NAMIBIA UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS

DEPARTMENT OF SOFTWARE ENGINEERING

Assignment 2 — Smart Public Transport Ticketing System

Course: Distributed Systems and Applications (DSA612S)
Assessment: Second Assignment — Group Project
Released: 19 September 2025
Deadline: 05 October 2025, 23:59
Total Marks: 100

Overview & Learning Objectives

This assignment asks you to design and implement a distributed **smart public-transport ticketing system (buses & trains).** The goal is to practise real-world distributed systems skills: microservices architecture, event-driven communication using Kafka, persistent storage (MongoDB or SQL), containerisation, and orchestration (Docker Compose or Kubernetes).

By the end of the assignment students should be able to:

- Design and implement microservices with clear boundaries and APIs.
- Apply event-driven design using Kafka topics and producers/consumers.
- Model and persist data in a database and reason about consistency trade-offs.
- Containerise services and orchestrate them for multi-service deployment.
- Demonstrate testing, monitoring, and fault-tolerance strategies for distributed systems.

Problem Description

Public transport is the backbone of many cities, but traditional ticketing systems still rely heavily on **paper tickets** or **standalone machines**. These systems create multiple problems: passengers often struggle to find convenient ways to purchase or top-up tickets, delays occur when validators fail, and administrators have limited visibility into ticket usage patterns.

The Windhoek City Council has therefore requested a **modern distributed ticketing platform** to replace its outdated system for **buses and trains**. The system must support **different user roles** (passengers, transport administrators, and validators on vehicles) and provide a seamless experience across devices.

From the **passenger's perspective**, the system should enable:

- Easy account creation and secure login.
- Browsing of available routes, trips, and schedules.
- Purchase of different ticket types (single-ride, multiple rides, or longer-term passes).
- Convenient validation of tickets on boarding, ensuring only valid tickets are used.
- Notifications about trip disruptions, delays, or cancelled routes.

From the **administrator's perspective**, the system should allow:

- Creation and management of routes and trips.
- Monitoring of ticket sales and passenger traffic.
- Publishing of service disruptions or schedule changes in real time.
- Generating reports on usage patterns to plan resources more effectively.

From the **system perspective**, the solution must be **scalable and fault-tolerant**. Since many passengers may purchase or validate tickets simultaneously (especially during peak hours), the system must handle **high concurrency**. Communication between services should be **event-driven** using Kafka to ensure that ticket requests, payment confirmations, and schedule updates are processed reliably and asynchronously.

All important data — such as users, trips, tickets, and payments — must be stored in a persistent data store (MongoDB or SQL). Each component should be deployed as a

microservice, running inside Docker containers, and orchestrated with Docker Compose or Kubernetes to simulate a production-like environment.

Required Services

1. **Passenger Service:** register/login, manage accounts, view tickets.
2. **Transport Service:** create and manage routes/trips, publish schedule updates.
3. **Ticketing Service:** handle ticket requests, lifecycle (CREATED → PAID → VALIDATED → EXPIRED).
4. **Payment Service:** simulate payments, confirm transactions via Kafka events.
5. **Notification Service:** send updates when trips change or tickets are validated.
6. **Admin Service** — manage routes, trips, ticket sales reports, and publish service disruptions or schedule changes.

Key Technologies

- **Ballerina** for all service implementations.
- **Kafka** topics (e.g., ticket.requests, payments.processed, schedule.updates).
- **MongoDB or SQL** for persistence.
- **Docker** for containerisation and **Docker Compose/Kubernetes** for orchestration.

Evaluation Criteria

- Kafka setup & topic management – 15%
- Database setup & schema design – 10%
- Microservices implementation in Ballerina – 50%
- Docker configuration & orchestration – 20%
- Documentation & presentation – 5%

Total = 100 marks

Creativity & Extensions (Bonus)

Students can shine by adding:

- Seat reservations with concurrency handling.
- Real-time dashboard for trips/tickets.
- Kubernetes deployment with autoscaling.
- Monitoring & metrics (Prometheus/Grafana).

Submission Instructions

- This assignment is to be completed by groups of 5-7 students each.
- For each group, a repository should be created on GitHub or GitLab. The repository should have all group members set up as contributors.
- All assignments must be **uploaded** to a GitHub or GitLab repository. Students who haven't pushed any code to the repository will not be given the opportunity to **present and defend the assignment.** More particularly, if a student's username does not appear in the commit log of the group repository, that student will be assumed not to have contributed to the project and thus be awarded the mark 0.
- The assignment will be group work, but individual marks will be allocated based on each student's contribution to the assignment.
- Marks for the assignment will only be allocated to students who have presented the assignment.
- It's the responsibility of all group members to make sure that they are available for the assignment presentation**. An assignment cannot be presented more than once.**
- The submission deadline date is **Sunday, 05 October 2025, at 23h59.** Please note that commits after that deadline will not be accepted. Therefore, a submission will be assessed based on the clone of the repository at the deadline.
- Any group that fails to submit on time will be awarded a mark of 0. Late submissions are not to be entertained.
- There should be no assumption about the execution environment of your code. It could be

run using a specific framework or on the command line.

- In the case of plagiarism (groups copying from each other or submissions copied from the Internet), all submissions involved will be awarded the mark 0, and each student will receive a warning.
- 100% AI-generated codes will be awarded a zero. AI tools should be used as a guide only.

End of Assignment