### Module 1: User Authentication and Profile Management

1. **Sign In API**

- **Task:** Develop a sign in API for store managers.

- **Sub tasks:**

- Create an endpoint `/signin`.

- Handle POST requests.

- Validate user credentials.

- Generate and return a JWT token.

- **Testing:**

- Unit Tests: Validate input, check JWT token generation.

- Integration Tests: Validate interaction with the user database.

- End-to-End Tests: Verify the entire sign-in process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Follow OWASP guidelines for authentication.

2. **Profile Details API**

- **Task:** Develop an API to fetch profile details.

- **Sub tasks:**

- Create an endpoint `/profile`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return profile details.

- **Testing:**

- Unit Tests: Validate JWT token parsing, profile data fetching.

- Integration Tests: Validate interaction with the profile database.

- End-to-End Tests: Verify the entire profile fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure secure handling of JWT tokens.

3. **Sign Out API**

- **Task:** Develop a sign out API to invalidate the JWT token.

- **Sub tasks:**

- Create an endpoint `/signout`.

- Handle POST requests.

- Invalidate the JWT token.

- **Testing:**

- Unit Tests: Validate token invalidation logic.

- Integration Tests: Validate interaction with the token store.

- End-to-End Tests: Verify the entire sign-out process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Follow secure token invalidation practices.

### Module 2: Dashboard Components

1. **Dashboard Overview**

- **Task:** Develop a dashboard providing an overview of key metrics.

- **Sub tasks:**

- Design the dashboard layout.

- Integrate widgets for real-time data display.

- **Testing:**

- Unit Tests: Validate individual widget data fetching.

- Integration Tests: Validate interaction with data sources.

- End-to-End Tests: Verify the entire dashboard rendering process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Follow responsive design principles.

2. **Order Statistics API**

- **Task:** Develop an API to fetch active and inactive orders.

- **Sub tasks:**

- Create an endpoint `/orders`.

- Handle GET requests with query parameters (type, startDate, endDate).

- Validate JWT token.

- Fetch and return order statistics.

- **Testing:**

- Unit Tests: Validate query parameter parsing, data fetching.

- Integration Tests: Validate interaction with the orders database.

- End-to-End Tests: Verify the entire order statistics fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Optimize query performance.

3. **Shipment Statistics API**

- **Task:** Develop an API to fetch shipment statistics.

- **Sub tasks:**

- Create an endpoint `/shipment`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return shipment statistics.

- **Testing:**

- Unit Tests: Validate data fetching logic.

- Integration Tests: Validate interaction with the shipments database.

- End-to-End Tests: Verify the entire shipment statistics fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure efficient data retrieval.

4. **Online/Offline Data Trends API**

- **Task:** Develop an API to fetch online/offline data trends.

- **Sub tasks:**

- Create an endpoint `/trends`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return data trends.

- **Testing:**

- Unit Tests: Validate trend data fetching logic.

- Integration Tests: Validate interaction with the trends database.

- End-to-End Tests: Verify the entire data trends fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure accurate data representation.

### Module 3: Task and Stock Management

1. **Task Creation API**

- **Task:** Develop an API to create tasks.

- **Sub tasks:**

- Create an endpoint `/tasks`.

- Handle POST requests.

- Validate JWT token.

- Accept task details (taskType, assignee, priorityLevel, description, dueDate, location).

- Save and return task details.

- **Testing:**

- Unit Tests: Validate input data, task creation logic.

- Integration Tests: Validate interaction with the tasks database.

- End-to-End Tests: Verify the entire task creation process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Follow RESTful API design principles.

2. **Stock Creation API**

- **Task:** Develop an API to create stock records.

- **Sub tasks:**

- Create an endpoint `/stock`.

- Handle POST requests.

- Validate JWT token.

- Accept stock details (productId, productName, quantity, dateAdded).

- Save and return stock details.

- **Testing:**

- Unit Tests: Validate input data, stock creation logic.

- Integration Tests: Validate interaction with the stock database.

- End-to-End Tests: Verify the entire stock creation process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure data consistency and integrity.

3. **Retrieve Summary Data API**

- **Task:** Develop an API to retrieve inventory summary data.

- **Sub tasks:**

- Create an endpoint `/inventory/summary`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return summary data (totalStock, lowStock, outOfStock, highDemandOrders).

- **Testing:**

- Unit Tests: Validate data fetching logic.

- Integration Tests: Validate interaction with the inventory database.

- End-to-End Tests: Verify the entire summary data retrieval process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Optimize data retrieval for performance.

### Module 4: Inventory and Order Management

1. **Inventory Orders API**

- **Task:** Develop an API to fetch daily inventory orders.

- **Sub tasks:**

- Create an endpoint `/inventory/orders`.

- Handle GET requests with query parameter `view=daily`.

- Validate JWT token.

- Fetch and return daily order data.

- **Testing:**

- Unit Tests: Validate query parameter parsing, data fetching logic.

- Integration Tests: Validate interaction with the orders database.

- End-to-End Tests: Verify the entire daily order data fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure efficient data retrieval and accurate representation.

2. **Newly Added Stock Information API**

- **Task:** Develop an API to fetch newly added stock information.

- **Sub tasks:**

- Create an endpoint `/inventory/newStock`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return new stock details.

- **Testing:**

- Unit Tests: Validate data fetching logic.

- Integration Tests: Validate interaction with the stock database.

- End-to-End Tests: Verify the entire new stock data fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure data consistency and integrity.

### Module 5: Food Safety Blogs and Articles

1. **Food Blog Section API**

- **Task:** Develop an API to fetch the food blog section.

- **Sub tasks:**

- Create an endpoint `/blogs`.

- Handle GET requests.

- Validate JWT token.

- Fetch and return blog data (id, image, title, description, date).

- **Testing:**

- Unit Tests: Validate data fetching logic.

- Integration Tests: Validate interaction with the blogs database.

- End-to-End Tests: Verify the entire blog data fetching process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure accurate data representation and secure handling.

2. **Yearly View of Articles**

- **Task:** Organize and display articles by year.

- **Sub tasks:**

- Design the layout for yearly view.

- Fetch and categorize articles by year.

- **Testing:**

- Unit Tests: Validate data categorization logic.

- Integration Tests: Validate interaction with the articles database.

- End-to-End Tests: Verify the entire yearly view rendering process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Follow responsive design principles.

3. **Detailed View of Articles**

- **Task:** Provide a detailed view for each article.

- **Sub tasks:**

- Design the layout for detailed view.

- Fetch and display detailed information for each article.

- **Testing:**

- Unit Tests: Validate data fetching logic.

- Integration Tests: Validate interaction with the articles database.

- End-to-End Tests: Verify the entire detailed view rendering process.

- **Code Coverage:** Ensure 98% code coverage.

- **Best Practices:** Ensure accurate data representation and secure handling.