Railway Imperium

The Premise

You are to program a game that runs on a server and is played by addressing its API. The API calls can be made manually, using a frontend or mobile application. In the different courses participating in this project, you will develop models, the API(s), the backend as well as multiple frontends and mobile applications using various technologies.

The game

You are to implement a game in a fictional world where the economy is dependent on railways. The player takes on the role of a railway mogul, developing his company by building railway networks between cities and rural businesses. Passengers and mail can travel these networks and industries in cities can get the resources they need and sell their products to faraway cities. Investments need to pay off by making money of the trains. The main gameplay mechanics are:

Resources

Towns, industries, and rural businesses produce goods, passengers, and mail. All of them grow naturally over time as their needs are met. If their needs aren't met, they shrink again, producing less but also consuming less.

Railways

Railways are built between railway stations. A railway can carry an arbitrary number of trains and we assume they are multi-track, that is, multiple trains can travel the network bi-directionally without accidents happening. Railways have a cost based on their distance. Railways can only be built between stations that have a free platform.

Stations

Stations need to be built in either a town or rural business. Only one station can be built per town or business. They are used as the destination or origin of train routes.

Towns

Towns have levels that are dependent on their population. An increase in level increases the needs a town has. The more populous a town, the more its level increases, the more needs need to be fulfilled to increase the population. Towns can have one industry per two levels of population, starting at level 2, up to a maximum of three industries.

Each town produces passengers and mail depending on their level of population. Both of these needs to be transported to their destinations by train.

Industry

Industry is built in a town with enough population. An industry produces a specific good from raw materials typically produces by rural businesses. The goods can be consumed locally by the town or can be transported by trains to different towns. Industries have a certain warehouse capacity and cannot produce once their warehouse is at capacity.

Cost and Income

Every asset, be it train, station, railway or industry, costs money to operate. These costs are dependent on the level, length, or distance travelled of the respective asset. Trains generate income based on their haul as well as the distance travelled. Industries generate income based on the number of goods produced.

The project

For this project, you need to implement the game to run on a server. The data is stored persistently in a database. The game can be played by addressing an API. A frontend allows to play the game directly in the browser and using a mobile phone using a native app.

You need to realise the following for the courses listed below:

- API specification and documentation [**FUWEB**]
- API implementation [**FUWEB**, **JAVA3**]
- Database documentation and implementation [**BASDO3**]
- Frontend implementation for browser [**FUWEB**]
- Frontend implementation for PC [**JAVA3**]
- Frontend implementation for native mobile app [**JAVA3**, **MOBAP**]

API specification and documentation

You are to create this API using the OAS in version 3.*.*. Share a link to your public specification when you post in Moodle. Your documentation must also be publicly available for ease of access. The documentation is graded in **FUWEB**.

API implementation

The API is implemented in multiple courses using multiple technologies. Your APIs of FUWEB and JAVA3 should be interchangeable with your frontends without need to adapt anything.

FUWEB

The API implementation in your **FUWEB** course will be done in Node.js and in PHP. Both instances need to run separately and cannot be interdependent. They need to, however, address the same database as information must be kept persistent. The API requires authentication using a **JWT**. This requires that the API, in addition to the endpoints implementing the game, also exposes endpoints to **register** and **login**.

JAVA3

The API implementation in your **JAVA3** course will be done using JSP/JSF and/or Spring Boot. You need to use the same database as in your FUWEB implementation. The API requires authentication using a **JWT**. This requires that the API, in addition to the endpoints implementing the game, also exposes endpoints to **register** and **login**.

BASDO3 – Conceptual and Physical Data Model, Secure Database Interactions, and Documentation

Objective: In this part of the project, you will focus on designing the database structure and ensuring secure interactions with the database using stored procedures. Your primary goal is to create a robust and secure database component for the multiplayer game.

Part 1: Data Modelling

1. **Conceptual Data Model (CDM):** Create a conceptual data model that represents the high-level entities, relationships, and attributes required for the game.

2. **Physical Data Model:** Refine the conceptual model into a detailed physical data model using Workbench. Define all the attributes for each entity, their data types, constraints (such as unique or required fields), and relationships between tables.
3. **Validation and Feedback:** Submit your conceptual and physical data models for validation and feedback by the instructor. Make necessary revisions based on the feedback to ensure data integrity and efficiency.

Part 2: Secure Database Interactions

1. **Database Interactions via Stored Procedures:** Implement a strategy where the frontend application interacts with the database exclusively through API calls that, in turn, utilize stored procedures. The stored procedures should handle Create, Read, Update, and Delete (CRUD) operations in a secure and controlled manner.
2. **Data Exchange Format:** The frontend should send data to the API in JSON format. Stored procedures should expect JSON objects as input parameters and return standardized JSON response objects. Ensure data validation and error handling within the stored procedures.
3. **Security Best Practices:** Implement security best practices such as JSON schema validation, input data validation, parameterized queries to prevent SQL injection, and appropriate access controls.
4. **Transaction Management:** Where applicable, implement database transactions to ensure data consistency and integrity during multiple database operations.
5. **Error Management:** Develop robust error-handling mechanisms within the stored procedures to provide informative and secure error responses in case of issues.
6. **Audit Trails:** Implement a mechanism to log and track actions within the database.

Part 3: Documentation and Testing

1. **Documentation:** Create comprehensive documentation that includes:
    - Conceptual and physical data model diagrams (ERDs).
    - Description of stored procedures, including their purpose and parameters.
    - Database schema and schema creation script.
    - Backup and recovery plan/procedure.

- Security measures implemented (e.g., JSON schema validation, access controls).
- Audit trail design and its significance.

2. **Test Scripts:** Develop test scripts to validate the functionality and security of the stored procedures. These scripts should cover various scenarios, including valid and invalid inputs, error conditions, and transactions.
3. **Screencast Proof:** Record a screencast demonstrating the execution of your test scripts to validate the correctness and security of your stored procedures.

Deliverables:

1. Project documentation with the above components.
2. Test scripts for testing stored procedures, covering various scenarios.
3. A screencast demonstrating the execution of your test scripts to validate the correctness and security of your stored procedures.

Frontend implementation for browser

The implementation of the frontend for use in a browser needs to be done using established frameworks and cannot be done manually.

FUWEB

Identify a framework for the programmatic generation of an adequate frontend respecting responsive design. Your frontend must allow players to access all the game's features. You are free to choose the framework you think best but must validate your choice with your teacher.

Frontend implementation for PC

JAVA3

You need to implement a classical Java application that uses your API that you have implemented. Your frontend must allow players to access all the game's features.

Frontend implementation for native mobile app

JAVA3

Create an Android application using your API and giving the player all the possible interactions with your API. You are allowed to use Frameworks for the Android implementation, but you need to confirm them with your teacher first. The implementation can be done in Java or in Kotlin.

MOBAP

Create an iOS application using the developed API. The use of frameworks supporting the development of your iOS application is allowed. **Before using any framework**, please discuss your choice with your teacher to receive the **required** approval.

The following additional functionalities/features must be implemented:

- Your application must allow the user to select (at least) the following languages:
  - English
  - Any language of your choice (preferably French or German)
- The layout of your application must be responsive and user friendly independent of the device used.
- The connection with the developed API is performed by using a JWT token for user authentication.
- The storage of any relevant data must follow best practices.
  - What is supposed to be saved on the database?
  - What is supposed to be saved locally inside your application?

Deliverables:

- Mock-Ups explaining the base structure of your application and its behaviour.
- Documentation explaining the crucial parts of your application and design decisions you took.
- Video demonstration of your application.

Backend implementation details

World generation

The backend manages worlds. Each world consists of a 1000 by 1000px grid with a coordinate system. This system is used to place world assets. World assets are either towns or rural businesses. The world needs to be generated with 15 towns and 30 rural businesses. Each must be spaced such that no other asset can be within 50px in any direction. The world is generated once and then kept persistent. Once a world reaches 20 players, a new world is generated, and players directed to that world until the process repeats.

Towns and needs

Towns have a name that makes them easily identifiable. They keep track of their population which starts at 500 people. This is also the minimum possible. Towns grow if their needs are met. The following table summarizes town levels as well as their needs.

LEVEL	POPULATION	NEEDS	MAX INDUSTRY	MAIL GENERATED	PASSENGERS GENERATED
1	500-1k	1	0	5	10
2	1k-2k	2	1	10	20
3	2k-5k	3	1	50	40
4	5k-10k	4	2	200	100
5	10k-20k	5	3	250	150

The level of their need translates to the following table:

LEVEL	GOOD	CONSUMPTION/DAY	STOCKPILE
1	Grain	0,5	10
	Beverage	0,8	
	Wood	0,3	
	Meat	0,5	
2	Milk	0,3	20
	Bread	0,8	
	Fruit	0,5	

	Planks	0,4	
3	Leather	0,3	25
	Wool	0,3	
	Cheese	0,5	
4	Furniture	0,3	35
	Clothing	0,8	
	Metals	0,4	
5	Jewellery	0,3	50
	Tools	0,2	

In addition, all goods consumption goes up by 1 unit per day per level. For example, a town at level 3, having level 3 needs, consumes 2,5 units of meat per day initial 0,5 plus 2*1 for being level 3. This applies to all goods across all levels.

If more than 75% of a town's needs are met, the town grows. Per day the demand is met, the town grows 10%. For every day the demand is met by less than 50%, the town dwindles by 10%. Depending on the level of the town, they can stockpile goods. The stockpile is cumulative and not on a by good basis.

Rural Businesses

Rural businesses are what keeps the economy going. They produce goods that towns directly need or that are transformed by industry to make other goods. The following table shows you the types of businesses available. For this game, production always meets the demand. Rural businesses can be bought for 250k. They cannot be sold.

BUSINESS	PRODUCES
RANCH	Cattle, Milk
FIELD	Grain, Wheat
FARM	Leather, Wool
LUMBERYARD	Wood
PLANTATION	Fruit
MINE	Ore, Gems

Industry

Industry is built in towns if building slots are available (see town level and max industries). Industries have a cost to install (500k) as well as an upkeep cost (500/day). They also consume resources to produce goods. The following list includes all industries available to be built.

INDUSTRY	CONSUMES	PRODUCES
BREWERY	Wheat	Beverage
BUTCHER	Cattle	Meat
BAKERY	Grain	Bread
SAWMILL	Wood	Planks
CHEESEMAKER	Milk	Cheese
CARPENTER	Planks, Leather	Furniture
TAILOR	Wool, Leather	Clothing
SMELTER	Ore	Metals
SMITHY	Metals	Tools
JEWLER	Tools, Gems	Jewellery

Each industry consumes 2 units of goods per day from the town's stores and produces 2 units of goods except for industries consuming multiple different goods, these consume 1 unit of each good and still produce 2 units of goods. Each industry can stockpile 20 units of goods. If they aren't consumed or transported off, which lowers the stockpile, the industry will stop working until there is room for their output. However, they still cost upkeep per day.

Stations, Trains, and Tracks

Trains travel between stations. For each departing train, it knows its destination and the needs of the destination. It loads up to 10 wagons with goods to satisfy the needs of the destination town. Should not all wagons be loaded, the rest is filled with mail or passengers (random distribution per wagon). Trains can load wagons on the trips to and from the destination.

Stations are built in towns or rural businesses. The cost to create a station is a flat 100k with an operational cost of 1000 per week. Tracks are built between stations. Tracks have a distance which is calculated by using the position of the stations (and thus their cities or businesses). The travel distance is 50px per day.

Trains can be bought for 50k and always travel between two stations. Trains have an operational cost of 500 per day. The stations are fixed when the train is created. Trains can be deleted, refunding their cost. Stations cannot be deleted.

Funds and income

Each player has a company that starts with 500k funds. Each day they pay their operational costs which are automatically deduced. Income is generated by trains and industries.

- Industries generate 1,5k income per good they produce.
- Rural businesses generate 500 income per good they sell.
- Trains generate income bases on their wagonload:
    - delivered goods award 1000 income per level of need they satisfy,
    - delivered mail awards 100 income per day of travel from their origin,
    - passengers award 10000 income minus 500 per day of travel with a minimum of 2000.

If a player runs out of funds, they can't build anymore. If debt accumulates to more than 50k, they lose the game. In such an event, all their assets are sold (removed from the game).

Frontend implementation details

The frontend must serve a responsive website that is implemented with using a framework and respecting at least HTML5 and CSS3. If other technologies are used, they must use their most up-to-date, respectively stable versions.

The frontend is implemented in **FUWEB**, **JAVA3**, and **MOBAP** using different technologies.