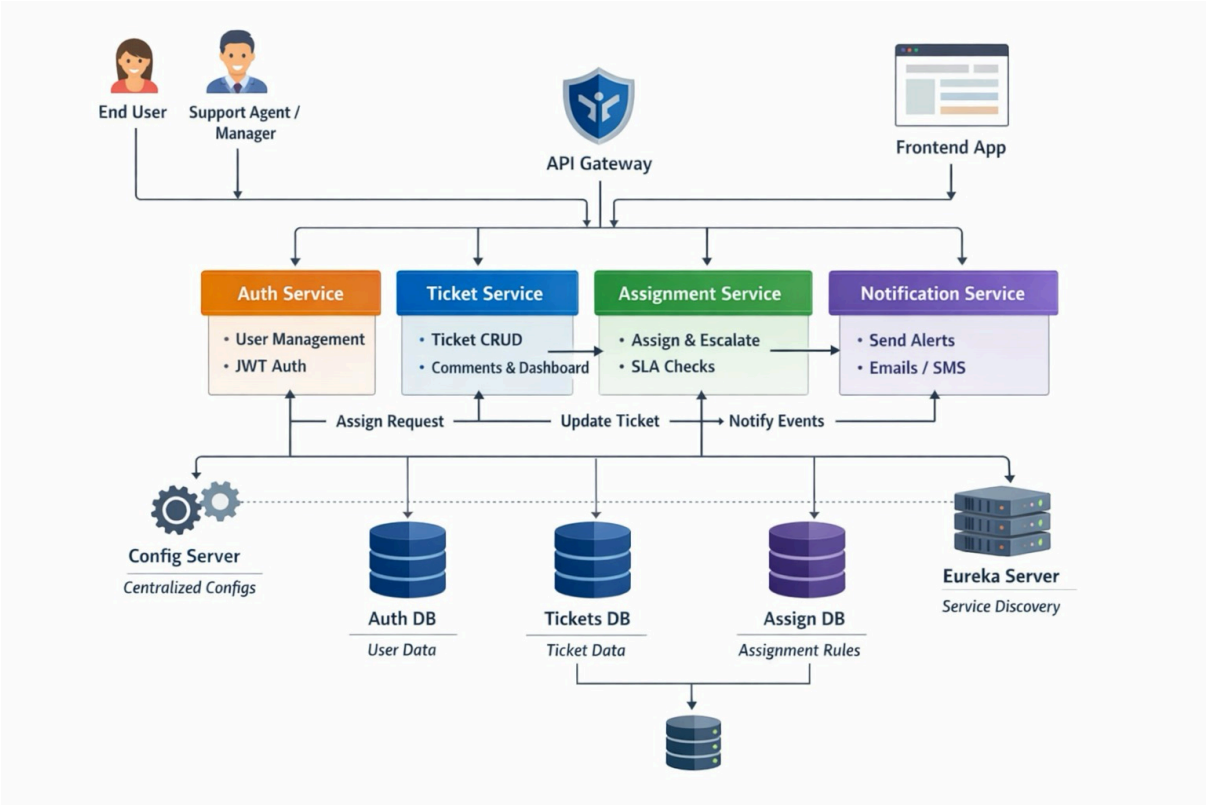
SOFTWARE DESIGN DOCUMENT(SDD)

Smart Ticket & Issue Management System



---

1. DOCUMENT CONTROL

Item	Details
Project Name	Smart Ticket & Issue Management System
Version	1.0
Author	Sujay Nimmagadda
Date	29/12/2025
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 Smart Ticket & Issue Management System.

It is intended for:

- Developers
- Reviewers
- Interview discussions
- Maintenance & enhancement planning

---

3. SYSTEM OVERVIEW

Business Objective

Provide a scalable, secure, and maintainable system to:

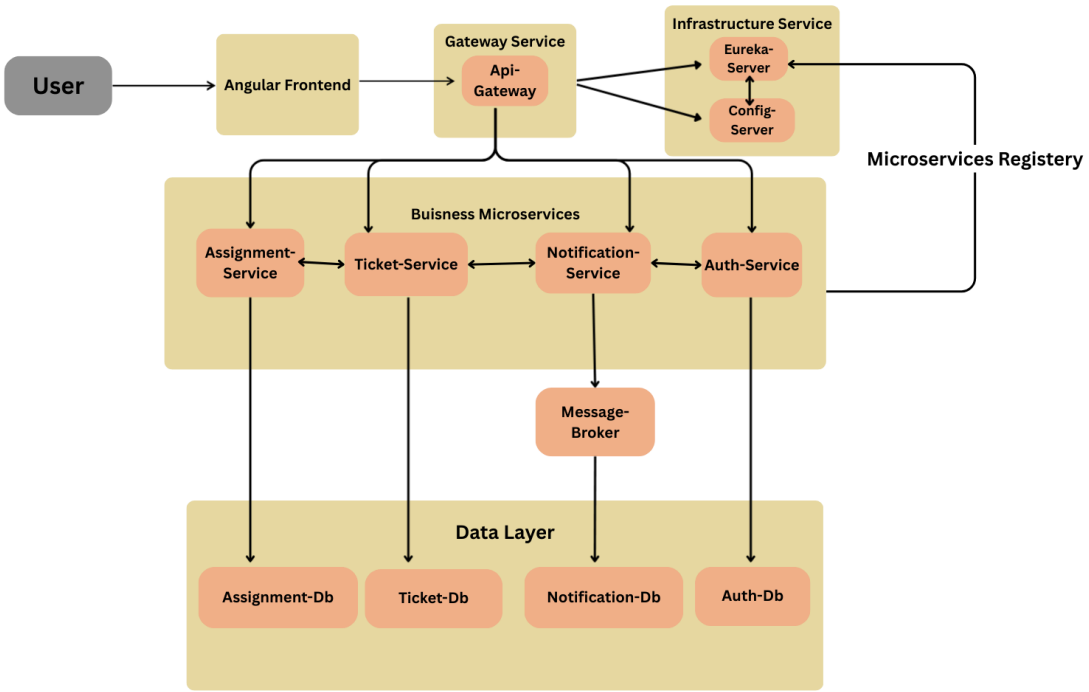
- Manage Users & Identities
- Handle Ticket Lifecycle
- Ticket Allocation
- Support growth through microservices

High-Level Features

- User Registration & Secure Login
- Real-time Ticket Tracking
- Role-based access (USER / ADMIN / Manager / Agent)

---

4. ARCHITECTURE OVERVIEW (HLD)



---

5. TECHNOLOGY STACK

Backend

- Java 17
- Spring Boot
- Spring Web
- Spring Data MongoDB
- Spring Validation

Frontend

- Angular
- Angular Material
- RxJS

Database

- MongoDB & Postgres

DevOps

- Docker
- Docker Compose
- Jenkins
- SonarQube

Testing

- JUnit 5
- Mockito
- Spring Boot Test

---

6. MICROSERVICES DESIGN

6.1 Auth Service

Responsibilities

- User registration
- Authentication
- Role management

APIs

```
POST /auth/register
POST /auth/login
GET  /auth/health
```

Database

- users collection (postgres)

6.2 Ticket Service

Responsibilities

- Ticket CRUD
- Lifecycle Management
- Reopen/Cancel
- History/Activity Logs
- Comments
- Dashboard

APIs

```
POST /tickets
GET  /tickets
GET  /tickets/{id}
PUT  /tickets/{id} (for updating status)
DELETE /tickets/{id}
GET /tickets/user/{userId}
POST /tickets/{id}/reopen (Reopen resolved/closed ticket)
GET /tickets/dashboard/summary
GET /tickets/dashboard/agent
```

Database

- Ticket collection (Mongodb)

6.3 Assignment Service

Responsibilities

- Assignment of Tickets
- Priority Decision
- SLA Rules

APIs

```
POST /api/assign/{ticketId}
GET  /api/escalations/manager/{mgrId}
GET  /api/agents/workload
GET  /api/escalations/logs
GET  /api/agent/{agentId}
```

Database

- assignment collection (Postgres)

6.4 Notification Service

Responsibilities

- Sends notifications

APIs

```
POST /notify/ticket-created
POST /notify/ticket-assigned
POST /notify/sla-breach
```

Database

- notification collection (Mongodb)

6.5 Api Gateway

Responsibilities

- forwarding api points

7. DATA DESIGN (LLD)

User Document

```
{
  "id": "u101",
  "email": "user@test.com",
  "password": "encrypted",
  "role": "USER",
  "active": true
}
```

Ticket Document

```
{
  "id": "p201",
  "title": "Phone",
  "description":"",
  "category":"",
  "priority":"",
  "status": "",
  "created_at": "",
  "updated_at":"",
  "due_date":"",
```

```
  "user_id":"",
  "assigned_agent_id"
}
```

Assignment Document

```
{
  "assignmentId": "assgn_8821",
  "ticketId": "T-1000",
  "agentId":"agent_1",
  "assignedBy":"manager_1",
  "assignedAt":"Time Stamp",
  "priority":"HIGH",
  "sla":{
    "responseTimeLimit":"Time Stamp",
    "resolutionTimeLimit":"TIme STamp",
    "isEscalated":boolean
  },
  "status":"ACTIVE/INACTIVE"
}
```

Notification Document

```
{
  "id": "N1001",
  "emailId": "abc@gmail.com",
  "description":"",
  "created_at": "Time Stamp",
}
```

---

8. API DESIGN & VALIDATION

- RESTful principles
- JSON request/response
- Bean Validation at DTO layer
- Service-level business rule enforcement
- Standard HTTP status codes

---

9. ERROR HANDLING STRATEGY

Global Exception Handling

- Centralized using `@ControllerAdvice`
- Standard error response format

```
{
  "timestamp": "TimeStamp",
  "status": 400,
  "error": "Validation Error",
  "message": "Invalid Ticket Data"
}
```

---

10. SECURITY DESIGN

- Password encryption (BCrypt)
- Role-based access control
- JWT authentication
- Secure API access

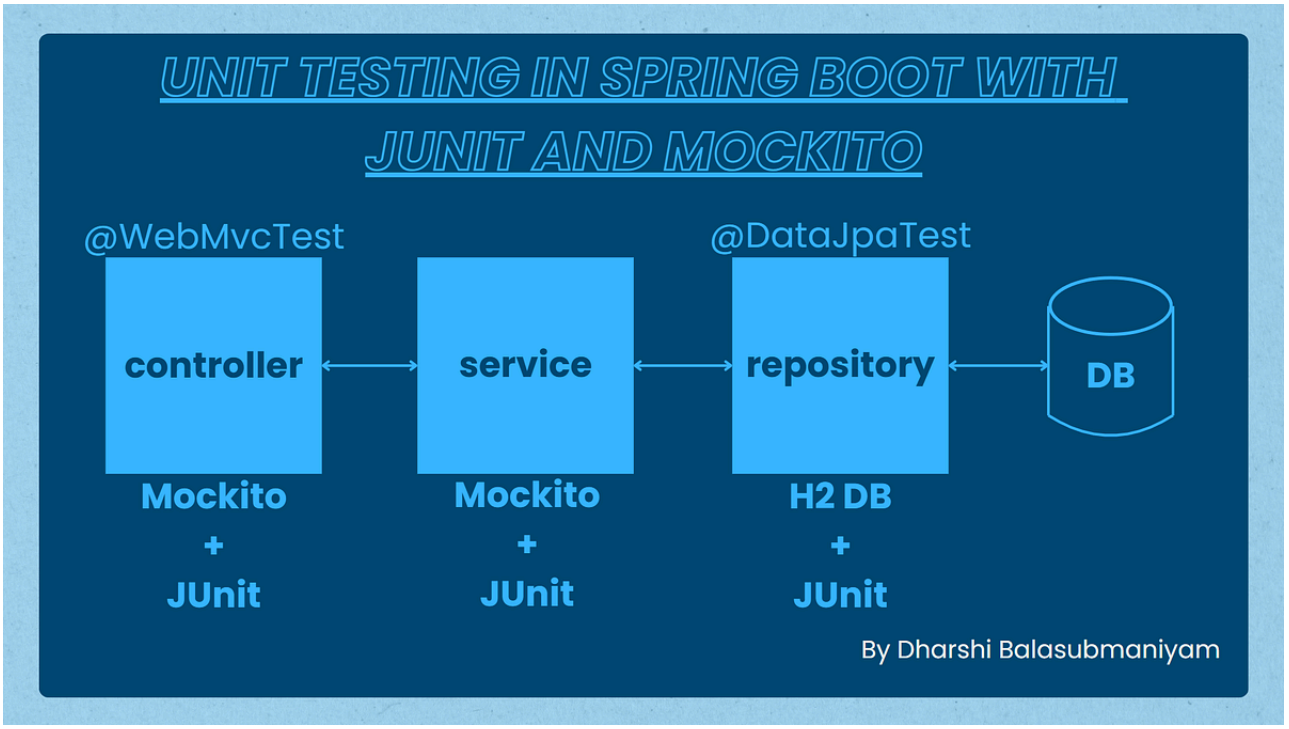
---

11. NON-FUNCTIONAL REQUIREMENTS

Area	Design Decision
Scalability	Stateless services, Docker
Performance	Pagination, async calls
Availability	Independent services
Maintainability	POM-like layered backend
Security	Validation, encryption
Observability	Logging & monitoring

---

12. TESTING STRATEGY



Backend

- Unit tests (Service layer)
- Controller tests (MockMvc)

- Minimum 90% coverage

Frontend

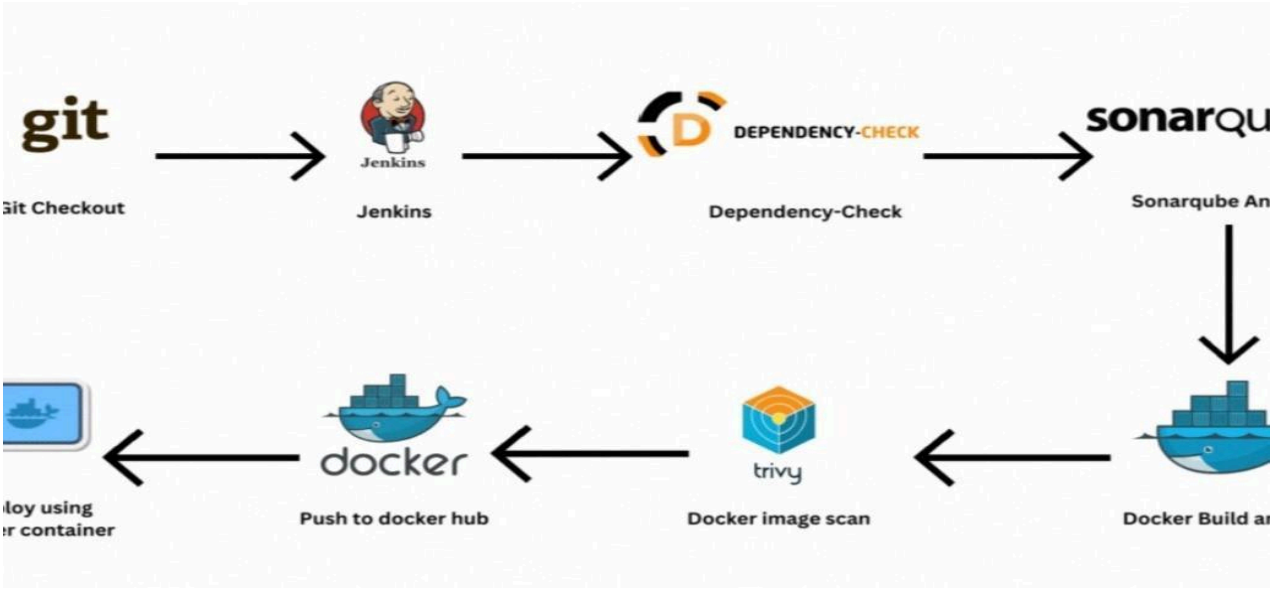
- Component tests
- Service tests

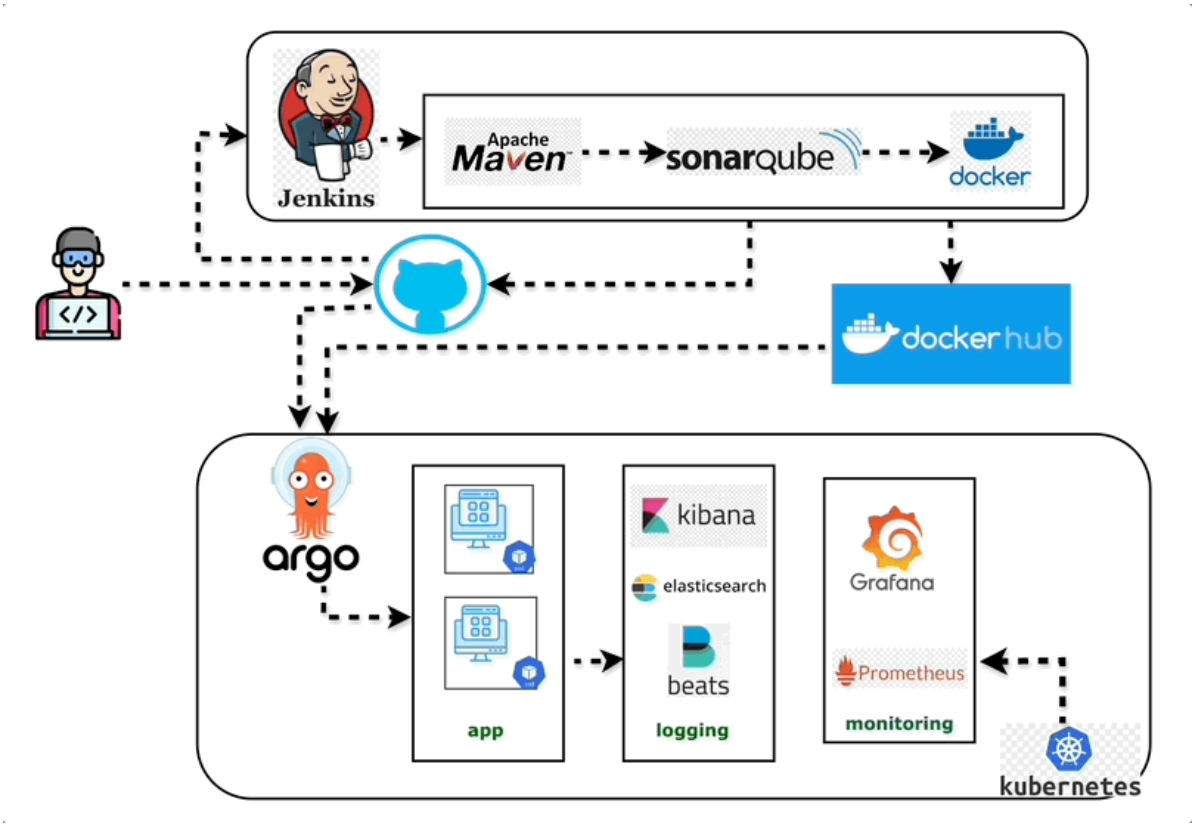
Quality Gates

- 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

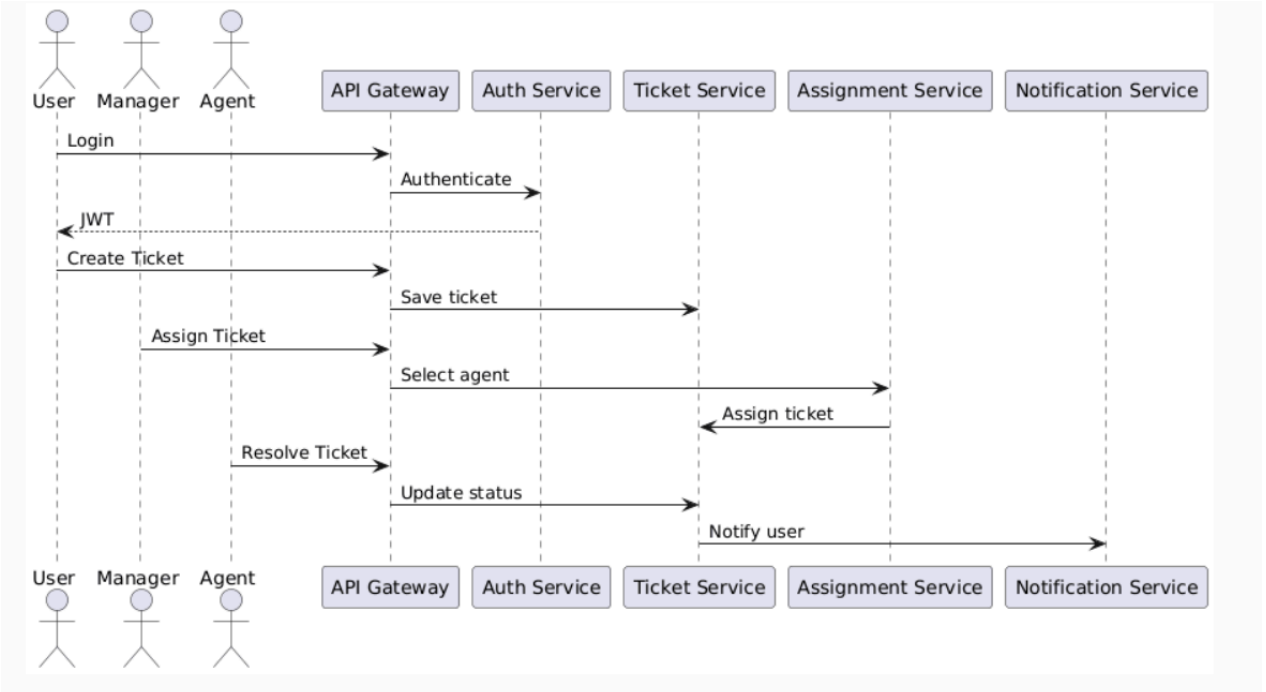
- Spring Cloud Gateway
- Kafka-based async communication
- Kubernetes deployment
- Centralized logging (ELK)

---

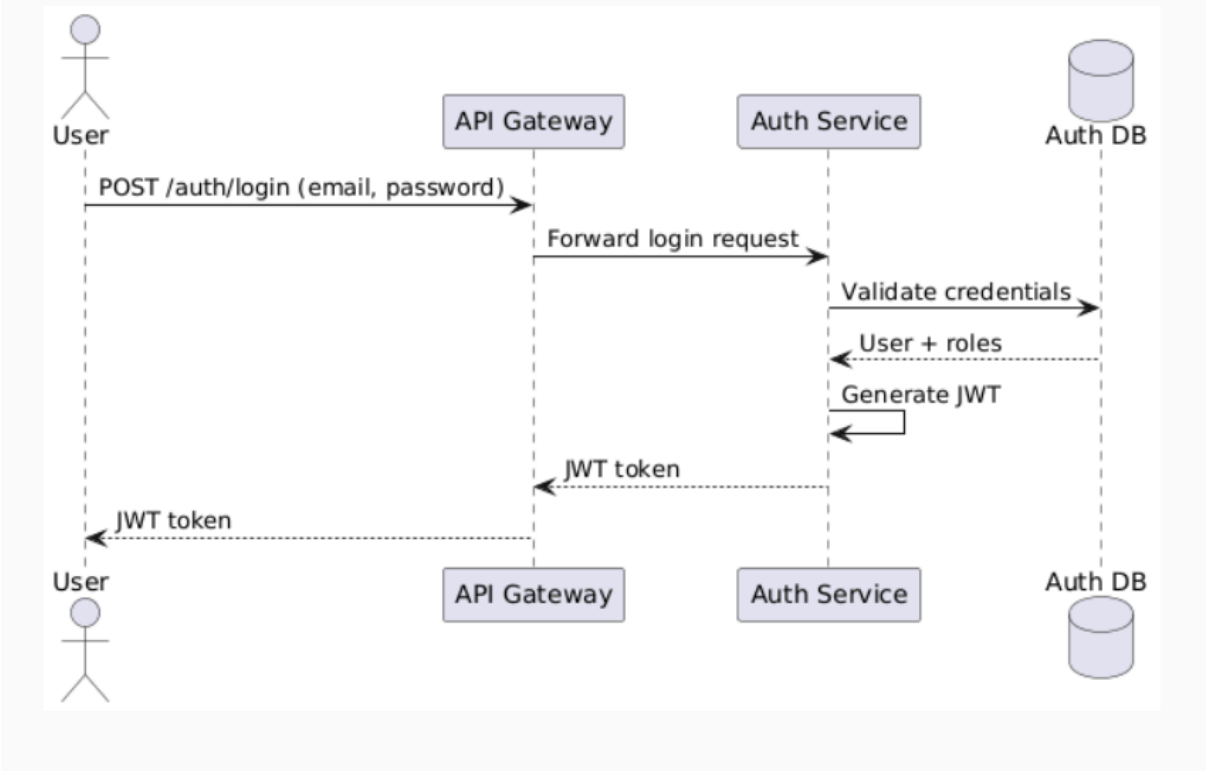
17. CONCLUSION

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

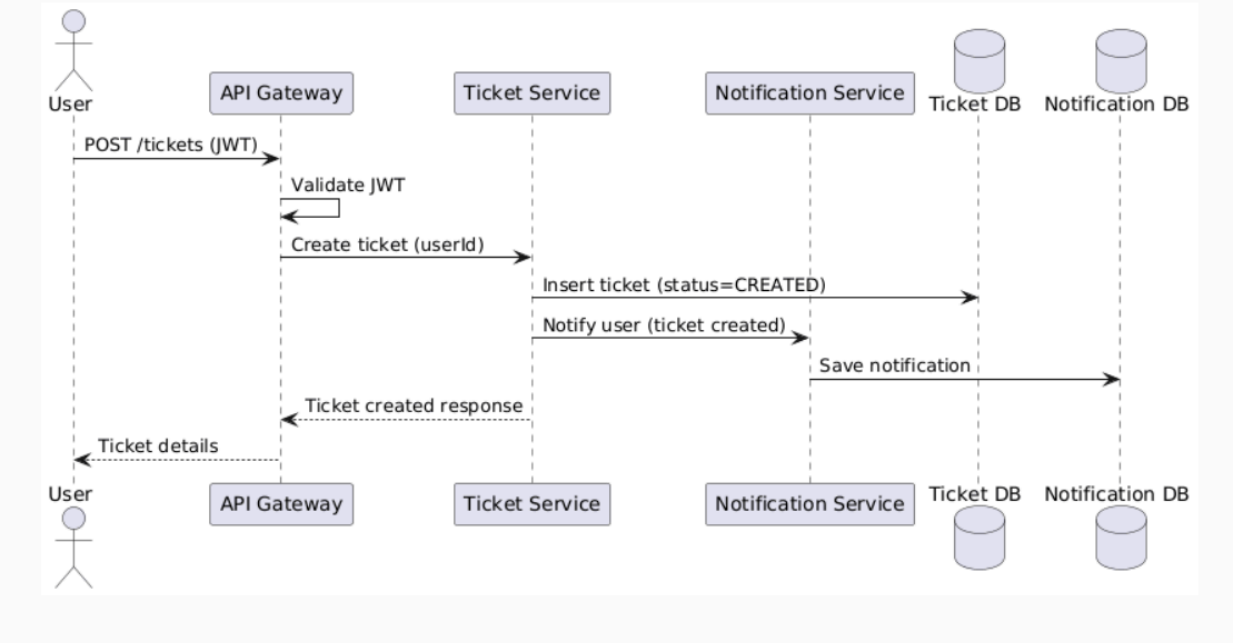
SEQUENCE DIAGRAMS



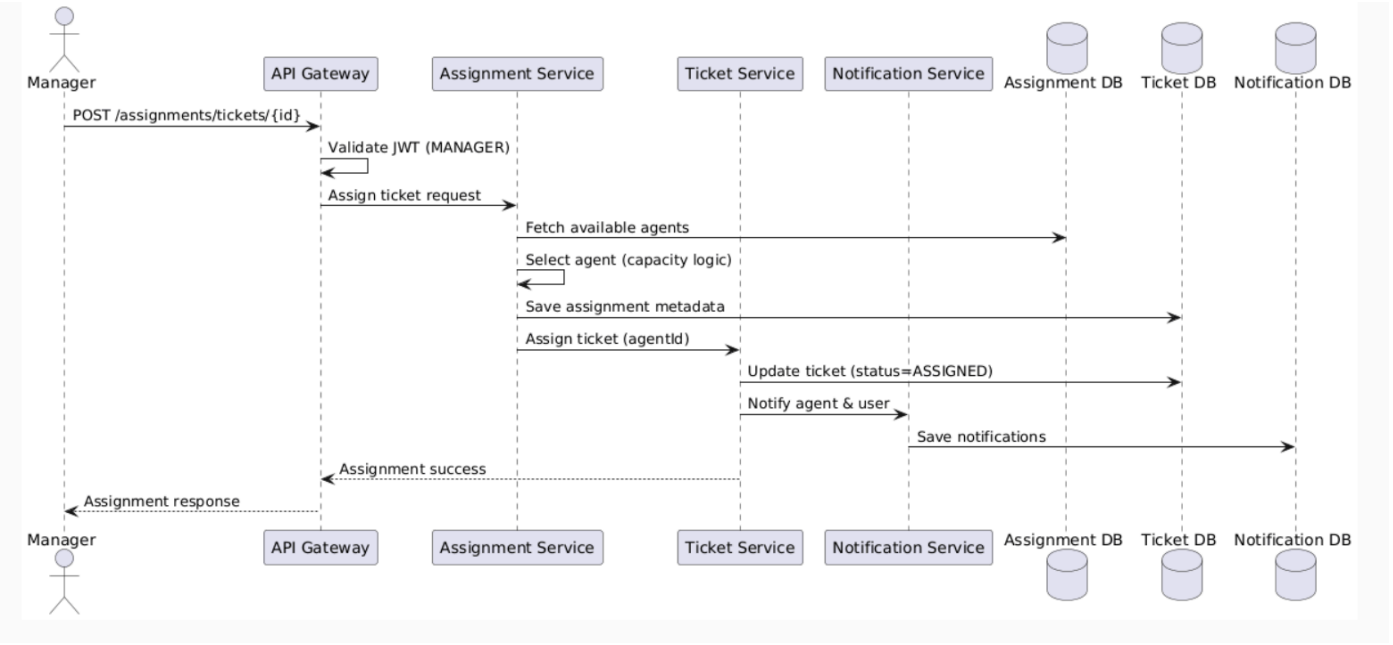
User Login Flow -



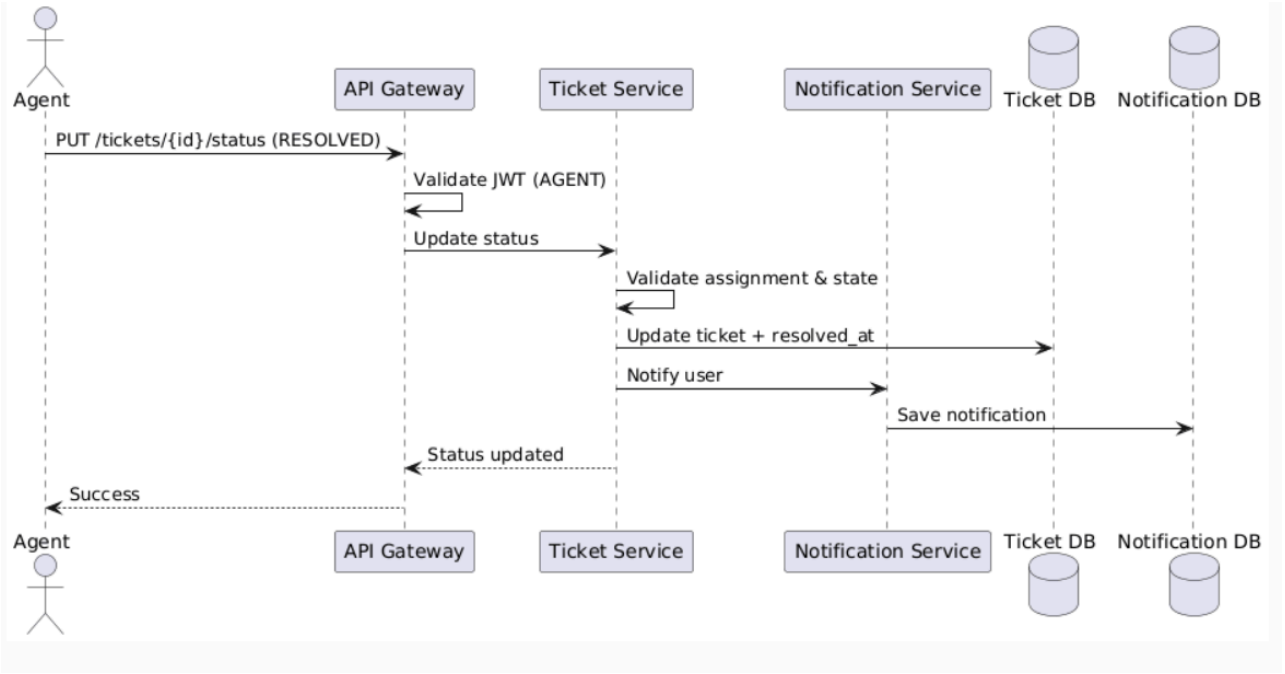
Ticket Creation -



Manager Assigns Ticket to Agent -



Agent Resolves Ticket -



1. User Registration — 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

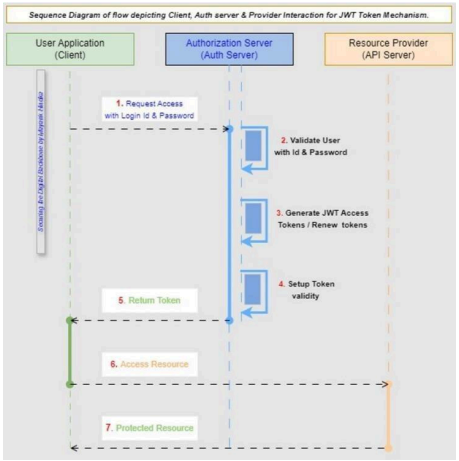
- User
- Angular UI
- User Service
- MongoDB

Key Design Points

- Validation at API boundary
- Encryption at service layer
- No direct DB access from UI

---

2. User Login — Sequence Diagram



Scenario

Registered user logs into the system.

Flow

1. User submits login form
2. Angular calls `POST /users/login`
3. User Service fetches user from MongoDB
4. Password is verified
5. (Optional) JWT token generated
6. Response sent to Angular
7. Angular stores token and navigates user

Participants

- User
- Angular UI
- User Service
- MongoDB

Failure Cases

- Invalid credentials → 401
- Inactive user → 403

---

3. Ticket Listing — Sequence Diagram

Scenario

User views available products.

Flow

1. Angular loads product page
2. `GET /products` sent to Product Service
3. Product Service queries MongoDB
4. Product list returned to Angular

Participants

- Angular UI
- Product Service
- MongoDB

---

4. Order Placement — Sequence Diagram (MOST IMPORTANT)

Scenario

User places an order for one or more products.

Flow

1. User clicks **Place Order** in Angular
2. Angular sends `POST /orders` to Order Service
3. Order Service validates request
4. Order Service calls Product Service to check stock
5. Product Service validates availability
6. Order Service creates order
7. Product Service reduces inventory
8. Order saved in MongoDB
9. Success response returned to Angular

Participants

- User
- Angular UI
- Order Service
- Product Service
- MongoDB

Key Design Decisions

- Inventory validation before order creation
- Atomic stock update logic
- Failure stops order creation

---

5. Order Status Update (Admin) — Sequence Diagram

Scenario

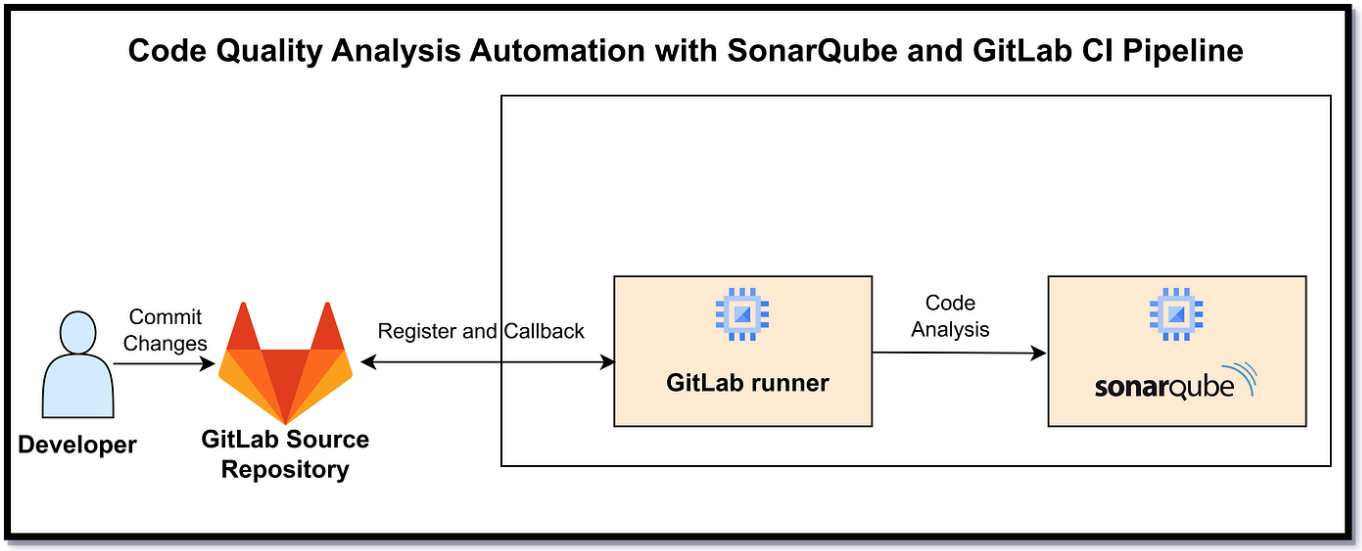
Admin updates order status.

Flow

1. Admin sends update request from Angular
2. `PUT /orders/{id}/status` sent to Order Service
3. Role validation (ADMIN only)
4. Order status transition validated
5. Order updated in MongoDB
6. Updated status returned

Valid Transitions

- CREATED → PLACED
- PLACED → COMPLETED
- PLACED → CANCELLED



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

---

7. 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