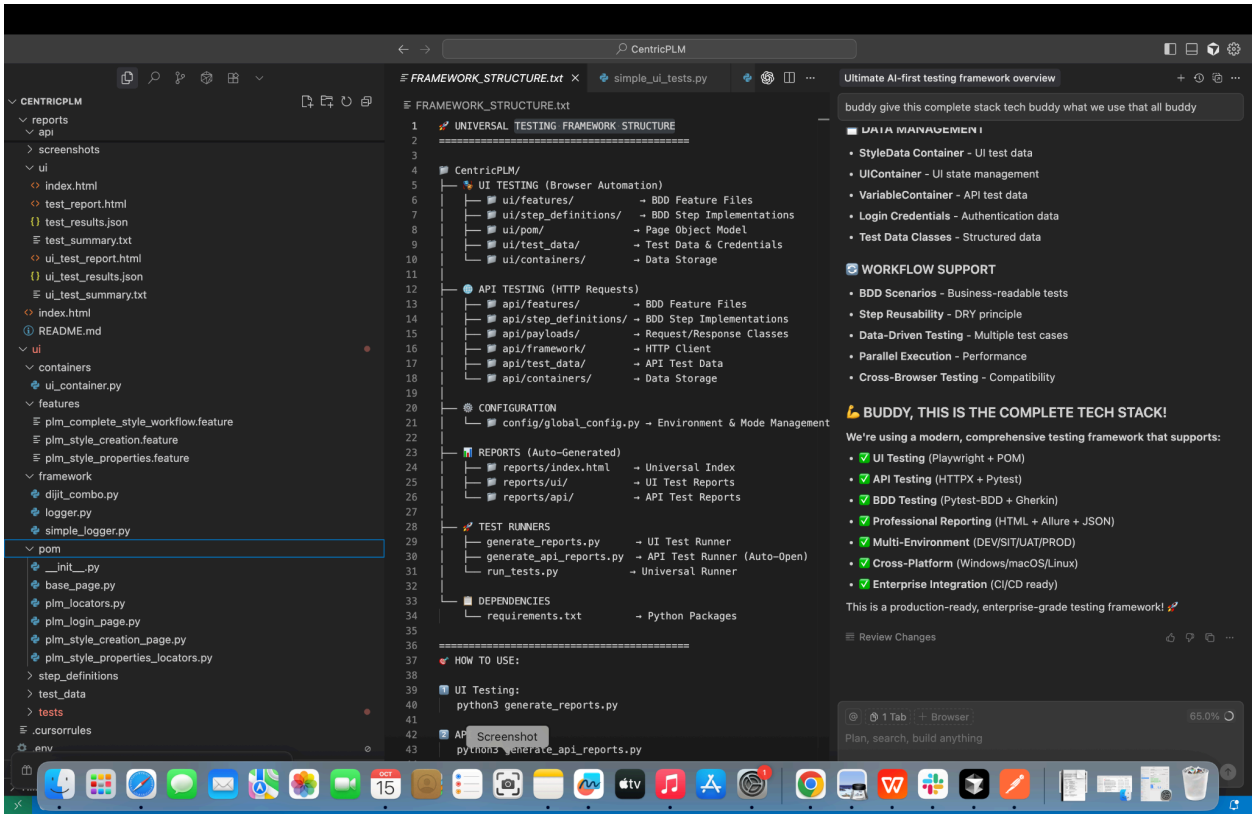
# DATA HANDLING

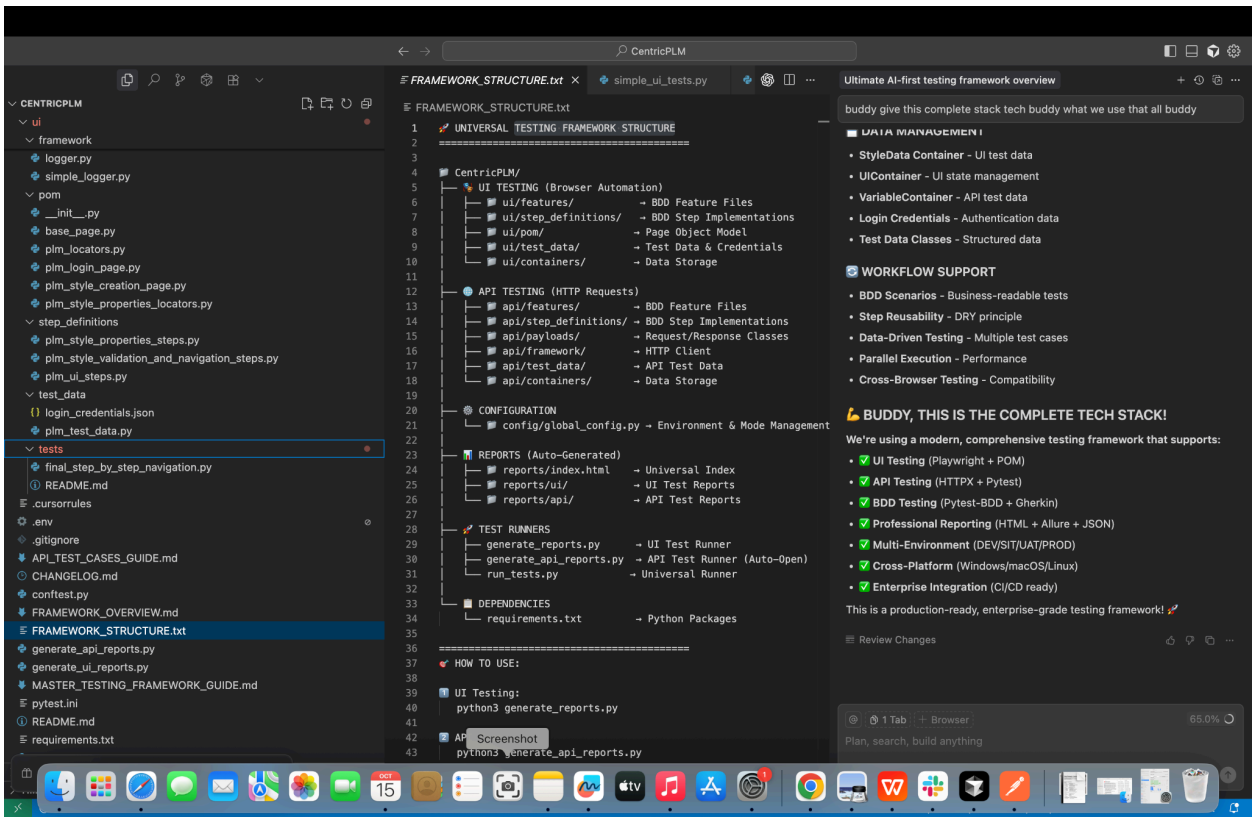
- **VariableContainer** – Stores API runtime data
- **UIContainer** – Maintains UI session data
- **StyleData Container** – Holds style-specific details
- **Test Data Classes** – Structured data management


CentricPLM
API_TEST_CASES_GUIDE.md
simple_ui_tests.py
plm_steps.py
CENTRICPLM
.vscode
api
containers
features
framework
payloads
step_definitions
test_data
tests
config
global_config.py
reports
api
screenshots
ui
index.html
README.md
ui
containers
features
framework
pom
step_definitions
test_data
tests
.cursorrules
.env
.gitignore
API_TEST_CASES_GUIDE.md
CHANGELOG.md
conftest.py
FRAMEWORK_OVERVIEW.md
FRAMEWORK_STRUCTURE.txt
generate_api_reports.py
generate_ui_reports.py
API_TEST_CASES_GUIDE.md > # 🔵 API Testing - Test Cases & Development Guide
# 🔵 API Testing - Test Cases & Development Guide
## 🎯 **Different Ways to Test APIs**
### 🔧 **2. Integration Testing**
- **Purpose**: Test API works with other systems
- **Examples**: Database integration, external service calls
- **Focus**: System interactions and data flow
### 🔧 **3. Performance Testing**
- **Purpose**: Test API performance under load
- **Examples**: Response time, throughput, concurrent users
- **Focus**: Speed and scalability
### 🔧 **4. Security Testing**
- **Purpose**: Test API security measures
- **Examples**: Authentication, authorization, input validation
- **Focus**: Data protection and access control
### 🔧 **5. Error Handling Testing**
- **Purpose**: Test API handles errors gracefully
- **Examples**: Invalid input, network failures, server errors
- **Focus**: Error messages and recovery
### 🔧 **6. Data Validation Testing**
- **Purpose**: Test API validates input data correctly
- **Examples**: Required fields, data types, format validation
- **Focus**: Data integrity and business rules
---
## 📊 **Test Case Categories**
### 🎨 **Style Management Tests**
- Style creation with valid data
- Style creation with invalid data
- Style update operations
- Style deletion operations
- Style search and filtering
- Style data validation
### 🔐 **Authentication Tests**
- Va
- Invalid
Screenshot
Ultimate AI-first testing framework overvi
@generate_api_reports.py this report i want buddy in run time no other document so check buddy
documents. Let me run it for you!
🚀 RUNNING API REPORT GENERATOR DIRECTLY:
Ran command: python3
$ python3 generate_api_reports.py
upports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
========== test session starts ==========
platform darwin -- Python 3.9.6, pytest-7.4.3, pluggy-1.6.0 -- /Library/Developer/CommandLineTools/usr/bin/python3
cachedir: .pytest_cache
metadata: {'Python': '3.9.6', 'Platform': 'macOS-15.5-arm64-arm-64bit', 'Packages': {'pytest': '7.4.3', 'pluggy': '1.6.0'}, 'Plugins': {'allure-pytest': '2.15.0', 'html': '4.1.1', 'xdist': '3.8.0', 'json-report': '1.5.0', 'metadata': '3.1.1', 'anyio': '4.10.0', 'dash': '3.2.0', 'Faker': '37.11.0', 'base-url': '2.1.0', 'bdd': '8.1.0', 'playwright': '0.7.1', 'cov': '7.0.0'}, 'Base URL': ''}
rootdir: /Users/poovaarasans/CentricPLM
🎉 PERFECT BUDDY! API REPORT GENERATOR RAN SUCCESSFULLY!
✅ WHAT HAPPENED:
1 Tab
Browser
63.7%
Plan, search, build anything


CentricPLM
API_TEST_CASES_GUIDE.md
simple_ui_tests.py
plm_steps.py
CENTRICPLM
api
test_data
tests
config
global_config.py
reports
api
screenshots
ui
index.html
README.md
ui
containers
features
framework
pom
step_definitions
test_data
tests
.cursorrules
.env
.gitignore
API_TEST_CASES_GUIDE.md
CHANGELOG.md
conftest.py
FRAMEWORK_OVERVIEW.md
FRAMEWORK_STRUCTURE.txt
generate_api_reports.py
generate_ui_reports.py
MASTER_TESTING_FRAMEWORK_GUIDE.md
pytest.ini
README.md
requirements.txt
run_tests.py
view_reports.py
API_TEST_CASES_GUIDE.md > # 🔵 API Testing - Test Cases & Development Guide
# 🔵 API Testing - Test Cases & Development Guide
## 🎯 **Different Ways to Test APIs**
### 🔧 **2. Integration Testing**
- **Purpose**: Test API works with other systems
- **Examples**: Database integration, external service calls
- **Focus**: System interactions and data flow
### 🔧 **3. Performance Testing**
- **Purpose**: Test API performance under load
- **Examples**: Response time, throughput, concurrent users
- **Focus**: Speed and scalability
### 🔧 **4. Security Testing**
- **Purpose**: Test API security measures
- **Examples**: Authentication, authorization, input validation
- **Focus**: Data protection and access control
### 🔧 **5. Error Handling Testing**
- **Purpose**: Test API handles errors gracefully
- **Examples**: Invalid input, network failures, server errors
- **Focus**: Error messages and recovery
### 🔧 **6. Data Validation Testing**
- **Purpose**: Test API validates input data correctly
- **Examples**: Required fields, data types, format validation
- **Focus**: Data integrity and business rules
---
## 📊 **Test Case Categories**
### 🎨 **Style Management Tests**
- Style creation with valid data
- Style creation with invalid data
- Style update operations
- Style deletion operations
- Style search and filtering
- Style data validation
### 🔐 **Authentication Tests**
- Va
- Invalid
Screenshot
Ultimate AI-first testing framework overvi
@generate_api_reports.py this report i want buddy in run time no other document so check buddy
documents. Let me run it for you!
🚀 RUNNING API REPORT GENERATOR DIRECTLY:
Ran command: python3
$ python3 generate_api_reports.py
upports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
========== test session starts ==========
platform darwin -- Python 3.9.6, pytest-7.4.3, pluggy-1.6.0 -- /Library/Developer/CommandLineTools/usr/bin/python3
cachedir: .pytest_cache
metadata: {'Python': '3.9.6', 'Platform': 'macOS-15.5-arm64-arm-64bit', 'Packages': {'pytest': '7.4.3', 'pluggy': '1.6.0'}, 'Plugins': {'allure-pytest': '2.15.0', 'html': '4.1.1', 'xdist': '3.8.0', 'json-report': '1.5.0', 'metadata': '3.1.1', 'anyio': '4.10.0', 'dash': '3.2.0', 'Faker': '37.11.0', 'base-url': '2.1.0', 'bdd': '8.1.0', 'playwright': '0.7.1', 'cov': '7.0.0'}, 'Base URL': ''}
rootdir: /Users/poovaarasans/CentricPLM
🎉 PERFECT BUDDY! API REPORT GENERATOR RAN SUCCESSFULLY!
✅ WHAT HAPPENED:
1 Tab
Browser
63.7%
Plan, search, build anything


CentricPLM
FRAMEWORK_STRUCTURE.txt
simple_ui_tests.py
Ultimate AI-first testing framework overview
buddy give this complete stack tech buddy what we use that all buddy
CENTRICPLM
.vscode
api
containers
variable_container.py
features
api_login.feature
style_management.feature
framework
base_api.py
payloads
__init__.py
login_payload.py
style_payload.py
step_definitions
api_steps.py
test_data
api_endpoints.json
style_test_data.py
tests
test_api_login.py
test_style_management.py
config
global_config.py
reports
api
api_console_report.html
api_demo_report.html
api_test_report.html
api_test_results.json
api_test_summary.txt
index.html
screenshots
ui
index.html
README.md
FRAMEWORK_STRUCTURE.txt
1 UNIVERSAL TESTING FRAMEWORK STRUCTURE
2 ==========
3
4 CentricPLM/
5 ├── UI TESTING (Browser Automation)
6 │ ├── ui/features/ → BDD Feature Files
7 │ ├── ui/step_definitions/ → BDD Step Implementations
8 │ ├── ui/pom/ → Page Object Model
9 │ ├── ui/test_data/ → Test Data & Credentials
10 │ └── ui/containers/ → Data Storage
11 │
12 ├── API TESTING (HTTP Requests)
13 │ ├── api/features/ → BDD Feature Files
14 │ ├── api/step_definitions/ → BDD Step Implementations
15 │ ├── api/payloads/ → Request/Response Classes
16 │ ├── api/framework/ → HTTP Client
17 │ ├── api/test_data/ → API Test Data
18 │ └── api/containers/ → Data Storage
19 │
20 ├── CONFIGURATION
21 │ └── config/global_config.py → Environment & Mode Management
22 │
23 ├── REPORTS (Auto-Generated)
24 │ ├── reports/index.html → Universal Index
25 │ ├── reports/ui/ → UI Test Reports
26 │ └── reports/api/ → API Test Reports
27 │
28 ├── TEST RUNNERS
29 │ ├── generate_reports.py → UI Test Runner
30 │ ├── generate_api_reports.py → API Test Runner (Auto-Open)
31 │ └── run_tests.py → Universal Runner
32 │
33 └── DEPENDENCIES
34 └── requirements.txt → Python Packages
35
36 ==========
37 HOW TO USE:
38
39 1 UI Testing:
40 python3 generate_reports.py
41
42 2 API
43 python3 generate_api_reports.py
Screenshot
DATA MANAGEMENT
StyleData Container - UI test data
UIContainer - UI state management
VariableContainer - API test data
Login Credentials - Authentication data
Test Data Classes - Structured data
WORKFLOW SUPPORT
BDD Scenarios - Business-readable tests
Step Reusability - DRY principle
Data-Driven Testing - Multiple test cases
Parallel Execution - Performance
Cross-Browser Testing - Compatibility
BUDDY, THIS IS THE COMPLETE TECH STACK!
We're using a modern, comprehensive testing framework that supports:
UI Testing (Playwright + POM)
API Testing (HTTPX + Pytest)
BDD Testing (Pytest-BDD + Gherkin)
Professional Reporting (HTML + Allure + JSON)
Multi-Environment (DEV/SIT/UAT/PROD)
Cross-Platform (Windows/macOS/Linux)
Enterprise Integration (CI/CD ready)
This is a production-ready, enterprise-grade testing framework!
Review Changes
1 Tab
Browser
65.0%
Plan, search, build anything


CentricPLM
FRAMEWORK_STRUCTURE.txt
simple_ui_tests.py
Ultimate AI-first testing framework overview
buddy give this complete stack tech buddy what we use that all buddy
CENTRICPLM
reports
api
screenshots
ui
index.html
test_report.html
test_results.json
test_summary.txt
ui_test_report.html
ui_test_results.json
ui_test_summary.txt
index.html
README.md
ui
containers
ui_container.py
features
plm_complete_style_workflow.feature
plm_style_creation.feature
plm_style_properties.feature
framework
dijit_combo.py
logger.py
simple_logger.py
pom
__init__.py
base_page.py
plm_locators.py
plm_login_page.py
plm_style_creation_page.py
plm_style_properties_locators.py
step_definitions
test_data
tests
.cursorrules
.env
UNIVERSAL TESTING FRAMEWORK STRUCTURE
CentricPLM/
UI TESTING (Browser Automation)
ui/features/ → BDD Feature Files
ui/step_definitions/ → BDD Step Implementations
ui/pom/ → Page Object Model
ui/test_data/ → Test Data & Credentials
ui/containers/ → Data Storage
API TESTING (HTTP Requests)
api/features/ → BDD Feature Files
api/step_definitions/ → BDD Step Implementations
api/payloads/ → Request/Response Classes
api/framework/ → HTTP Client
api/test_data/ → API Test Data
api/containers/ → Data Storage
CONFIGURATION
config/global_config.py → Environment & Mode Management
REPORTS (Auto-Generated)
reports/index.html → Universal Index
reports/ui/ → UI Test Reports
reports/api/ → API Test Reports
TEST RUNNERS
generate_reports.py → UI Test Runner
generate_api_reports.py → API Test Runner (Auto-Open)
run_tests.py → Universal Runner
DEPENDENCIES
requirements.txt → Python Packages
HOW TO USE:
UI Testing:
python3 generate_reports.py
Screenshot
DATA MANAGEMENT
StyleData Container - UI test data
UIContainer - UI state management
VariableContainer - API test data
Login Credentials - Authentication data
Test Data Classes - Structured data
WORKFLOW SUPPORT
BDD Scenarios - Business-readable tests
Step Reusability - DRY principle
Data-Driven Testing - Multiple test cases
Parallel Execution - Performance
Cross-Browser Testing - Compatibility
BUDDY, THIS IS THE COMPLETE TECH STACK!
We're using a modern, comprehensive testing framework that supports:
UI Testing (Playwright + POM)
API Testing (HTTPX + Pytest)
BDD Testing (Pytest-BDD + Gherkin)
Professional Reporting (HTML + Allure + JSON)
Multi-Environment (DEV/SIT/UAT/PROD)
Cross-Platform (Windows/macOS/Linux)
Enterprise Integration (CI/CD ready)
This is a production-ready, enterprise-grade testing framework!
Review Changes
1 Tab
Browser
65.0%
Plan, search, build anything


CentricPLM
FRAMEWORK_STRUCTURE.txt
simple_ui_tests.py
Ultimate AI-first testing framework overview
buddy give this complete stack tech buddy what we use that all buddy
CENTRICPLM
ui
framework
logger.py
simple_logger.py
pom
__init__.py
base_page.py
plm_locators.py
plm_login_page.py
plm_style_creation_page.py
plm_style_properties_locators.py
step_definitions
plm_style_properties_steps.py
plm_style_validation_and_navigation_steps.py
plm_ui_steps.py
test_data
login_credentials.json
plm_test_data.py
tests
final_step_by_step_navigation.py
README.md
.cursorrules
.env
.gitignore
API_TEST_CASES_GUIDE.md
CHANGELOG.md
conftest.py
FRAMEWORK_OVERVIEW.md
FRAMEWORK_STRUCTURE.txt
generate_api_reports.py
generate_ui_reports.py
MASTER_TESTING_FRAMEWORK_GUIDE.md
pytest.ini
README.md
requirements.txt
UNIVERSAL TESTING FRAMEWORK STRUCTURE
CentricPLM/
UI TESTING (Browser Automation)
ui/features/ → BDD Feature Files
ui/step_definitions/ → BDD Step Implementations
ui/pom/ → Page Object Model
ui/test_data/ → Test Data & Credentials
ui/containers/ → Data Storage
API TESTING (HTTP Requests)
api/features/ → BDD Feature Files
api/step_definitions/ → BDD Step Implementations
api/payloads/ → Request/Response Classes
api/framework/ → HTTP Client
api/test_data/ → API Test Data
api/containers/ → Data Storage
CONFIGURATION
config/global_config.py → Environment & Mode Management
REPORTS (Auto-Generated)
reports/index.html → Universal Index
reports/ui/ → UI Test Reports
reports/api/ → API Test Reports
TEST RUNNERS
generate_reports.py → UI Test Runner
generate_api_reports.py → API Test Runner (Auto-Open)
run_tests.py → Universal Runner
DEPENDENCIES
requirements.txt → Python Packages
HOW TO USE:
UI Testing:
python3 generate_reports.py
Screenshot
DATA MANAGEMENT
StyleData Container - UI test data
UIContainer - UI state management
VariableContainer - API test data
Login Credentials - Authentication data
Test Data Classes - Structured data
WORKFLOW SUPPORT
BDD Scenarios - Business-readable tests
Step Reusability - DRY principle
Data-Driven Testing - Multiple test cases
Parallel Execution - Performance
Cross-Browser Testing - Compatibility
BUDDY, THIS IS THE COMPLETE TECH STACK!
We're using a modern, comprehensive testing framework that supports:
UI Testing (Playwright + POM)
API Testing (HTTPX + Pytest)
BDD Testing (Pytest-BDD + Gherkin)
Professional Reporting (HTML + Allure + JSON)
Multi-Environment (DEV/SIT/UAT/PROD)
Cross-Platform (Windows/macOS/Linux)
Enterprise Integration (CI/CD ready)
This is a production-ready, enterprise-grade testing framework!
Review Changes
1 Tab
Browser
65.0%
Plan, search, build anything