Seminar 08 - Working with the database

Agenda

- Database - best practices
- Using Prisma as the ORM tool to work with databases
- Express.js - creating a web server
- Twig.js - rendering html for the browsers

Database - best practices

- Write ERD for the database
- Model the tables according to the ERD
- Deleting data: records should have a visibility attribute (f.e. `deletedAt`) - deleting tables can cause issues
- Separate tables for addresses, prices and data that can change over time
- Storing multimedia in the database - a BAD idea (when talking about relational DBs) - databases are often cached in-memory
- Primary keys should always be either UUIDs or integers with autoincrement function
- Joining many-to-many relations done via join tables

Prisma - Install

- Add TypeScript to the project, as done in the previous seminar
- Add Prisma to the project

```
npm i prisma
```

Extend the `tsconfig.json` from the last seminar with these lines - **if** your code does not compile (however, there should not be any issues)

```
  "compilerOptions": {
    "sourceMap": true,
    "outDir": "dist",
    "strict": true,
    "lib": ["esnext"],
    "esModuleInterop": true
  }
```

Prisma - Schema & Migrations

This command will bootstrap the Prisma in the repository:

```
npx prisma init
```

Created files: `prisma/schema.prisma` and `.env` file with the database connection string.

Schema contains our table definitions.

```prisma
model Artist {
  id              Int      @default(autoincrement()) @id
  name            String   @db.VarChar(255)
  verified        Boolean  @default(false)
  profilePicture  String?
  coverPicture    String?
  description     String
  albums          Album[]
}

model Album {
  id              Int      @default(autoincrement()) @id
  artist          Artist   @relation(fields: [artistId], references: [id])
  artistId        Int
  name            String   @db.VarChar(255)
  releaseDate     DateTime
  description     String
  coverPicture    String?
}
```

Schema - example

Connecting to the database

The connection string is stored in the `.env` file - Prisma uses it to create a connection to the DB:

```
DATABASE_URL="postgresql://johndoe:randompassword@localhost:5432/mydb?schema=public"
```

- **NEVER** commit these files - they should never be tracked by the versioning software

If you have your own computer, you can run a Postgres database in a container via the provided `compose` file

Connecting to the database - on school computers

- We, unfortunately, cannot run containers at school 😔
- We can, however, connect Prisma to `sqlite` database, which is by default available (at least on the `nymfe` machines) on school computers
- Create a file `database.db` in the `prisma` folder
- Modify the portion of the `prisma.schema` file:

```
datasource db {
  provider = "sqlite"
  url      = "file:./database.db"
}
```

Now you can follow along with the seminar!

Prisma - Schema & Migrations

After writing the schema, we need to generate a migration.
Migration is a file with SQL definitions, which defines the database tables.
Every schema change must be reflected by running another migration (which will update the DB) and re-compiling the Prisma client.

```
npx prisma migrate dev --name init
```

This command will also generate a new client for us

Adding Prisma to the code

```
import { PrismaClient } from '@prisma/client'

const prisma = new PrismaClient()

const main = async () => {
  // ... you will write your Prisma Client queries here
}

main()
  .catch(e => {
    throw e
  })
  .finally(async () => {
    await prisma.$disconnect()
  })
```

Express.js - Web Application Framework

- Framework that allows quickly building web applications
- Provides a very minimal, precise set of tools necessary for creating web applications
- Used by many other JS/TS frameworks as their backbone

Express - Install

```
npm i express
npm i -D @types/express
```

Adding express to the code

```
import express, { Express, Request, Response } from 'express';

const app: Express = express();
const port = 8080;

app.listen(port, () => {
  console.log(`Server is running at https://localhost:${port}`);
});
```

Express - routes and handlers

- The data is firstly processed by a pipeline of functions called middleware
- These functions can, for example, check privileges or handle things that need to happen to every request before it is processed individually
- The request then gets processed via a router
- Router then routes the requests it defines the flow of individual requests
- Each route has an assigned handler - a function that processes the request individually

```
app.get('/', (req: Request, res: Response) => {
  res.send('Express + TypeScript Server');
});
```

Express - Server-side rendering

In contrast to the later part of the course, which specialises in Single Page Applications, dynamically displaying content from the database can also be done via **server-side rendering**.

For example, popular CMS (content management system) WordPress uses this approach.

We need to define an HTML template (+ styles with CSS), which will be filled with the data loaded from the database.

For templates, we will use a popular library called **Twig.js**.

Twig - Install

```
npm i twig
npm i -D @types/twig
```

Adding Twig to the code

```
import Twig from 'twig';
import express from 'express';

const app = express();
const port = 3000;

// This section is optional and used to configure twig.
app.set("twig options", {
    allow_async: true, // Allow asynchronous compiling
    strict_variables: false
});

app.get('/', (req, res) => {
  res.render('index.twig', {
    message : "Hello World"
  });
});

app.listen(port);
```

All three technologies together

```
import Twig from 'twig';
import express from 'express';
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();
const app = express();

app.get('/', async (req, res) => {
  const albums = await prisma.albums.findMany({
    where: {
      name: {
        equals: "Thriller"
      }
    }
  });
  res.render('index.twig', {
    albums: albums
  });
});

app.listen(3000);
```

Demo - Spotify database & basic server-side rendering

Hands on: Iteration 06

You can find the assignment in GitLab issues as well as in the interactive syllabus.

Let's take a look together.

Before you start:

- Please check whether your tutor has already accepted your MR
- If they have, make sure you have merged your solution from the previous week

*Note: if your tutor has **not** seen your MR, it's completely ok. You do **not** need to have the previous iteration merged to be able to work on a new one - **iterations are independent**. However, if you **do** have an accepted MR that still has not been merged, make sure to merge it first.*