CMPS 350 Web Development Fundamentals Spring 2023
Lab 11 – Data Management Using Prisma and Postgres Database

Objective

You will learn how to use

- PostgreSQL as the database
- Prisma as the ORM for migrations and database access.
- Modelling Data Using Prisma

Overview

This Lab is based on Lab 10 Banking App. Therefore, you are required to implement a database to deliver the same functionality as the file-based system provided in the base solution.

The tasks for this Lab are:

- Implement and test the Banking App database schema and repository methods.

Project Setup

1. Download "Lab11-Data Management" from the GitHub Repo and copy it to your repository.

2. Make sure you have Postgresql database installed in your computer from https://www.postgresql.org/download/]

You can follow this step by step guide https://www.datacamp.com/tutorial/installing-postgresql-windows-macosx

Banking App

Open the **BankingApp** in VsCode and follow the steps below.

Connecting to PostgresQL Database Using Prisma

1. Install the **prisma** package using **npm install prisma --save-dev**
2. You can now invoke the Prisma CLI by prefixing it with npx
    a. npx prisma
3. Next, set up your Prisma project by creating your Prisma Schema file with the following command **npx prisma init .** This command does two things:
    a. creates a new directory called **prisma** that contains a file named **schema.prisma,** which contains the Prisma schema with your database connection variable and schema models
    b. creates the .env file in the root directory of the project, which is used for defining environment variables (such as your database connection)

4. To connect your database, you need to set the **URL** field of the **datasource** block in your Prisma schema to your database connection URL inside the ***prisma/schema.prisma and .env files***

prisma/schema.prisma

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

In this case, the `url` is set via an environment variable which is defined in `.env`:

.env

5. Replace the URL with ***DATABASE_URL="postgresql://postgres:postgressql@localhost:5432/bankdb"***

Creating the Database Schemas and Models

The class diagram below shows the entities of the Banking App.

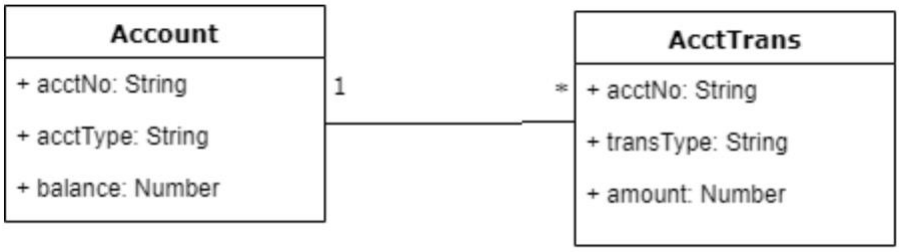

Figure 1 Banking Entities Diagram

1. Add the the above Prisma data model to your Prisma schema in **prisma/schema.prisma**:
2. Inside the ***model*** directory create two models and name them "**account**" and "**transaction**"
3. You should make a one to many relationship between the two models as shown in figure 1.
4. Export the models to your PostgreSQL database by using the following Prisma command **npx prisma migrate dev --name init**
5. Anytime you make changes to the models, you need to **npx prisma migrate dev ,** if the changes are only related to type then you can use **npx prisma db push**
6. To view your database, run the following command **npx prisma studio**

Sending queries to your database with Prisma Client

1. To get started with Prisma Client, you need to install the @prisma/client package **npm install @prisma/client**

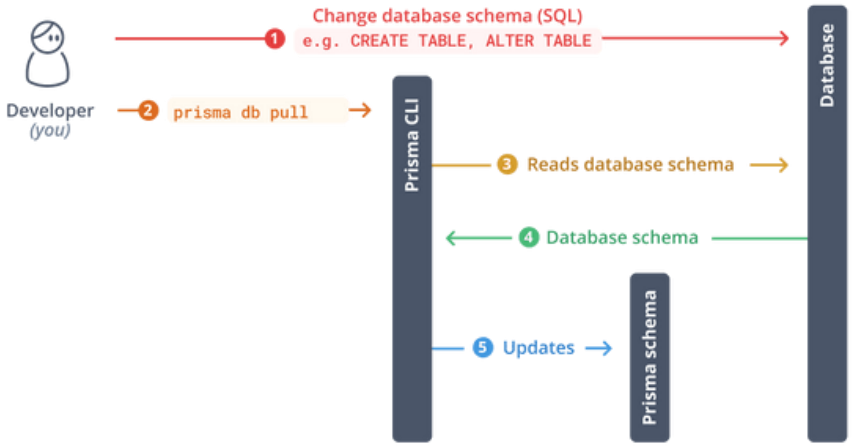

Figure 2 Example Prisma schema for a PostgreSQL database. Adapted from Prisma (source).

2. Implement the following repository methods using the **Account** and the **Transaction** Models.
    a. **getAccounts(type):** a method that returns a list of accounts, filtered by account type if specified. It uses prisma ORM to query the database.
    b. **addAccount(account):** a method that adds a new account to the database
    c. **updateAccount(account, accountNo):** a method that updates an existing account in the database
    d. **getAccount(accNo):** a method that retrieves an account by account number
    e. **deleteAccount(accNo):** a method that deletes an account from the database based on account number,
    f. addTransaction(transaction, accountNo): a method that executes a transaction (either deposit or withdrawal) on an account and adds a record of the transaction to the database using prisma. It calls getAccount and updateAccount methods internally to update the balance.
3. Ensure you implement a method to load the **data**/**accounts.json** data to the **PostgreSQL** Accounts collection to initialize.
4. Test each method using **Mocha** or **Postman**.
5. Test using the user interface you implemented in Lab 10.