System Design Document (Template) — COMP 301: Software Architectures and Tools

> **Purpose:** This template guides you to produce a compact, clear, and professional System Design Document (SDD) for your microservices project (e.g., the EventPlanner). Fill each section with project-specific information. Use diagrams where indicated.

---

1. Executive Summary

Provide a 2–4 paragraph overview of the system, its purpose, high-level architecture, main functional areas, and primary stakeholders.

- **Project name:** EventPlanner
- **Authors / Team:** Team members' names and roles
- **Target users / stakeholders:** e.g., event organizers, attendees, admins
- **Scope:** Short description of what this document covers and any known limitations

Guidance: Keep it concise. This section should give a reader (e.g., instructor) enough context to understand the rest of the document quickly.

---

2. Goals & Non-Functional Requirements (NFRs)

List measurable goals and NFRs such as:

- **Availability:** e.g., 99.5% uptime
- **Scalability:** e.g., must handle 1,000 concurrent users
- **Performance:** e.g., average API response < 300 ms
- **Security:** e.g., OAuth2/JWT for user authentication, TLS in transit
- **Maintainability:** e.g., modular services, documented APIs
- **Data Retention & Compliance:** e.g., retain logs for 90 days

Guidance: Make NFRs specific and testable when possible.

---

3. Scope and Requirements

3.1 Functional Requirements

Use numbered items and map them to microservices where appropriate.

1. Users can create an account (User Service).
2. Organizers can create events (Event Catalog Service).
3. Users can book tickets (Booking Service).
4. Payments are processed and recorded (Payment Service).
5. Admins can view system metrics (Monitoring Service).

3.2 Constraints & Assumptions

- Must be implemented with Spring Boot.
- Each microservice should have its own Git repository (or mono-repo with clear module boundaries).
- Deploy to Render as independent services.
- Assume student teams have Docker installed for local testing.

---

4. High-Level Architecture

Example Component Diagram (Mermaid)

```
graph TD
    A[User Interface] -->|REST| B[API Gateway]
    B --> C[User Service]
    B --> D[Event Catalog Service]
    B --> E[Booking Service]
    E --> F[Payment Service]
    D --> G[(MongoDB)]
    C --> H[(PostgreSQL)]
    E --> I[(Redis Cache)]
```

Guidance: Replace placeholders with your own architecture. Show key components, databases, and communication paths.

---

5. Component Design (Per Service)

For each microservice, create a subsection with the following structure. Duplicate the subsection per service.

5.x Service Name (e.g., EventCatalogService)

- **Purpose:** Short sentence
- **Responsibilities:** Bullet list
- **Exposed APIs:** Brief table (see API spec section)
- **Data storage:** DB type, key tables/collections, retention
- **Dependencies:** Other services, external APIs, message brokers
- **Scaling strategy:** Horizontal, vertical, stateful/stateless notes
- **Configuration:** Env vars, config server keys

Guidance: Keep each service focused; one service → one bounded context.

---

6. Example Sequence Diagram — Ticket Booking Flow

```
sequenceDiagram
    participant U as User
    participant G as API Gateway
    participant B as Booking Service
    participant E as Event Service
    participant P as Payment Service

    U->>G: POST /bookTicket
    G->>B: Forward booking request
    B->>E: Check event availability
    E-->>B: Availability confirmed
    B->>P: Process payment
    P-->>B: Payment success
    B-->>U: Booking confirmation response
```

Guidance: Use sequence diagrams to illustrate inter-service communication for core use cases.

---

7. Data Model

Example ER Diagram (Mermaid)

```
erDiagram
    USER ||--o{ BOOKING : makes
    EVENT ||--o{ BOOKING : includes
    USER {
        UUID id
        string name
        string email
        string passwordHash
    }
    EVENT {
        UUID id
        string title
        datetime date
        string location
        int capacity
    }
    BOOKING {
        UUID id
        UUID userId
        UUID eventId
        datetime bookingDate
        boolean paymentStatus
    }
```

Guidance: Include ERD or schema diagram for each microservice. Clearly indicate relationships and key constraints.

---

8. Security & Operational Concerns

- **Auth:** JWT tokens issued by User Service

- **Transport:** HTTPS enforced
- **Secrets:** Managed through Render Environment Variables
- **Monitoring:** Use Render metrics dashboard
- **Logging:** Application logs collected via `stdout`

---

9. CI/CD Workflow

Example GitHub Actions outline:

```yaml
name: CI Pipeline
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: '23'
      - name: Build with Maven
        run: mvn clean package -DskipTests
      - name: Run Tests
        run: mvn test
```

Guidance: Extend with Docker build and Render deploy steps if applicable.

---

10. Quick Checklist (for Students — Submit with SDD)

- [ ] Executive Summary included
- [ ] Functional & Non-functional requirements listed
- [ ] Architecture and diagrams attached
- [ ] Component descriptions for each microservice
- [ ] API specs for core endpoints
- [ ] Data model (ERD or collections) included
- [ ] Security and CI/CD considerations documented
- [ ] Test plan and sample results included
- [ ] Render service URL documented in README

---

11. Grading Rubric (10 Points Total)

Criterion	Description	Points
1. Clarity & Organization	Document is structured, readable, and follows template	**1**
2. Completeness	All major sections are filled with relevant content	**1**

Criterion	Description	Points
3. Architecture Design	Logical, modular design; diagrams well presented	**2**
4. Component Descriptions	Each microservice is clearly defined and justified	**1**
5. API Documentation	Clear endpoint specifications and data flow	**1**
6. Data Model Design	Well-structured ERD and relationships	**1**
7. Security & CI/CD Coverage	Key security and deployment details present	**1**
8. Testing Strategy	Realistic and traceable testing approach	**1**
9. Professional Presentation	Neat formatting, diagrams, and references	**1**

Total: 10 points

> 💡 *Instructor Tip:* Minor deductions (0.25–0.5 pts) can be applied for missing diagrams, unclear writing, or poor alignment between architecture and requirements.

Prepared for: COMP 301 — Fall 2025
Template version: 2025-10-25