**DOCUMENTATION OF SMART FACTORY PROJECTS:**

## 1. SmartFactory API Integration Project

### Purpose

The **SmartFactory\_API\_Integration** serves as the backbone of the SmartFactory system, enabling seamless communication between various components, such as product management API and order processing system API. It allows other applications and services to perform operations like adding, updating, retrieving, and deleting product data and orders.

### Tool and Design Choices

* **Language:** C# - The choice of C# with is primarily driven by its strong support for RESTful API development, scalability, and performance
* **Framework:** ASP.NET Core - Selected for its lightweight nature and high performance, making it suitable for building scalable APIs.
* **Testing Framework:** NUnit - Chosen for its ease of use in writing tests

### Project Structure

* **Controllers**: Contains the ProductController and OrderController, which define API endpoints for CRUD operations. Each controller is responsible for handling incoming HTTP requests and returning appropriate responses.
* **Models**: Defines the Product and Order classes, which represent the data structures used in the API. Data annotations are used for validation.
* **Data Access Layer**: Uses Entity Framework Core for data retrieval and manipulation, encapsulating database logic within repositories.
* **Services**: Implements business logic, such as validation and data transformation, separating concerns from the controllers.

### Core Logic

* **API Endpoints**:
  + **Products**: /api/products supports operations like GET (retrieve products), POST (add a new product), PUT (update existing product), and DELETE (remove a product).
  + **Orders**: /api/orders supports similar operations for order management.
* **Data Validation**: Input data is validated using data annotations in the models to ensure integrity before processing.
* **Error Handling**: Global error handling is implemented using middleware, ensuring that errors are logged and standardized error responses are returned to clients.

### Key Testing Coverage

1. **CRUD Operations**: Tests cover all CRUD functionalities for both products and orders, ensuring correct implementation and error handling.
2. **API Response Validation**: Validates that endpoints return expected status codes (200, 201, 404, 400) based on different input scenarios.
3. **Data Integrity**: Tests to ensure that data added via the API is correctly stored and retrieved from the database, confirming data persistence.

#### Running the Project

1. Clone the repository from GitHub.
2. Open the solution in Visual Studio.
3. Restore the NuGet packages.
4. Run the project using dotnet run or through Visual Studio.
5. Execute tests using the command dotnet test.

## 2. SmartFactory UI Integration Project

### Purpose

The SmartFactory\_UI\_Integration provides a user-friendly interface for monitoring and managing factory operations. It allows users to view real-time data on inventory, orders, and system configurations, facilitating easier decision-making and operational management.

### Tool Choices

* **HTML**: Used for Basic webpage for Interactive UI Tests.
* **Robot Framework & SeleniumLibrary**: The keyword-driven approach of Robot Framework simplifies UI test automation. SeleniumLibrary allows for interaction with web elements, validating UI behavior effectively.

### Project Structure

* **HTML Files**: The dashboard.html and config.html files provide the structure for the main dashboard and configuration settings, respectively.
* **Robot File (Tests)**: Organizes automated UI tests using Robot Framework, ensuring coverage of critical user interactions and functionalities.

### Core Logic

* **Dashboard Page**: Displays data from the backend, including current inventory levels, recent order statuses, and graphical representations of data (e.g., charts for inventory trends).
* **Configuration Page**: Allows users to set and save integration parameters like synchronization intervals, impacting how data is synchronized between IT and OT systems.

### Key Testing Coverage

* **UI Navigation Tests:** Validate page loads, button clicks, and alert handling.
* **Input Validation:** Ensure that the configuration page handles invalid inputs and displays appropriate error messages.

#### Running the Project

1. Clone the repository from GitHub
2. Open the Tests Folder Directory
3. Execute UI tests by running the Robot Framework tests through the command line with robot ../SmartFactory\_UI\_Integration/Tests/ui\_tests.robot.

## 3. SmartFactory Data Consistency Project

### Purpose

The SmartFactory\_DataConsistencyCheck focuses on ensuring that data remains accurate and synchronized between the IT and OT systems within the SmartFactory environment. It aims to detect discrepancies and validate the integrity of data after operations.

### Tool Choices

* **C# and .NET Core**: Utilized for backend services that interface with the main API, allowing for efficient data retrieval and comparison logic.
* **NUnit**: Selected for implementing unit tests that validate data consistency, taking advantage of its support for testing asynchronous and synchronous code.

### Project Structure

* **Services**: Contains classes for handling data retrieval from the API and comparing it against expected values. This layer abstracts the logic needed to check data consistency.
* **Tests**: NUnit test classes designed to validate that the data processed through the UI matches what is stored in the backend database.

### Core Logic

* **Data Comparison Logic**: The project retrieves data from the API and performs comparisons against expected values (e.g., verifying that saved synchronization intervals match the expected settings).
* **Validation of Business Rules**: Ensures that business rules related to data integrity (e.g., unique order identifiers, valid product details) are enforced throughout the process.

### Key Testing Coverage

1. **Data Retrieval Validation**: Confirms that the data retrieved from the backend matches user inputs or system actions (e.g., updated synchronization intervals).
2. **Integrity Checks**: Validates that data modifications in one part of the system (UI or API) correctly reflect in the other, ensuring consistency.
3. **Error Handling Tests**: Ensures that the application handles unexpected data states gracefully, logging discrepancies and notifying users as appropriate.

#### Running the Project

1. Clone the repository from GitHub.
2. Open the solution in Visual Studio.
3. Restore the NuGet packages.
4. Run the project using dotnet run or through Visual Studio.
5. Execute tests using the command dotnet test.