**i-Transform – On-Demand Car Wash**

**Low Level Design (LLD)**

**1. Project Overview**

i-Transform is a microservices-based car wash platform that allows customers to book premium car wash services at their preferred time and location. Washers (service providers) receive wash requests, perform the service, and receive payments. Admins manage users, bookings, and monitor service quality.

The system is built using Spring Boot + Spring Cloud ecosystem, and follows best practices like Eureka service discovery, RabbitMQ messaging, and JWT-based authentication.

**StakeHolders**

**1. Customers**

* **Book car washes**
* **Choose washer and schedule time**
* **Make payments**
* **Rate the service**

**2. Washers**

* **Accept or reject wash requests**
* **View schedule and customer details**
* **Update wash status (e.g., started, completed)**

**3. Admin**

* **Manage customers and washers**
* **Monitor ongoing and completed washes**
* **Generate reports and analytics**
* **Handle user issues or feedback**

**4. Developers / DevOps Team**

* **Maintain codebase and CI/CD pipelines**
* **Ensure microservices are up and discoverable (via Eureka)**
* **Monitor RabbitMQ queues, logs, databases, etc.**

**5. Frontend/UI Designers**

* **Create user interfaces for web/mobile**
* **Ensure seamless experience across user roles**

**Optional: QA/Testers**

* **Ensure functionality, performance, and security are intact across all services**

**2. Microservices Overview**

| **Service Name** | **Responsibilities** |
| --- | --- |
| customer-service | Customer registration, login, car management, booking history |
| washer-service | Washer registration, profile, accepting jobs, updating status |
| admin-service | Admin panel, user & washer management, report generation |
| booking-service | Wash now/schedule later, assign washers, manage order status |
| payment-service | Payment processing and invoice handling |
| notification-service | Sends async notifications via RabbitMQ |
| eureka-server | Registers and discovers microservices |
| api-gateway | Routes client requests to appropriate services |

**3. Technology Stack**

* **Backend**: Java, Spring Boot, Spring Cloud, Spring Data JPA
* **Databases**: MySQL / MongoDB / H2
* **Message Broker**: RabbitMQ
* **Authentication**: Spring Security + JWT
* **Service Discovery**: Eureka
* **Routing**: Spring Cloud Gateway
* **Frontend**: Angular
* **Testing**: JUnit, Mockito
* **Dev Tools**: Postman, Swagger, Maven

**4. User Flow**

**Customer Flow**

1. Register/Login
2. Add cars and payment info
3. Book or schedule a car wash
4. Receive updates via notifications
5. View receipts and give ratings

**Washer Flow**

1. Login/Register
2. Receive wash requests
3. Accept/reject bookings
4. Update wash status
5. View profile and ratings

**Admin Flow**

1. Manage customers and washers
2. Activate/deactivate users
3. Assign pending requests
4. Generate reports
5. Manage service plans, add-ons, promos

**5. Service Interaction Flow**

Customer → Booking-Service → Notification (via RabbitMQ)

Customer → Booking-Service → Washer-Service

Washer → Booking-Service → Update status

Admin → All services

All requests → API Gateway → Eureka → Respective microservice

**6. Entity Design Overview**

**Customer**

* id, name, email, phone, password, address, cars

**Washer**

* id, name, email, phone, rating, status, location

**Booking**

* id, customerId, washerId, package, time, status, addons

**Payment**

* id, bookingId, amount, status, receipt

**7. Security Design**

* JWT issued on login via auth-service or inside customer-service
* API Gateway validates tokens before forwarding
* Role-based authorization in all services using Spring Security

**8. Sample API Endpoints**

**Customer Service**

* POST /customers/register
* GET /customers/{id}
* PUT /customers/{id}

**Washer Service**

* POST /washers/register
* GET /washers/{id}
* PUT /washers/update-status

**Booking Service**

* POST /bookings/create
* GET /bookings/customer/{id}
* PUT /bookings/assign

**Notification Service (RabbitMQ)**

* Queue: wash-notify
* Listens to: booking.created, booking.completed, payment.done

**9. Testing Strategy**

| **Tool** | **Purpose** |
| --- | --- |
| JUnit | Unit test service logic |
| Mockito | Mock repository/service calls |
| Postman | API Testing (CRUD + Auth) |

**10. Future Enhancements**

* Real-time washer tracking
* Integration with SMS gateways (Twilio)
* Admin analytics dashboard
* OTP verification for bookings
* Stripe/PayPal integration

**11. Architecture Diagram**



