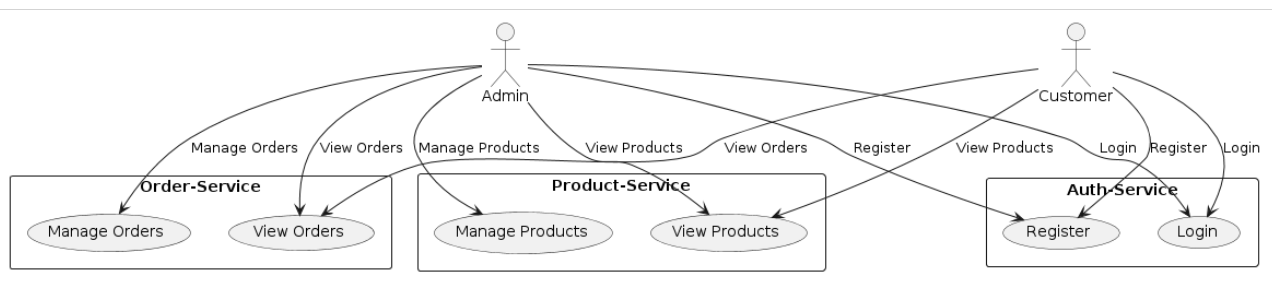
**Advanced Programming**

Assignment 4

Students: Isenov Dias, Kazikhanov Dias, Temen Arnur

1. Use case diagram



* auth-service:  
  User registration and authentication.   
  User login and logout.   
  Authorization checks.
* product-service:   
  Browsing products.   
  Viewing product details.   
  Managing product catalog (add/update/remove products).
* order-service:   
  Placing orders.   
  Viewing order details.   
  Canceling or returning orders.

**Roles:**

* **Customer**: The primary actor who interacts with the system to clothes store.
* **Admin**: An actor who might interact with the system to manage products and services.

1. Architecture choice & its justification

**Clear Architecture**

**Maintainability:** Keeps the codebase manageable and understandable.

**Testability:** The separation of concerns allows for more straightforward testing.

**Scalability:** Minimal impact on existing code while adding new features and services.

**Flexibility:** Adapts to changes in user requirements and technology stacks without significant refactoring.

**Architectural style**

REST provides principles for designing web services, focusing on scalability, simplicity, and performance.

In a microservices architecture, RESTful services are a common way to ensure services are decoupled and can communicate efficiently.

REST does not prescribe a complete system design but rather a style for the interaction between system components

clothing-store-microservices/

├── /auth-service

│ ├── /cmd

│ │ └── main.go

│ ├── /internal

│ │ ├── /domain

│ │ ├── /repository

│ │ ├── /service

│ │ └── /delivery

│ └── /pkg

│

├── /product-service

│ ├── /cmd

│ │ └── main.go

│ ├── /internal

│ │ ├── /domain

│ │ ├── /repository

│ │ ├── /service

│ │ └── /delivery

│ └── /pkg

│

├── /order-service

│ ├── /cmd

│ │ └── main.go

│ ├── /internal

│ │ ├── /domain

│ │ ├── /repository

│ │ ├── /service

│ │ └── /delivery

│ └── /pkg

│

├── /api-gateway

│ ├── /cmd

│ │ └── main.go

│ ├── /internal

│ │ ├── /domain

│ │ ├── /repository

│ │ └── /delivery

│ └── /pkg

│

└── /shared

├── /config

├── /logger

**Project Structure Overview**

**cmd**

* **Purpose:** This directory contains the entry points for the services.
* **Description:** It includes the main files and scripts that initiate the execution of the application.

**internal**

* **Purpose:** Houses the internal application logic and structure.
* **Description:** This directory is further subdivided into:
  + **domain:** Contains the core project logic and entities, representing the business rules and domain objects.
  + **repository:** Includes interfaces and their implementations for data access, managing database interactions.
  + **delivery:** Contains handlers and controllers responsible for handling data delivery to and from the external world.

**pkg**

* **Purpose:** Contains common utilities and packages used across the service.
* **Description:** This directory includes reusable code and libraries that can be utilized by different parts of the application.

**shared**

* **Purpose:** Contains resources shared across multiple services.
* **Description:** This includes configurations, logging utilities, middleware, and other common functionalities that are used across the entire project.

**api-gateway**

* **Purpose:** Acts as a single entry point to the microservices.
* **Description:** This component manages incoming requests, routing them to the appropriate microservice and handling cross-cutting concerns like authentication and load balancing.