**Vehicle Service Management System (VSMS) - Design Document**

**1. DOCUMENT CONTROL**

|  |  |
| --- | --- |
| **Item** | **Details** |
| **Project Name** | Vehicle Service Management System (VSMS) - GearUp |
| **Version** | 1.0 |
| **Author** | Srihari Eetyala |
| **Date** | December 29, 2025 |
| **Status** | Final |

**2. PURPOSE OF THE DOCUMENT**

This document describes the system architecture, design decisions, component structure, APIs, data models, and non-functional aspects of the Vehicle Service Management System (VSMS).

**It is intended for:**

* Developers
* Reviewers
* Interview discussions
* Maintenance & enhancement planning
* Stakeholder presentations

**3. SYSTEM OVERVIEW**

**Business Objective**

Provide a scalable, secure, and maintainable system to:

* Manage vehicle service operations end-to-end
* Handle customer vehicle registration and service booking
* Assign technicians and manage service bays efficiently
* Track parts inventory and manage part requests
* Generate invoices and process payments
* Send automated email notifications

**High-Level Features**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **User Management** | Multi-role authentication (Customer, Technician, Manager, Inventory Manager, Admin) |
| **Vehicle Registration** | Customers can register and manage their vehicles |
| **Service Booking** | Book service appointments with pickup options |
| **Technician Assignment** | Auto/manual assignment with workload balancing |
| **Bay Management** | 10 service bays with real-time status tracking |
| **Inventory Management** | Parts catalog, stock tracking, low-stock alerts |
| **Part Requests** | Technicians request parts, managers approve |
| **Invoicing & Payments** | Generate invoices, accept CASH/CARD/UPI payments |
| **Email Notifications** | Automated emails via RabbitMQ messaging |
| **Analytics & Reports** | Revenue reports, service statistics, technician performance |

**4. ARCHITECTURE OVERVIEW (HLD)**

**Architecture Style**

* **Microservices Architecture**
* **REST-based communication**
* **Event-driven messaging (RabbitMQ)**
* **API Gateway pattern with JWT authentication**
* **Service Discovery (Netflix Eureka)**
* **Centralized Configuration (Spring Cloud Config)**
* **Containerized deployment (Docker**



**Core Components**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Component** | **Port** | **Purpose** |
| 1 | Angular Frontend | 4200 | User Interface (SPA) |
| 2 | API Gateway | 8080 | Single entry point, JWT validation, routing |
| 3 | Service Registry | 8761 | Netflix Eureka - Service Discovery |
| 4 | Config Server | 8888 | Centralized Configuration Management |
| 5 | Auth Service | 8081 | Authentication, User Management, JWT |
| 6 | Vehicle Service | 8082 | Vehicle Registration & Management |
| 7 | Service Request Service | 8083 | Service Requests, Invoices, Payments |
| 8 | Inventory Service | 8084 | Parts & Inventory Management |
| 9 | Notification Service | 8086 | Email Notifications via RabbitMQ |

**5. TECHNOLOGY STACK**

**Backend**

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **Purpose** |
| Java | 21 | Programming Language |
| Spring Boot | 3.4.1 | Application Framework |
| Spring Cloud | 2024.0.0 | Microservices Framework |
| Spring Security | 6.x | Security & Authentication |
| Spring Data JPA | 3.x | Data Persistence |
| Netflix Eureka | - | Service Discovery |
| Spring Cloud Config | - | Centralized Configuration |
| Spring Cloud Gateway | - | API Gateway |
| Resilience4j | - | Circuit Breaker |
| OpenFeign | - | Inter-service Communication |
| JWT (jwt) | 0.12.3 | Token-based Authentication |

**Frontend**

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **Purpose** |
| Angular | 20.3.0 | Frontend Framework |
| TypeScript | 5.9.2 | Programming Language |
| RxJS | 7.8.0 | Reactive Programming |
| Bootstrap | 5.3.8 | CSS Framework |
| Bootstrap Icons | 1.13.1 | Icon Library |
| Chart.js | 4.5.1 | Data Visualization |

**Database**

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **Purpose** |
| PostgreSQL | 15 | Relational Database (one DB per service) |

**Messaging**

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **Purpose** |
| RabbitMQ | 3 | Async Message Broker for Notifications |

**DevOps & Infrastructure**

|  |  |
| --- | --- |
| **Technology** | **Purpose** |
| Docker | Containerization |
| Docker Compose | Multi-container Orchestration |
| Jenkins | CI/CD Pipeline |
| SonarQube | Code Quality Analysis |
| JaCoCo | Code Coverage |

**Testing**

|  |  |
| --- | --- |
| **Technology** | **Purpose** |
| JUnit 5 | Unit Testing |
| Mockito | Mocking Framework |
| Spring Boot Test | Integration Testing |

**6. MICROSERVICES DESIGN**

**6.1 Auth Service (Port: 8081)**

**Responsibilities:**

* User registration (Customer, Technician)
* Authentication & JWT token generation
* User management (CRUD operations)
* Role-based access control
* Technician approval workflow

**Database:** vsms\_auth\_db

**APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/auth/login | Public | User login, returns JWT |
| GET | /api/auth/me | Authenticated | Get current user info |
| POST | /api/auth/logout | Authenticated | User logout |
| PUT | /api/auth/change-password | Authenticated | Change password |

**Customer APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/customers | Public | Customer registration |
| GET | /api/customers | ADMIN | Get all customers |
| GET | /api/customers/{id} | Owner/MANAGER/ADMIN | Get customer by ID |
| PUT | /api/customers/{id} | Owner/ADMIN | Update customer |
| DELETE | /api/customers/{id} | Owner/ADMIN | Delete customer |

**Technician APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/technicians | Public | Technician registration |
| GET | /api/technicians | MANAGER/ADMIN | Get all technicians |
| GET | /api/technicians/{id} | TECHNICIAN/MANAGER/ADMIN | Get by ID |
| GET | /api/technicians/available | MANAGER/ADMIN | Get available technicians |
| GET | /api/technicians/specialization/{spec} | MANAGER/ADMIN | Filter by specialization |
| GET | /api/technicians/pending | ADMIN | Get pending approvals |
| PUT | /api/technicians/{id}/review | ADMIN | Approve/Reject technician |
| PUT | /api/technicians/{id}/duty | Owner/MANAGER | Toggle duty status |
| DELETE | /api/technicians/{id} | Owner/ADMIN | Delete technician |

**Manager APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/managers | ADMIN | Create manager |
| GET | /api/managers | ADMIN | Get all managers |
| PUT | /api/managers/{id} | Owner/ADMIN | Update manager |
| DELETE | /api/managers/{id} | ADMIN | Delete manager |

**Sample Payloads:**

**Login Request:**

{

"email": "customer@vsms.com",

"password": "password123"

}

**Login Response:**

{

"accessToken": "eyJhbGciOiJIUzI1NiIs...",

"userId": 1,

"email": "customer@vsms.com",

"role": "CUSTOMER",

"firstName": "Rahul",

"lastName": "Sharma"

}

**Customer Registration:**

{

"email": "newcustomer@email.com",

"password": "securePassword123",

"phone": "9876543210",

"firstName": "John",

"lastName": "Doe",

"address": "123 Main Street",

"city": "Bangalore",

"state": "Karnataka",

"zipCode": "560001"

}

**6.2 Vehicle Service (Port: 8082)**

**Responsibilities:**

* Vehicle registration and management
* Associate vehicles with customers
* Vehicle CRUD operations

**Database:** vsms\_vehicle\_db

**APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/vehicles | CUSTOMER | Create vehicle |
| GET | /api/vehicles/{id} | Owner/MANAGER/ADMIN | Get vehicle by ID |
| GET | /api/vehicles/customer/{customerId} | Owner/MANAGER/ADMIN | Get customer vehicles |
| PUT | /api/vehicles/{id} | Owner | Update vehicle |
| DELETE | /api/vehicles/{id} | Owner | Delete vehicle |

**Sample Payloads:**

**Create Vehicle Request:**

{

"plateNumber": "KA01AB1234",

"brand": "Maruti Suzuki",

"model": "Swift",

"year": 2020,

"fuelType": "PETROL",

"vehicleType": "FOUR\_WHEELER"

}

**Vehicle Response:**

{

"id": 1,

"customerId": 1,

"plateNumber": "KA01AB1234",

"brand": "Maruti Suzuki",

"model": "Swift",

"year": 2020,

"fuelType": "PETROL",

"vehicleType": "FOUR\_WHEELER",

"createdAt": "2026-01-05T10:00:00"

}

**6.3 Service Request Service (Port: 8083)**

**Responsibilities:**

* Service request creation and management
* Technician and bay assignment
* Service status tracking
* Invoice generation
* Payment processing

**Database:** vsms\_service\_db

**Configuration:**

service.total-bays=10

**Service Request APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/service-requests | CUSTOMER | Create service request |
| GET | /api/service-requests/{id} | All authenticated | Get request by ID |
| GET | /api/service-requests/customer/{id} | Owner/MANAGER/ADMIN | Get by customer |
| GET | /api/service-requests/vehicle/{id}/history | CUSTOMER/MANAGER | Vehicle history |
| GET | /api/service-requests | MANAGER/ADMIN | Get all requests |
| GET | /api/service-requests/technician/{id} | TECHNICIAN/MANAGER | Get by technician |
| PUT | /api/service-requests/{id}/assign | MANAGER/ADMIN | Assign tech + bay |
| PUT | /api/service-requests/{id}/status | All authenticated | Update status |
| PUT | /api/service-requests/{id}/set-pricing | MANAGER | Set final pricing |
| PUT | /api/service-requests/{id}/cancel | Owner | Cancel request |
| PUT | /api/service-requests/{id}/reschedule | Owner | Reschedule |
| GET | /api/service-requests/stats | MANAGER/ADMIN | Dashboard statistics |
| GET | /api/service-requests/bays | MANAGER/ADMIN | Get all bay status |
| GET | /api/service-requests/bays/available | MANAGER/ADMIN | Get available bays |

**Invoice APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/invoices/generate/{serviceRequestId} | MANAGER/ADMIN | Generate invoice |
| GET | /api/invoices | MANAGER/ADMIN | Get all invoices |
| GET | /api/invoices/customer/{id} | Owner | Get my invoices |
| GET | /api/invoices/unpaid | MANAGER/ADMIN | Get unpaid invoices |
| PUT | /api/invoices/{id}/pay | Owner | Pay invoice |
| GET | /api/invoices/stats | MANAGER/ADMIN | Revenue statistics |

**Sample Payloads:**

**Create Service Request:**

{

"vehicleId": 1,

"serviceType": "REGULAR\_SERVICE",

"description": "Regular service and oil change",

"priority": "NORMAL",

"preferredDate": "2026-01-10T10:00:00",

"pickupRequired": true,

"pickupAddress": "123 MG Road, Bangalore"

}

**Assign Technician & Bay:**

{

"technicianId": 1,

"bayNumber": 3

}

**Pay Invoice:**

{

"paymentMethod": "UPI"

}

**6.4 Inventory Service (Port: 8084)**

**Responsibilities:**

* Parts catalog management
* Stock tracking and updates
* Low-stock alerts
* Part request workflow

**Database:** vsms\_inventory\_db

**Configuration:**

inventory.low-stock-threshold=10

**Part APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/parts | MANAGER/INVENTORY\_MANAGER | Add new part |
| GET | /api/parts | TECHNICIAN/MANAGER/INVENTORY\_MANAGER | Get all parts |
| GET | /api/parts/{id} | TECHNICIAN/MANAGER/INVENTORY\_MANAGER | Get part by ID |
| GET | /api/parts/low-stock | MANAGER/INVENTORY\_MANAGER | Get low stock |
| PUT | /api/parts/{id} | MANAGER/INVENTORY\_MANAGER | Update part |

**Part Request APIs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Endpoint** | **Access** | **Description** |
| POST | /api/part-requests | TECHNICIAN | Create part request |
| GET | /api/part-requests/pending | MANAGER/INVENTORY\_MANAGER | Get pending |
| GET | /api/part-requests/technician/{id} | TECHNICIAN | Get by technician |
| GET | /api/part-requests/service/{id}/total-cost | MANAGER | Get parts cost |
| PUT | /api/part-requests/{id}/approve | MANAGER/INVENTORY\_MANAGER | Approve |
| PUT | /api/part-requests/{id}/reject | MANAGER/INVENTORY\_MANAGER | Reject |

**Sample Payloads:**

**Create Part:**

{

"partNumber": "ENG-OIL-5W30",

"name": "Engine Oil 5W30",

"description": "Fully synthetic engine oil 5L",

"category": "FLUIDS",

"quantity": 50,

"unitPrice": 1500.00,

"reorderLevel": 10

}

**Create Part Request:**

{

"partId": 1,

"serviceRequestId": 3,

"requestedQuantity": 2,

"notes": "Required for oil change"

}

**6.5 Notification Service (Port: 8086)**

**Responsibilities:**

* Email notification sending
* Event-driven messaging via RabbitMQ
* Email templates for various events

**Database:** None (event-driven)

**RabbitMQ Queues & Events:**

|  |  |  |
| --- | --- | --- |
| **Queue** | **Event** | **Trigger** |
| vsms.customer.welcome.queue | CustomerWelcomeEvent | Customer registration |
| vsms.manager.created.queue | ManagerCreatedEvent | Manager creation |
| vsms.technician.approved.queue | TechnicianApprovedEvent | Technician approval |
| vsms.technician.rejected.queue | TechnicianRejectedEvent | Technician rejection |
| vsms.service.completed.queue | ServiceCompletedEvent | Service completion |
| vsms.invoice.generated.queue | InvoiceGeneratedEvent | Invoice generation |
| vsms.invoice.paid.queue | InvoicePaidEvent | Payment received |

**Event Payload Examples:**

**CustomerWelcomeEvent:**

{

"customerName": "Rahul Sharma",

"email": "customer@vsms.com"

}

**InvoiceGeneratedEvent:**

{

"customerEmail": "customer@vsms.com",

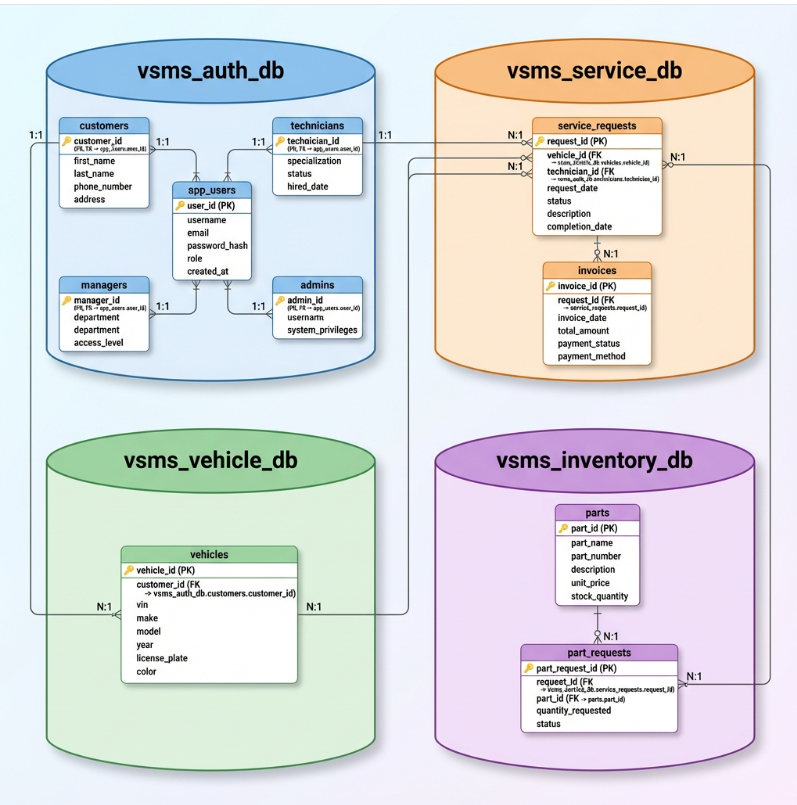
"customerName": "Rahul Sharma",

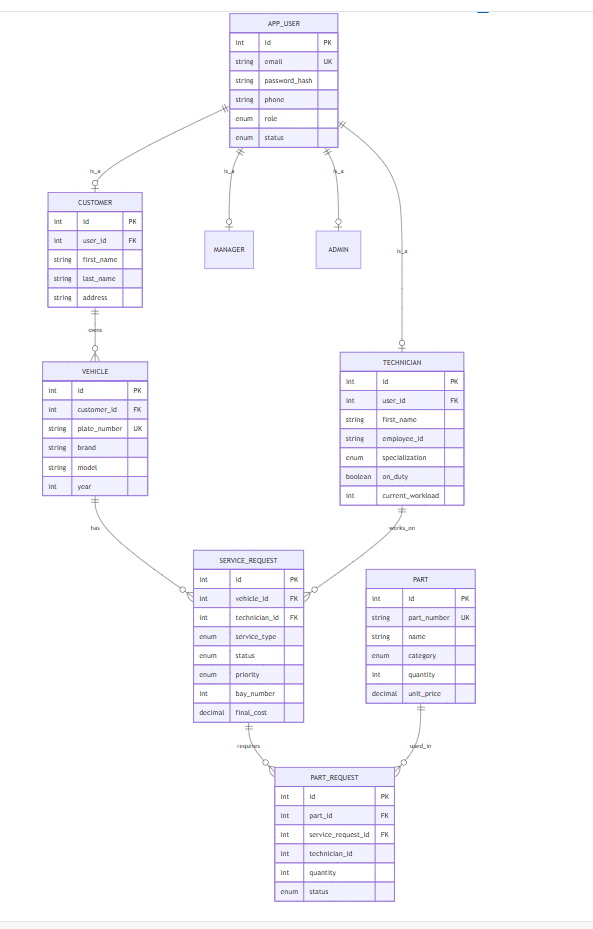
"invoiceNumber": "INV-2026-001",

"totalAmount": 5500.00

}

**7. DATA DESIGN (LLD)**





**Validation Rules**

**Customer Registration:**

|  |  |
| --- | --- |
| **Field** | **Validation** |
| email | Required, Valid email format, Unique |
| password | Required, Min 8 characters |
| phone | Required, 10 digits |
| firstName | Required, 2-50 characters |
| lastName | Required, 2-50 characters |

**Vehicle Registration:**

|  |  |
| --- | --- |
| **Field** | **Validation** |
| plateNumber | Required, Indian format (XX00XX0000), Unique |
| brand | Required |
| model | Required |
| year | Required, 1900-current year |
| fuelType | Required, Valid enum |
| vehicleType | Required, Valid enum |

**Service Request:**

|  |  |
| --- | --- |
| **Field** | **Validation** |
| vehicleId | Required, Must belong to customer |
| serviceType | Required, Valid enum |
| description | Optional, Max 1000 chars |
| priority | Default NORMAL |
| preferredDate | Required, Future date |

**9. ERROR HANDLING STRATEGY**

**Global Exception Handling**

All services implement centralized exception handling using @ControllerAdvice:

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<ErrorResponse> handleNotFound(ResourceNotFoundException ex) {

return ResponseEntity.status(404).body(new ErrorResponse(404, ex.getMessage()));

}

@ExceptionHandler(ValidationException.class)

public ResponseEntity<ErrorResponse> handleValidation(ValidationException ex) {

return ResponseEntity.status(400).body(new ErrorResponse(400, ex.getMessage()));

}

@ExceptionHandler(UnauthorizedException.class)

public ResponseEntity<ErrorResponse> handleUnauthorized(UnauthorizedException ex) {

return ResponseEntity.status(401).body(new ErrorResponse(401, ex.getMessage()));

}

}

**Exception Types by Service**

|  |  |
| --- | --- |
| **Service** | **Exception Types** |
| Auth Service | UserNotFoundException, InvalidCredentialsException, DuplicateEmailException |
| Vehicle Service | VehicleNotFoundException, DuplicatePlateException, UnauthorizedAccessException |
| Service Request | ServiceRequestNotFoundException, BayNotAvailableException, InvalidStatusTransitionException |
| Inventory | PartNotFoundException, InsufficientStockException, DuplicatePartNumberException |

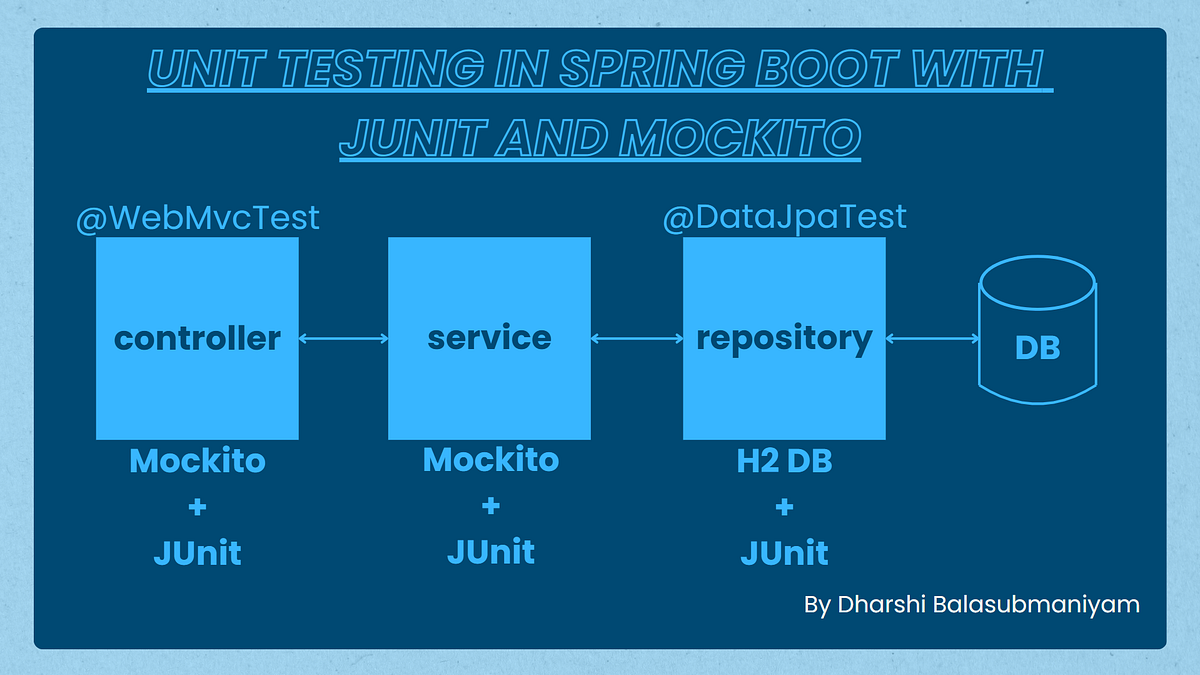
**HTTP Status Codes**

|  |  |  |
| --- | --- | --- |
| **Code** | **Meaning** | **Usage** |
| 200 | OK | Successful GET/PUT |
| 201 | Created | Successful POST |
| 204 | No Content | Successful DELETE |
| 400 | Bad Request | Validation errors |
| 401 | Unauthorized | Missing/invalid token |
| 403 | Forbidden | Insufficient permissions |
| 404 | Not Found | Resource doesn't exist |
| 409 | Conflict | Duplicate resource |
| 500 | Internal Error | Server error |

**10. SECURITY DESIGN**

****

**2. TESTING STRATEGY**



**Backend**

* Unit tests (Service layer)
* Controller tests (MockMvc)
* Minimum 90% coverage

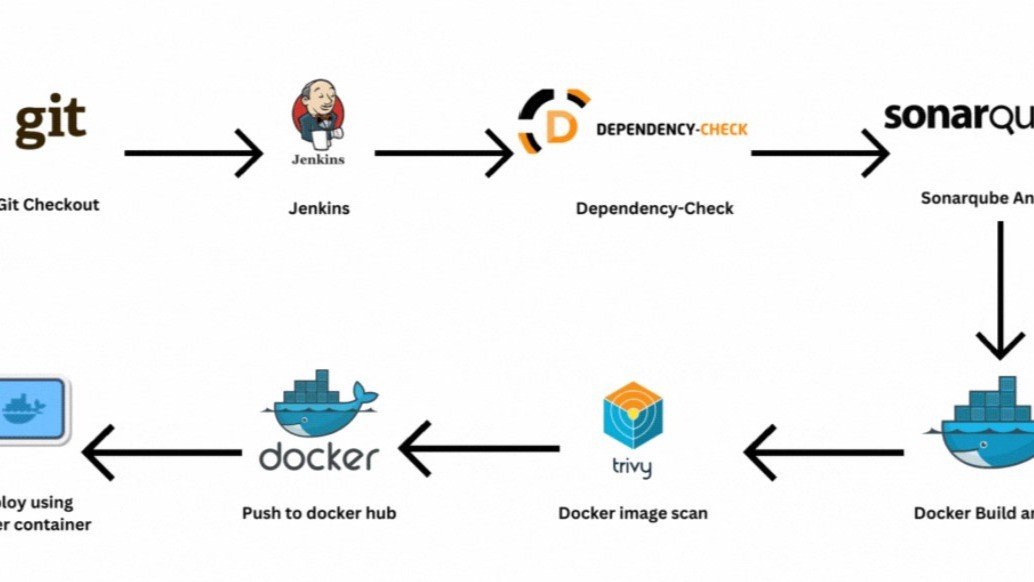
**Frontend**

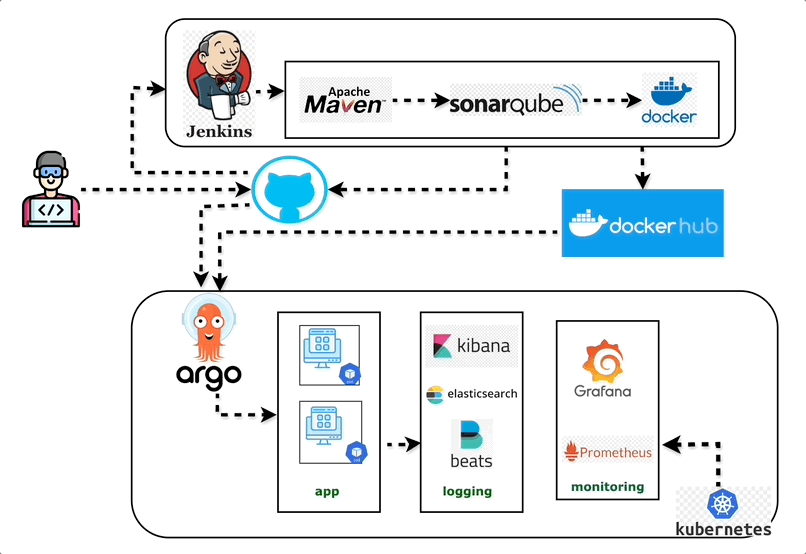
* Component tests
* Service tests

**Quality Gates**

* SonarQube enforced
* Build fails on violations

**13. CI/CD DESIGN**





**Pipeline Flow**

1. Git Commit
2. Jenkins Build
3. Unit Tests
4. SonarQube Scan
5. Quality Gate Check
6. Docker Build
7. Docker Compose Deploy

**14. DEPLOYMENT DESIGN**

* Docker image per microservice
* docker-compose for orchestration
* Environment-specific configs

**15. ASSUMPTIONS & CONSTRAINTS**

**Assumptions**

* Services communicate over REST
* MongoDB available
* Docker environment present

**Constraints**

* No distributed transactions
* Event-driven architecture out of scope

**16. FUTURE ENHANCEMENTS**

* Spring Cloud Gateway
* Kafka-based async communication
* Kubernetes deployment
* Centralized logging (ELK)

**17. CONCLUSION**

This design ensures:  
✔ Clean separation of concerns  
✔ Scalability & maintainability  
✔ Testability & CI/CD readiness  
✔ Interview-ready explanation

**SEQUENCE DIAGRAMS — VEHICLE SERVICE MANAGEMENT SYSTEM**

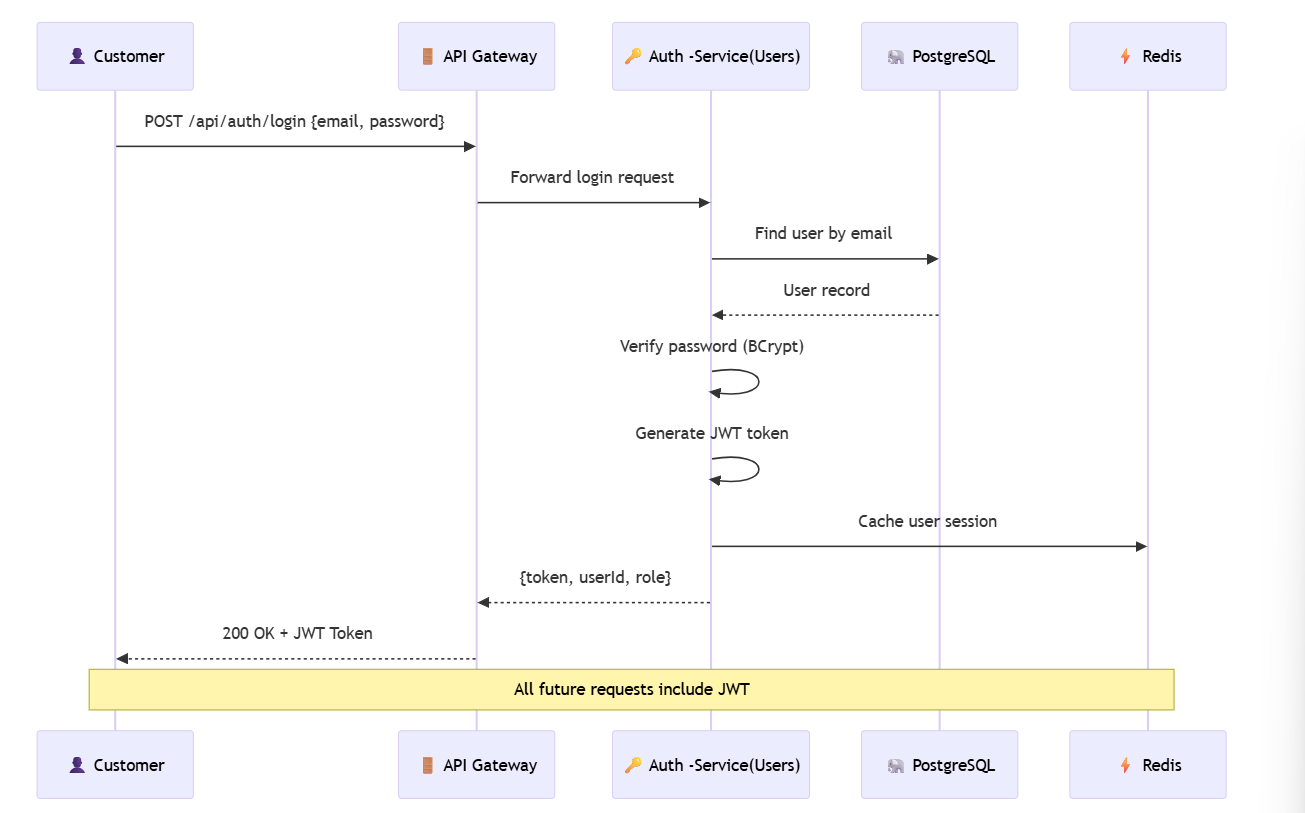
**1. User Authentication Flow**

**Scenario**

A registered user logs in using the Angular UI to access the system.

**Flow**

1. User enters email and password in Angular UI
2. Angular sends POST /api/auth/login to API Gateway
3. Gateway forwards request to Auth Service
4. Auth Service queries PostgreSQL for user by email
5. Password is verified using BCrypt
6. JWT token is generated with userId, role, expiry
7. Session is cached in Redis
8. Token is returned to Angular UI



**Participants**

|  |  |
| --- | --- |
| **Participant** | **Role** |
| Customer | End user logging in |
| API Gateway | Routes request, will validate JWT on future calls |
| Auth Service | Handles authentication logic |
| PostgreSQL | Stores user credentials |
| Redis | Caches session for performance |

**Key Design Points**

* Password hashing with BCrypt (never stored in plain text)
* JWT contains claims: userId, role, expiry
* Gateway validates JWT signature on every request
* Redis cache reduces database calls for session validation
* No direct database access from frontend

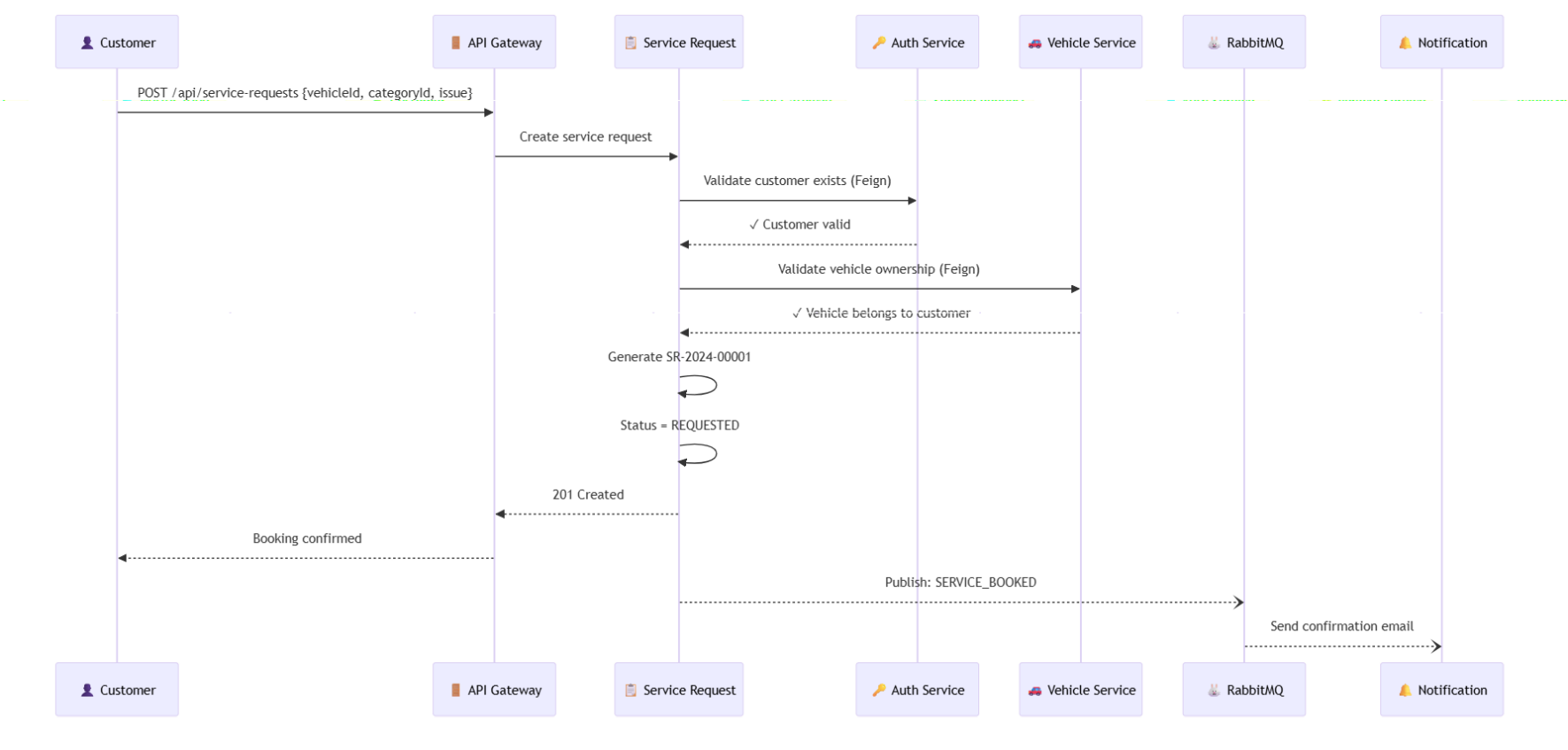
**2. Service Request Booking Flow— Sequence Diagram**

**Scenario**

A new user registers using the Angular UI.

**Flow**

1. User enters details in Angular UI
2. Angular sends POST /users/register to User Service
3. User Service validates request (email, password)
4. User Service checks email uniqueness in MongoDB
5. Password is encrypted
6. User is saved in MongoDB
7. Success response sent back to UI



**Participants**

|  |  |
| --- | --- |
| Participant | Role |
| Customer | Books the service |
| API Gateway | JWT validation + routing |
| Service Request | Core booking logic |
| Auth Service | Validates customer exists |
| Vehicle Service | Validates vehicle ownership |
| RabbitMQ | Async event messaging |
| Notification | Sends confirmation email |

**Key Design Points**

* Cross-service validation using OpenFeign clients
* Vehicle ownership verified before booking
* Request number auto-generated with pattern SR-YYYY-NNNNN
* Async notification doesn't block the response
* Event-driven architecture for loose coupling

**2. Technician Assignment Flow— Sequence Diagram**

**Scenario**

A manager assigns an available technician to a pending service request.

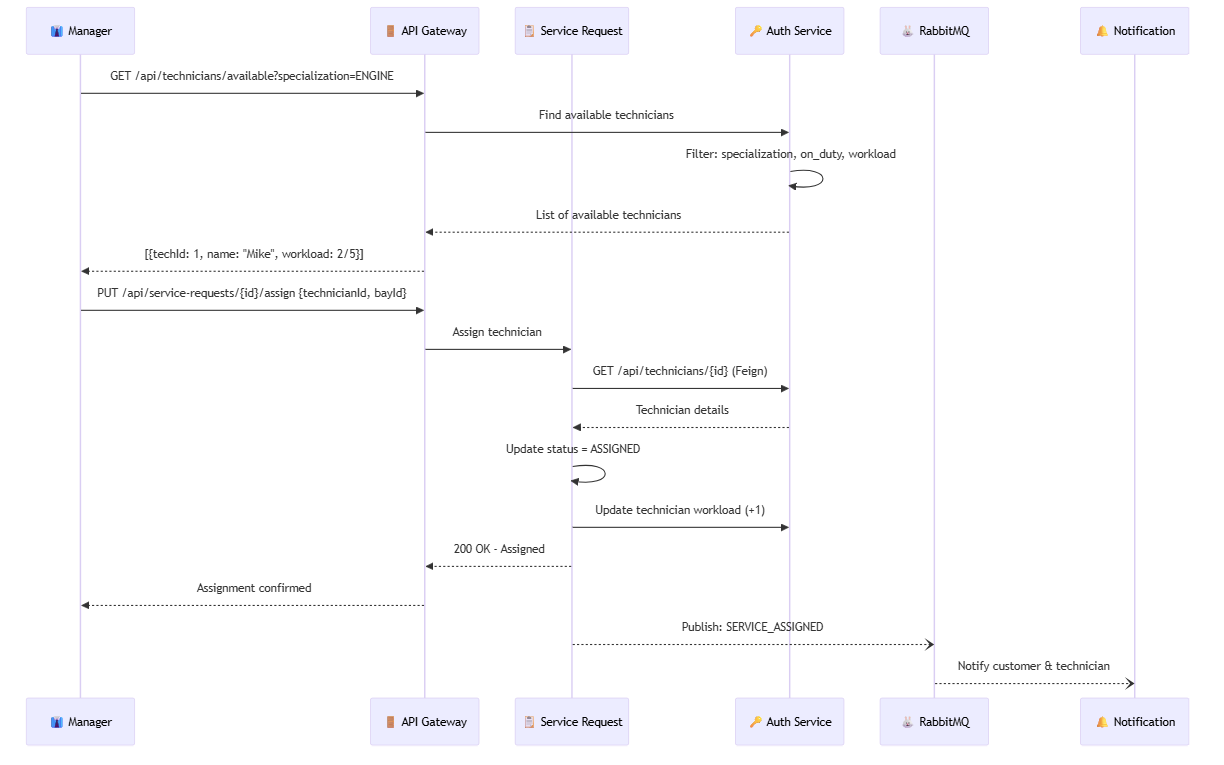
**Flow**

1. Manager views pending service requests in Angular UI
2. Manager clicks "Assign" on a request
3. Angular fetches available technicians filtered by specialization
4. Manager selects technician and service bay
5. Angular sends PUT /api/service-requests/{id}/assign to Gateway
6. Service Request calls Auth Service to get technician details
7. Service request is updated with technicianId, bayId, status=ASSIGNED
8. Auth Service updates technician workload (+1)
9. SERVICE\_ASSIGNED event published to RabbitMQ
10. Notification Service notifies both customer and technician

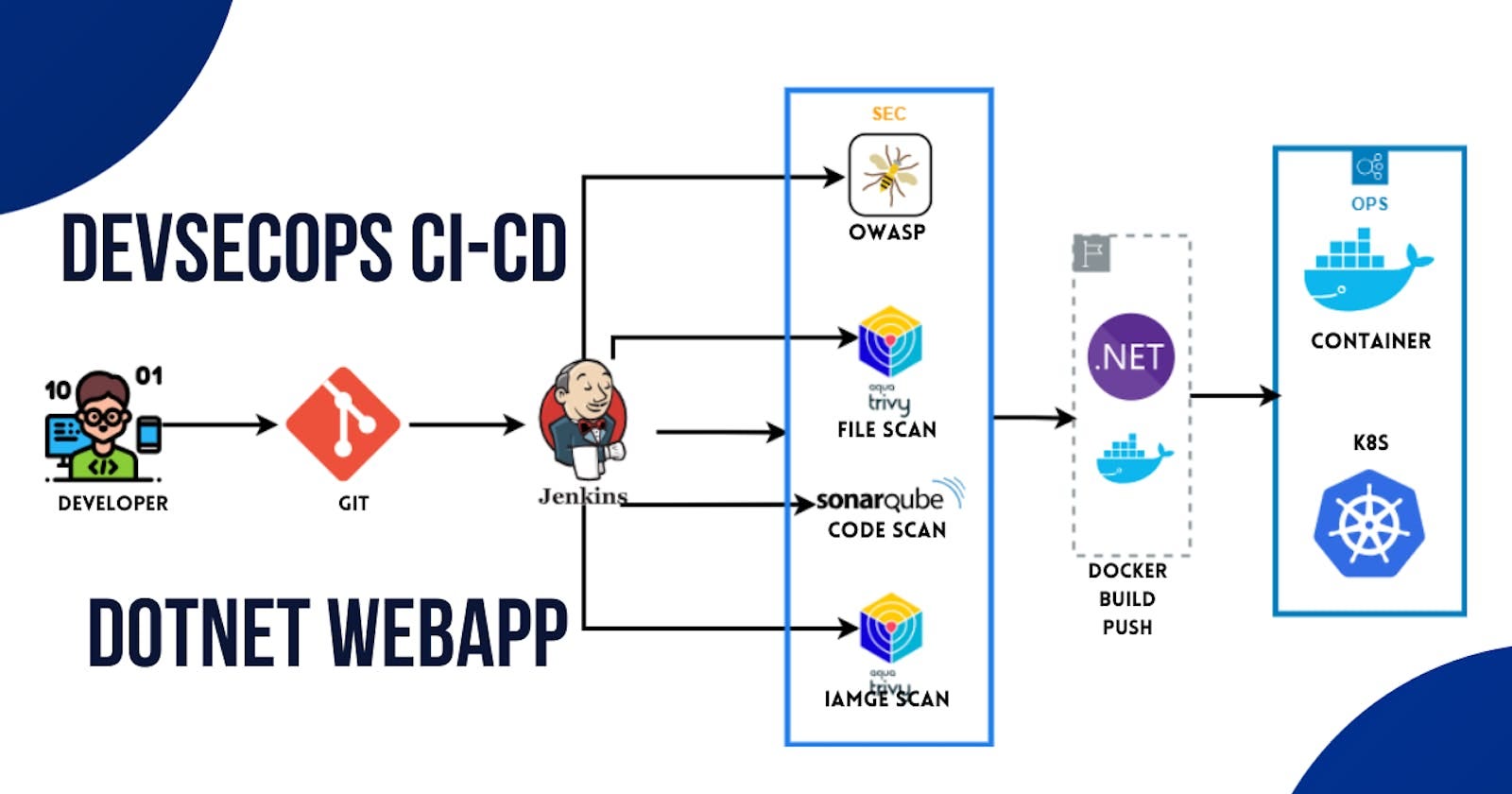
**Participants**

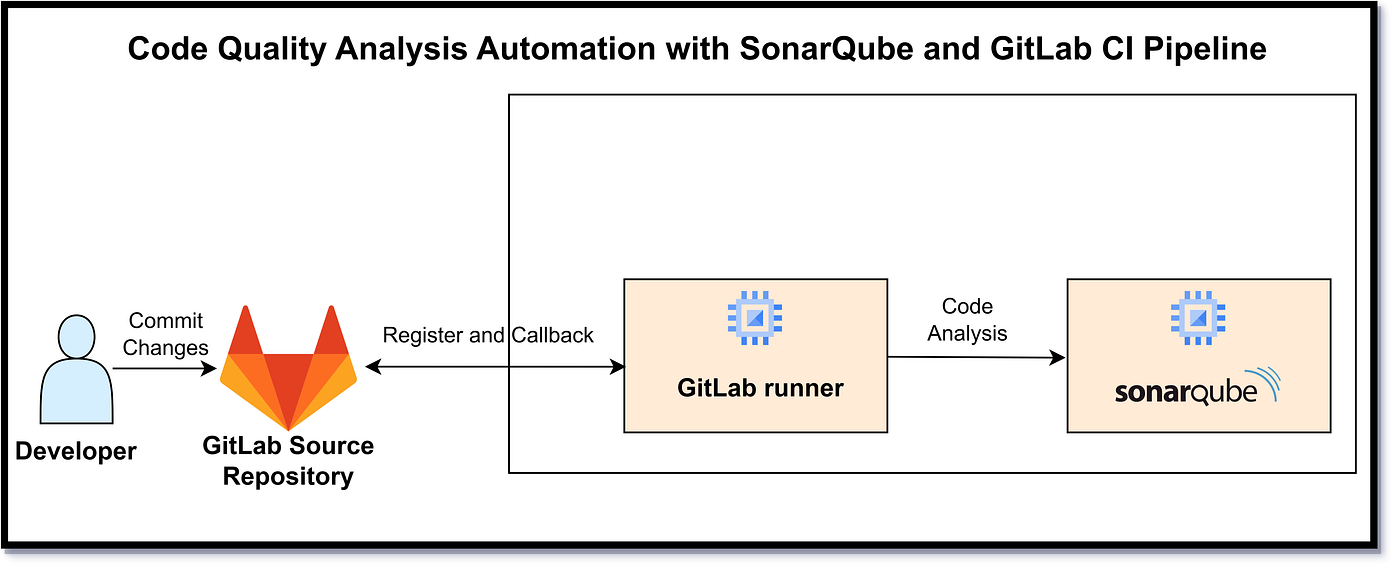
|  |  |
| --- | --- |
| **Participant** | **Role** |
| Manager | Assigns technician to service |
| API Gateway | JWT validation + routing |
| Service Request | Assignment logic |
| Auth Service | Manages technician data & workload |
| RabbitMQ | Async event messaging |
| Notification | Notifies customer & technician |

**Key Design Points**

* Technicians filtered by specialization matching service category
* Only on-duty technicians with available capacity shown
* Workload management prevents overloading technicians
* Service bay allocation tracked
* Both customer and technician notified asynchronously

**4. CI/CD Pipeline — Sequence Diagram**





**Scenario**

Developer pushes code to Git repository.

**Flow**

1. Developer pushes code
2. Jenkins pipeline triggered
3. Jenkins runs unit tests
4. Jenkins runs SonarQube scan
5. Quality gate checked
6. Docker images built
7. Docker Compose deploys services

**Participants**

* Developer
* Git
* Jenkins
* SonarQube
* Docker

**5. Error Handling — Sequence Diagram**

**Scenario**

Invalid request sent to backend.

**Flow**

1. Angular sends invalid request
2. Controller validation fails
3. Global exception handler triggered
4. Standard error response returned

**Key Point**

* Consistent error structure across services