

# Final project

## Databases for Developers

The final project assignment is to develop the backend part of a web application and deploy it to the cloud. You need to implement 3 solutions:

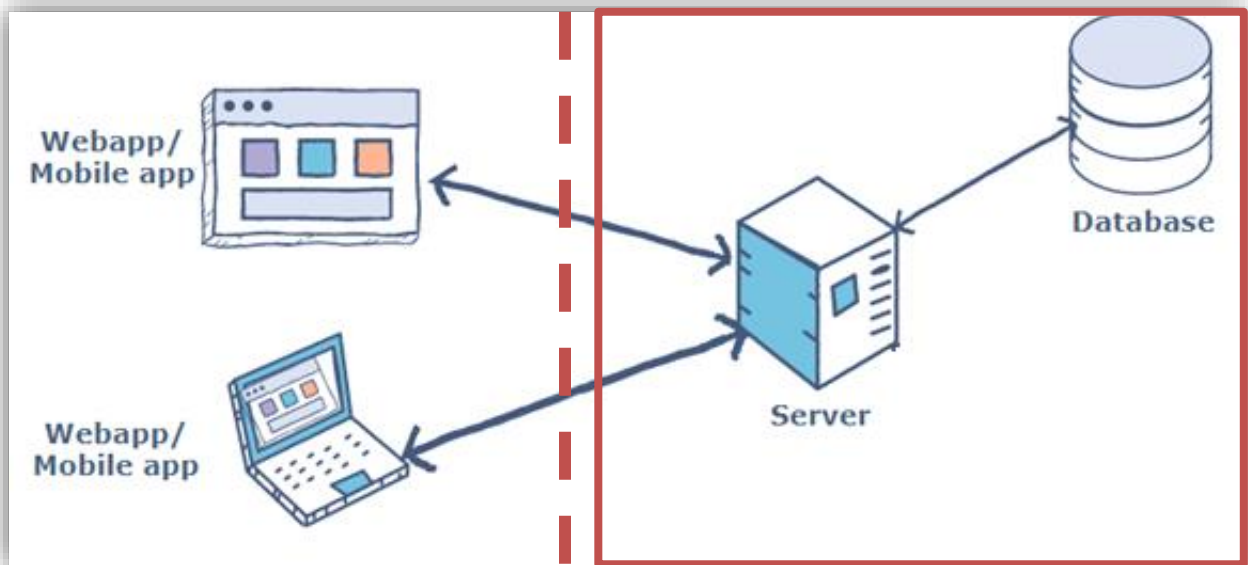
- relational database,
- document database,
- graph database.

**Here are some examples of the possible projects:**

- car rental system
- art collection system
- system for reviews of concerts and music releases
- banking system
- web shop
- movie rental / streaming system
- system for user reviews and recommendations of best restaurants etc – like [yelp.com](https://www.yelp.com)
- portfolio management system – for stock trading
- airline reservation system
- library management system
- job portal
- RPG – role-playing game
- ...

**Databases should be complex enough to cover the curriculum of the whole course. For example, a relational database should have at least 10 tables.**

*We are aiming for a following architecture where we focus on the backend part:*



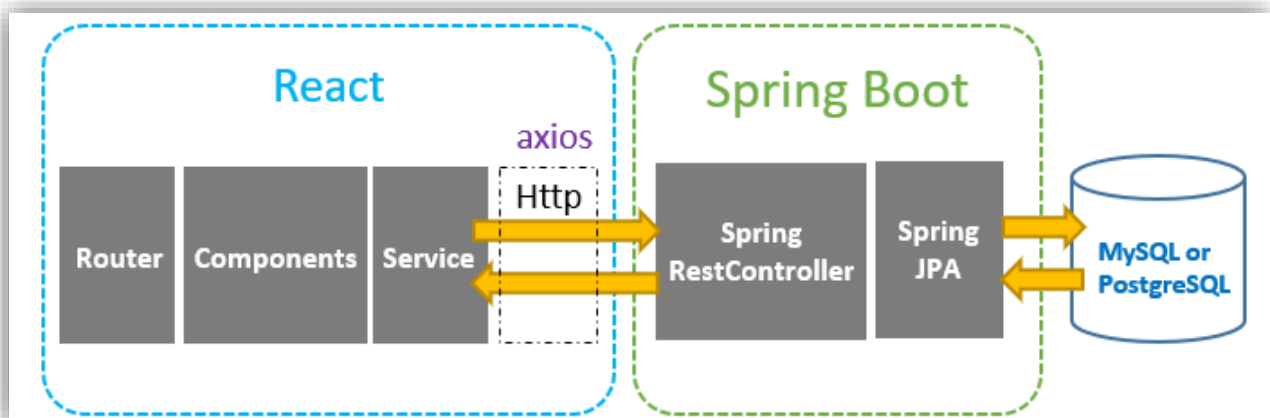
As you can see, we are working with a “monolith” design. We don’t want to complicate things with more complex architecture like using microservices etc. Instead, we want to focus on dealing with databases from the software developer perspective.

To practice different types of databases we will implement 3 different data sources:

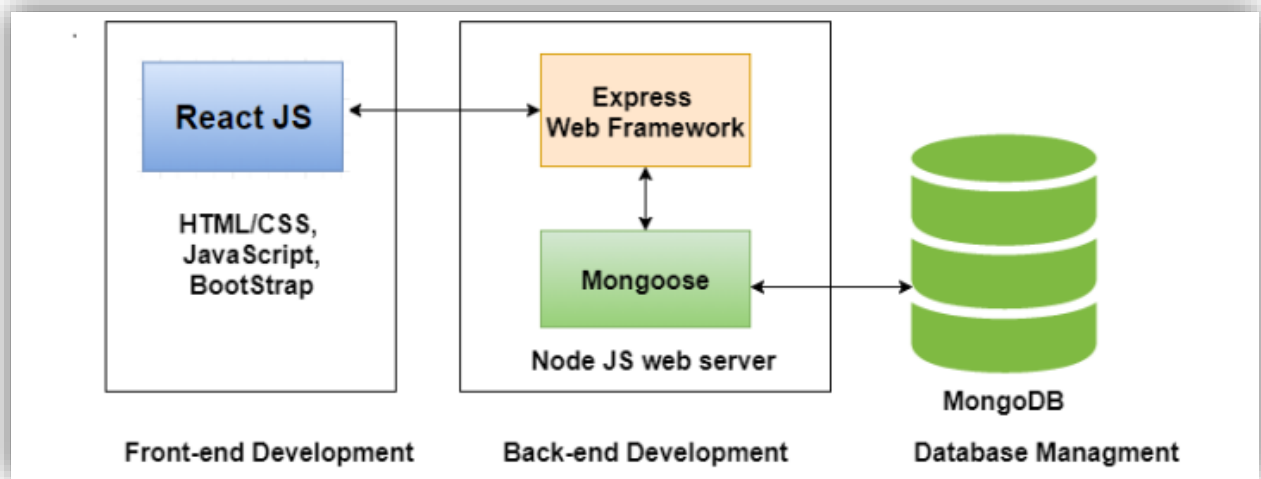
- Relational Database
- Document Database
- Graph Database

Examples of the system architecture (**Frontend is not part of the project**):

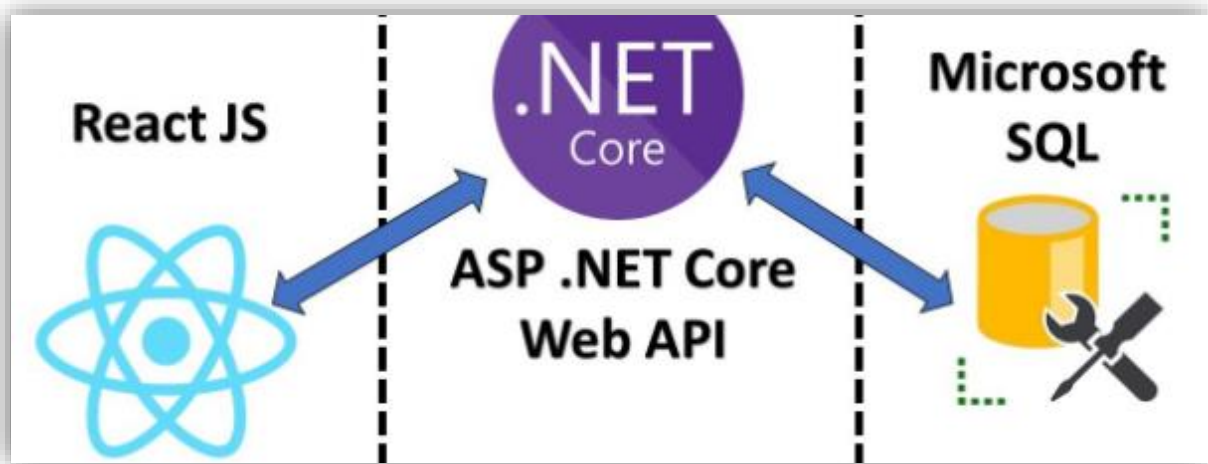
*React.js + Spring Boot + MySQL*



### *React.js + Node.js + MongoDB:*



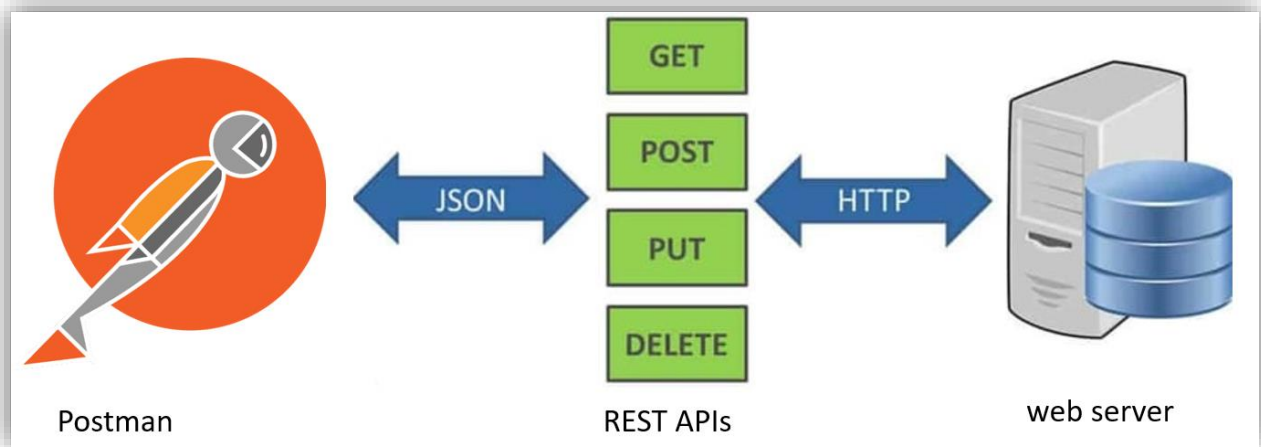
### *React.js + ASP.NET + SQL Server:*



### *Substituting the frontend with Postman or Swagger UI:*

The focus will be on the database and the backend server. Use Postman, Swagger UI, etc. to simulate the interaction with a web client. If you have a frontend, it is fine but it will not be the topic for the report and the exam.

## Using Postman



### *Project will consist of the following:*

#### **1.Part: Relational Database**

- Database design.
- A CRUD application
- Documentation for the database and the app

#### **2.Part: Document Database**

- Database design.
- A CRUD application
- Documentation for the database and the app

#### **3.Part: Graph Database**

- Database design.
- A CRUD application
- Documentation for the database and the app

### API documentation example:

Request	Endpoint	Body (x-www-form-urlencoded)	Description
GET	localhost:8080/product/show		This will return a list of all products. Initially there are 500 products stored in the Database.
POST	localhost:8080/product/search	key="word", value="lamp"	This will give back a Product that matches specific search string in its name. To search for lamps for example, use the value shown here.
POST	localhost:8080/credit_card/store	key="userId", value="1"	This request will store a new Credit Card for the selected User, providing userId

It is recommended to use the same API URLs for all 3 solutions. Most frameworks allow you to use [Swagger](#) to create a documentation for your APIs:

#### Example:

- .../mysql/products
- .../mongodb/products
- .../neo4j/products

You are free to choose whichever tools you consider appropriate. The recommendations are MySQL as relational database, MongoDB as a document database, and Neo4j as a graph database because we will use them during the lectures. However, you may choose any other databases of the same types if all the requirements are fulfilled.

For the CRUD application, you are free to choose any development stack – for example JavaScript or TypeScript for Node.js / Express,

- Java / Spring Boot,
- C# / ASP .NET,
- Python / Django,
- PHP / Laravel,
- ...

***The final project should contain following elements:***

- An Entity/Relationship Diagram.
- A relational database, including its tables, primary and foreign keys, indexes, constraints, and referential integrity checks.
- Test data.
- Transactions.
- Stored objects: Views, Stored procedures, Stored functions, Triggers, Events.
- An audit solution involving the use of triggers.
- Definition of users and privileges.
- A CRUD application covering the desired functionality.
- A solution using MongoDB or another document database:
  - Database design
  - CRUD application
- A solution using Neo4j or another graph database:
  - Database design
  - CRUD application
- Cloud deployment
- Integration tests (cooperation between the application and the database)

**Test data**

At the time of delivery, the relational database will contain data – all the tables should be filled with some test data.

**Database user privileges**

Users will be defined at database level. There will be, at least:

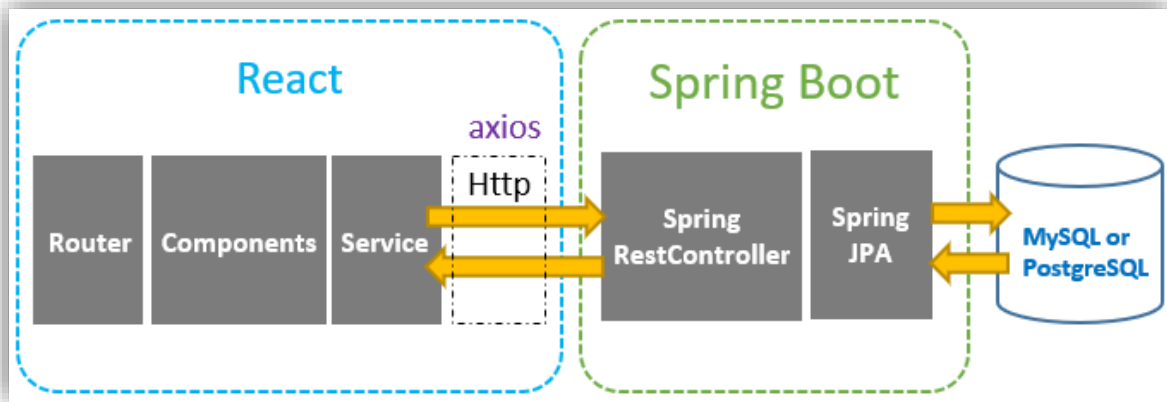
- A user for the application (with the minimum privileges it needs)
- A user with full database admin privileges
- A user with read-only privileges
- A user with restricted reading privileges, which will be unable to see some data

We need to implement users and roles. Then we can assign roles to the users. When a user logs in, his/her rights will be defined by the assigned roles – a user can have multiple roles. For example, an admin could be a customer as well...

We need to define a user for the CRUD application itself for connecting to the database server. The application should not have admin rights, it should only have the minimum rights it needs.

CRUD application (the frontend is not part of this project) - Examples with Spring Boot:

MySQL solution (or another relational database)



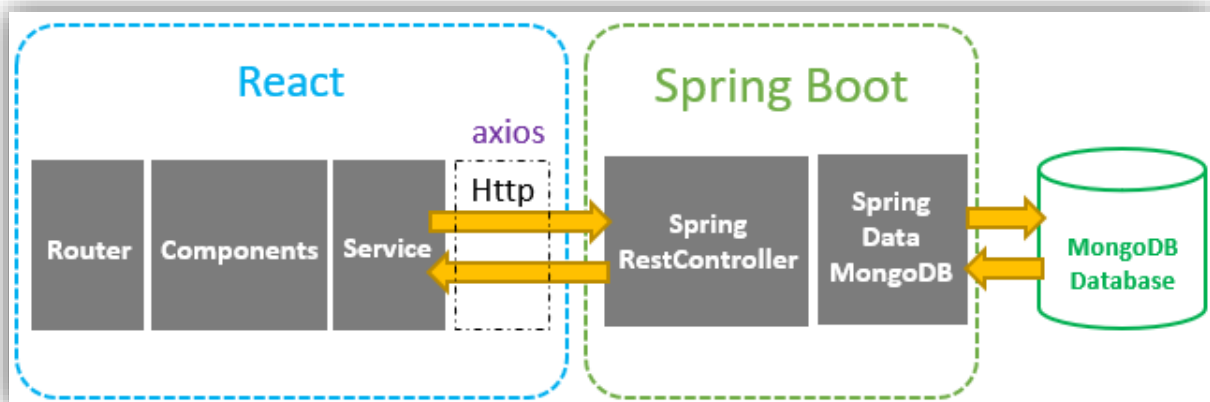
A CRUD application (create/read/update/delete) functionality to implement can be:

- Login/logout.
- Query data from the tables.
- Functionality depending on the business model

The application will comply with the following:

- All database queries will be prepared to avoid SQL injection.
- There will be a multiple search option which will search products by several fields, also implementing full text search.

MongoDB solution (or another document database)

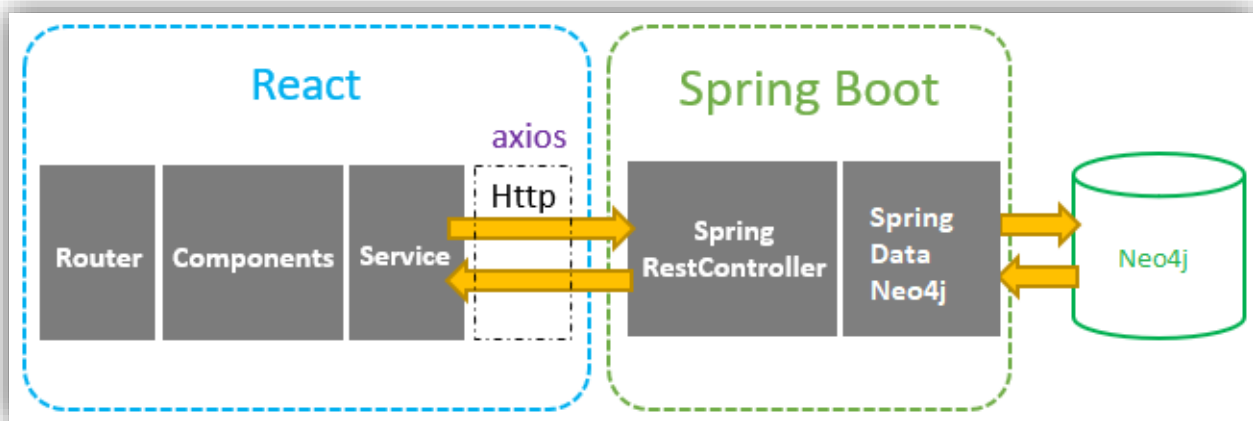


The goal is to implement the same functionality (or at least most of it) as with the relational database.

You can either add the document database as a parallel data source to existing web server (and create parallel REST APIs or GraphQL API) or you can create a new web server application just for this database.

To move from the relational model to document design, we usually need to de-normalize our relational model. For example, if we have tables like customers, orders, order\_items, products we can have all this information in a customer document which will contain multiple levels of embedded documents – a customer document will contain an array of order documents. Each order can contain multiple order\_item documents...

### Neo4j solution (or another graph database)



Part of the project is working a graph database. The goal is to implement the same functionality (or at least most of it) as with the relational database.

You can either add the graph database as another data source to the existing web server (and create parallel REST APIs or GraphQL API) or you can create a new server application just for this database.

Graph databases are very different from the relational databases. A graph database consists of nodes and relationships which makes it easy to traverse between nodes – for example if we want to know who a friend of a friend of a friend is... (social media app) or if we want to know what other products were purchased by customers who ordered or looked at a particular product... (product recommendation).

### Summary

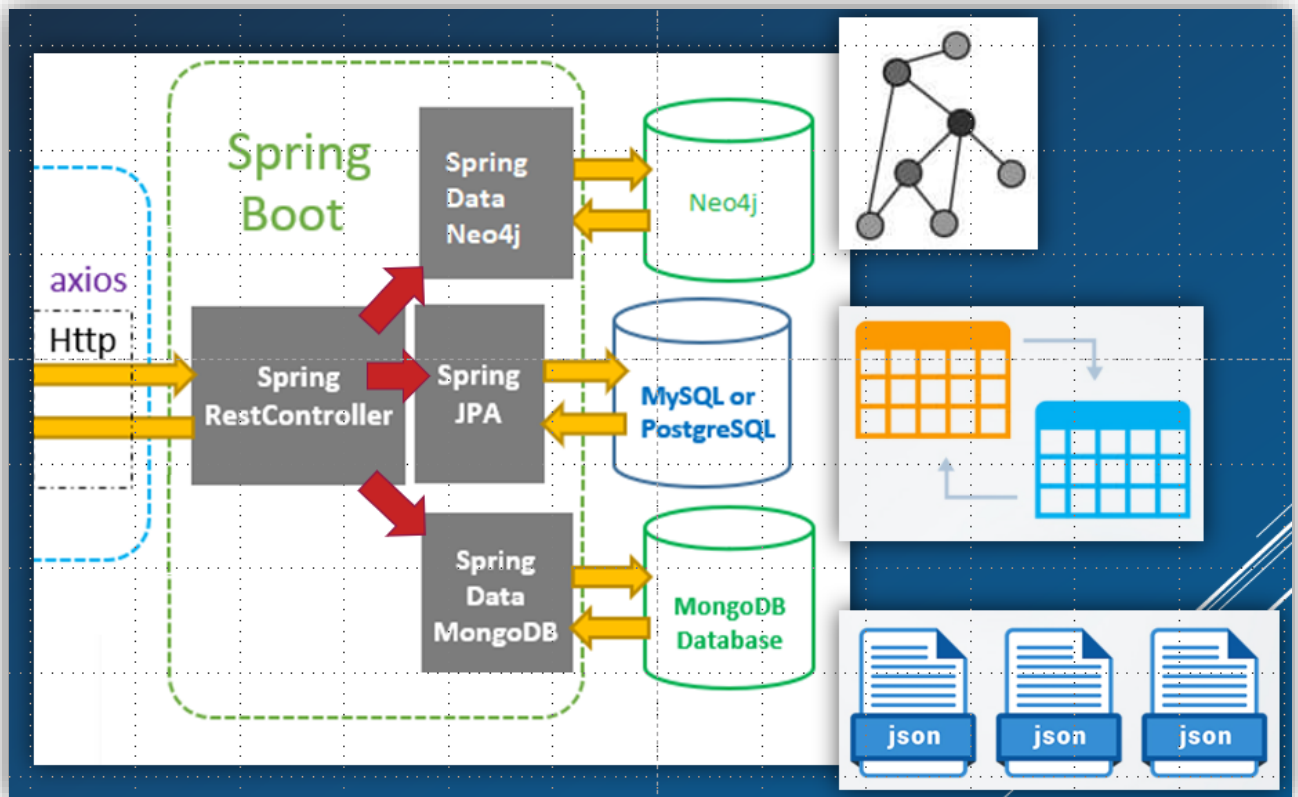
- We start with creating a system which uses an RDBMS like MySQL.
- Then we implement the same / similar functionality with MongoDB (or similar DB).
- Then we implement the same / similar functionality with Neo4j (or similar DB).



Implementing the same functionality with different database technologies simulates a scenario when a company decides to switch from one database technology to another.

It will also serve for comparing the strengths and weaknesses of used database types.

### *Example of all 3 solutions in one Spring Boot project*



### *Final Project Delivery*

The final group project will be uploaded individually to WISEflow. It will consist of a report and a series of artifacts.

#### **Final Project Artifacts (things to deliver)**

- Relational database scripts (one or several), including:
  - Database creation, including tables, keys, indexes, constraints, and referential integrity checks.
  - Load of test data
  - Stored procedures
  - Triggers
  - Views
  - Events
  - Creation of users and privileges

- The source code of the CRUD application - included as a link to an external public code repository (like GitHub).
- For MongoDB system:
  - dump file of the document database
  - Script for loading the test data
  - The source code of the CRUD application.
- For Neo4j system:
  - dump file of the graph database
  - Script for loading the test data
  - The source code of the CRUD application.
- A brief installation procedure that specifies how to organize the code and import the databases in a test environment with full operational capabilities.

## Final Project Report

The final report will have a big influence on the exam grade. It is the only information that is accessible to the external censor.

The report will have following structure:

Cover page, including:

- Title
- Full names of all students in the group
- Group number
- Date of delivery

List of figures

List of appendices

Table of contents (paginated index)

1. Introduction

1.1. Problem description (+ schema of the whole system)

1.2. Explanation of choices for databases and programming languages, and other tools.

**2. Relational database**

2.1. Intro to relational databases

2.2. Database design

2.2.1. Entity/Relationship Model (Conceptual -> Logical -> Physical model)

2.2.3. Normalization process

2.3. Physical data model

2.3.1. Data types

2.3.2. Primary and foreign keys

2.3.3. Indexes

2.3.4. Constraints and referential integrity

2.4. Stored objects – stored procedures / functions, views, triggers, events

2.5. Transactions. Explanation of the structure and implementation of transactions

2.6. Auditing. Explanation of the audit structure implemented with triggers

## 2.7. Security.

### 2.7.1. Explanation of users and privileges

### 2.7.2. SQL Injection – what is it and how it is dealt with in the project?

2.8. Description of the CRUD application for RDBMS – REST APIs, service layer, security – registration / login, transactions, etc.

## 3. Document database

### 3.1. Intro to document databases

### 3.2. Database design + features like indexes, PKs, constraints, etc.

3.3. Description of the CRUD application for the document database – REST APIs, service layer, security – registration / login, transactions, etc.

## 4. Graph database

### 4.1. Intro to graph databases

### 4.2. Database design + features

4.3. Description of the CRUD application for the document database – REST APIs, service layer, security – registration / login, transactions, etc.

5. Conclusions (discussing the similarities and differences between used database types)

6. References

## Report formal requirements

The formal requirements for the report can be found in the course catalog, in the exam section:

<https://katalog.ke.a.dk/course/9942152/2024-2025>