Assignment 2: Explore and Design APIs for OrderService using API Facade Pattern

Objective:

The objective of this assignment is to explore and design APIs for an OrderService using the API Facade Pattern. The OrderService is a critical part of any ecommerce platform and is responsible for managing orders, tracking shipments, and handling returns. The APIs designed for this service should be easy to use, efficient, and secure.

Tasks:

1. Identify the core functionalities of the OrderService and list them down.
2. Design the API facade for the OrderService using the API Facade Pattern.
3. Create a diagram showing the components of the OrderService and how they interact with each other.
4. Document the APIs using OpenAPI Specification (OAS) 3.0.
5. Implement and test the APIs using a suitable programming language and framework.

Deliverables:

1. A document listing down the core functionalities of the OrderService.
2. A document describing the API facade for the OrderService using the API Facade Pattern.
3. A diagram showing the components of the OrderService and how they interact with each other.
4. An OpenAPI Specification (OAS) 3.0 document for the APIs.
5. Source code for the APIs implemented in a suitable programming language and framework.
6. Test results and documentation.

Note: The design of the APIs should follow the REST architectural style, and the APIs should be designed to be scalable, secure, and easy to use.

Solution:

**OrderService API Facade**

**Overview**

The OrderService API Facade is a pattern that allows clients to access multiple microservices that are involved in handling orders through a single, unified API. The facade shields the clients from the complexities of dealing with different APIs and services, providing a simpler and more intuitive interface to work with.

**Requirements**

The OrderService API Facade must provide the following functionality:

* Ability to place a new order by providing the necessary order details (customer information, products, quantities, etc.)
* Ability to retrieve the status of an existing order by its order ID
* Ability to cancel an existing order by its order ID
* **Ability to retrieve the history of orders for a given customer** —-- >> < 250 ms - Cache - 50 ms
* Ability to retrieve the list of products that can be ordered
* Ability to retrieve the details of a specific product by its ID
* Ability to retrieve the inventory levels of a specific product by its ID
* Ability to update the inventory levels of a specific product by its ID

**API Design**

The OrderService API Facade should expose the following endpoints:

* POST /orders - Places a new order by providing the necessary order details
* GET /orders/{**orderId**} - Retrieves the status of an existing order by its order ID **UUID of all IDs**
* DELETE /orders/{orderId} - Cancels an existing order by its order ID
* GET /customers/{customerId}/orders - Retrieves the history of orders for a given customer
* GET /products - Retrieves the list of products that can be ordered
* GET /products/{productId} - Retrieves the details of a specific product by its ID
* GET /products/{productId}/inventory - Retrieves the inventory levels of a specific product by its ID
* PUT /products/{productId}/inventory - Updates the inventory levels of a specific product by its ID

**Architecture**

The OrderService API Facade should be implemented as a standalone microservice that sits between the clients and the actual microservices that handle the different aspects of **order management** (such as customer management, product management, order fulfillment, etc.). The facade should communicate with the underlying microservices via their respective APIs, abstracting away their details and providing a unified interface to the clients.

**Security**

The OrderService API Facade should enforce proper authentication and authorization mechanisms to ensure that only authorized clients can access the APIs and perform the allowed actions. OAuth2 or JWT tokens can be used to achieve this.

**Testing**

The OrderService API Facade should be thoroughly tested using unit tests, integration tests, and end-to-end tests to ensure its functionality and performance. Mocks and stubs can be used to simulate the behavior of the underlying microservices during testing.

**Monitoring**

The OrderService API Facade should be instrumented with proper logging and monitoring mechanisms to track its performance and usage. Metrics such as response times, error rates, and throughput can be monitored to ensure that the API is performing optimally and to detect any issues that may arise.

**Conclusion**

The OrderService API Facade is an effective pattern for simplifying the interaction between clients and microservices and for shielding clients from the complexities of dealing with different APIs and services. By providing a unified and intuitive interface, the facade can improve the overall user experience and increase the efficiency and maintainability of the system.