YUMMY – ONLINE FOOD ORDERING AND DELIVERY SYSTEM

LAB 6 REPORT – API GATEWAY PATTERN

I. Summary

Lab 6 focuses on the implementation of the **API Gateway Pattern**, a critical component in Microservices Architecture. The Gateway acts as the **single-entry point** for all client requests, routing them to the correct backend services (such as the Food Service) while handling cross-cutting concerns like security. The objective is to centralize security checks (token validation) and request routing to protect internal microservices and simplify client interactions.

II. Technology and tool installation

The API Gateway for the Yummy system is built using the Node.js environment to handle concurrent requests efficiently.

Tool	Purpose	Installation/Setup Guide
Node.js	Server-side execution environment for Gateway code.	Download from nodejs.org
Express.js	Web framework to build endpoints and handle middleware.	npm install express
Axios	HTTP Client to forward requests from Gateway to microservices.	npm install axios
Dotenv	Manage environment variables (URLs, Ports) without hard-coding.	npm install dotenv

Table 6.1. List of supporting tools used in Lab 6

III. Lab specific section: Microservices Decomposition & Communication

1. Project Configuration (.env)

The Gateway is configured to listen on port 5000 and defines target URLs for backend microservices based on previous lab designs.

- PORT=5000
- RESTAURANT_SERVICE_URL=http://localhost:3001/api/restaurants
- USER_SERVICE_URL=http://localhost:3002/api/users

2. Security Check (Security Stub)

A centralized middleware, validateToken, is implemented to intercept requests and verify the Authorization header.

- Unauthorized: Returns 401 if the header is missing or the token is invalid/expired.
- Validation Logic: Only valid-admin-token and valid-user-token are accepted for this lab simulation.

3. Routing Logic (Reverse Proxy)

The Gateway acts as a Reverse Proxy, receiving requests from the Flutter app and forwarding them to the internal network.

- Request Forwarding: Every request starting with /api/foods is proxied to the Food Service.
- Dynamic Mapping: The Gateway maps the sub-path from the incoming request to the target backend URL dynamically.

4. Resilience and Error Handling

To ensure system stability, the Gateway handles connection errors if a backend service is offline.

- If the Food Service (port 3001) fails to respond, the Gateway returns a 503 Service Unavailable status with a structured error message.

IV. Activity Practice: Testing the Gateway

1. Test 1: Unauthorized Access

- Action: Call the API without a token.
- Result: 401 Unauthorized. The Gateway successfully blocked the request before it reached the backend.

2. Test 2: Authorized Access (Success)

- Action: Access with valid-user-token.
- Result: 200 OK. The Gateway identified the token and correctly routed the request to the Food Service.

3. Test 3: Forbidden Access (Admin Check)

- Action: Attempt a POST request using a regular user token.
- Result: 403 Forbidden. The Gateway prevented a non-admin user from performing modification operations.

4. Test 4: Service Failure Handling

- Action: Shut down the Food Service and send a GET request.
- Result: 503 Service Unavailable. The Gateway correctly handled the service failure using a try...catch block.

V. Conclusion

The implementation of the API Gateway Pattern for the Yummy application successfully provides a unified access point. By centralizing security (Stub for token checking) and routing (Reverse Proxy via axios), the system reduces the complexity of individual microservices while enhancing overall resilience against backend failures.