Week 07: ERD, modeling, Prisma basics

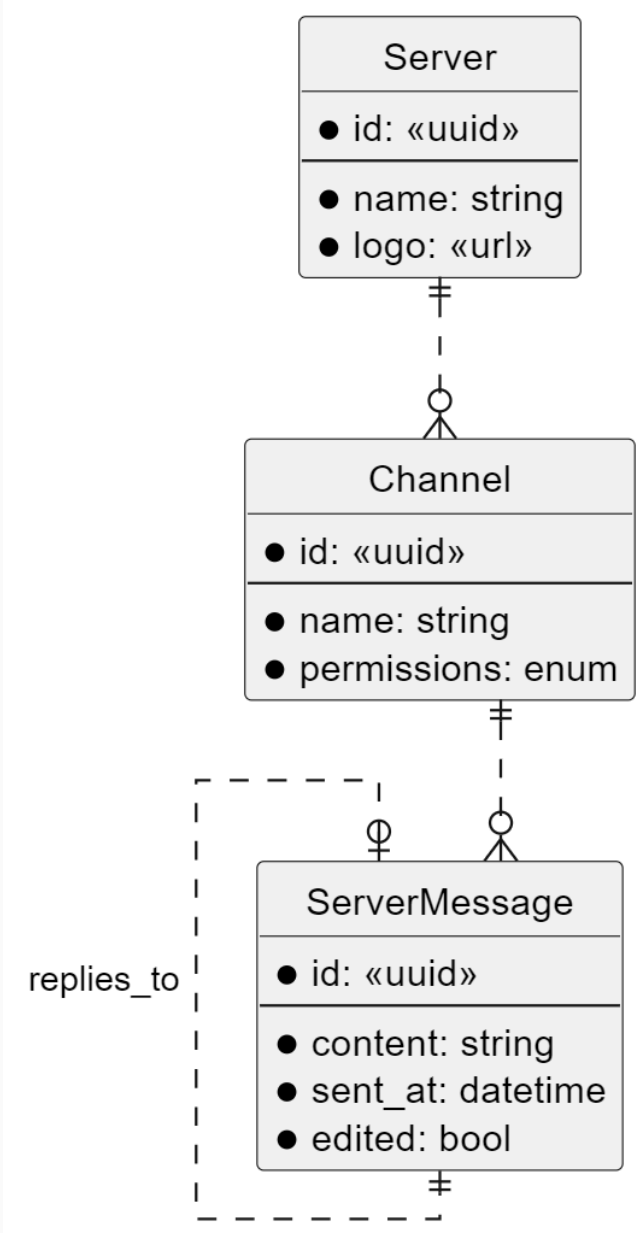
Agenda

- Modeling exercises
- From model to database schema
- Creating simple queries with Prisma and seeding the database

PlantUML

PlantUML is a tool for creating UML diagrams through plain text language, enabling easy version control and sharing. It supports various diagram types, including ER Diagrams.

```
@startuml lab07-diagram

hide circle

skinparam Linetype ortho

entity ServerMessage {
    * id: <<uuid>>
    ---
    * content: string
    * sent_at: datetime
    * edited: bool
}

entity Channel {
    * id: <<uuid>>
    ---
    * name: string
    * permissions: enum
}

entity Server {
    * id: <<uuid>>
    ---
    * name: string
    * logo: <<url>>
}

ServerMessage |o..|| ServerMessage: replies_to
Server ||..o{ Channel
Channel ||..o{ ServerMessage
@enduml
```



Useful links

- Online editor
- VSCode extension
- JetBrains extension

(The extensions require a working Java installation)

Database modeling: Level 1

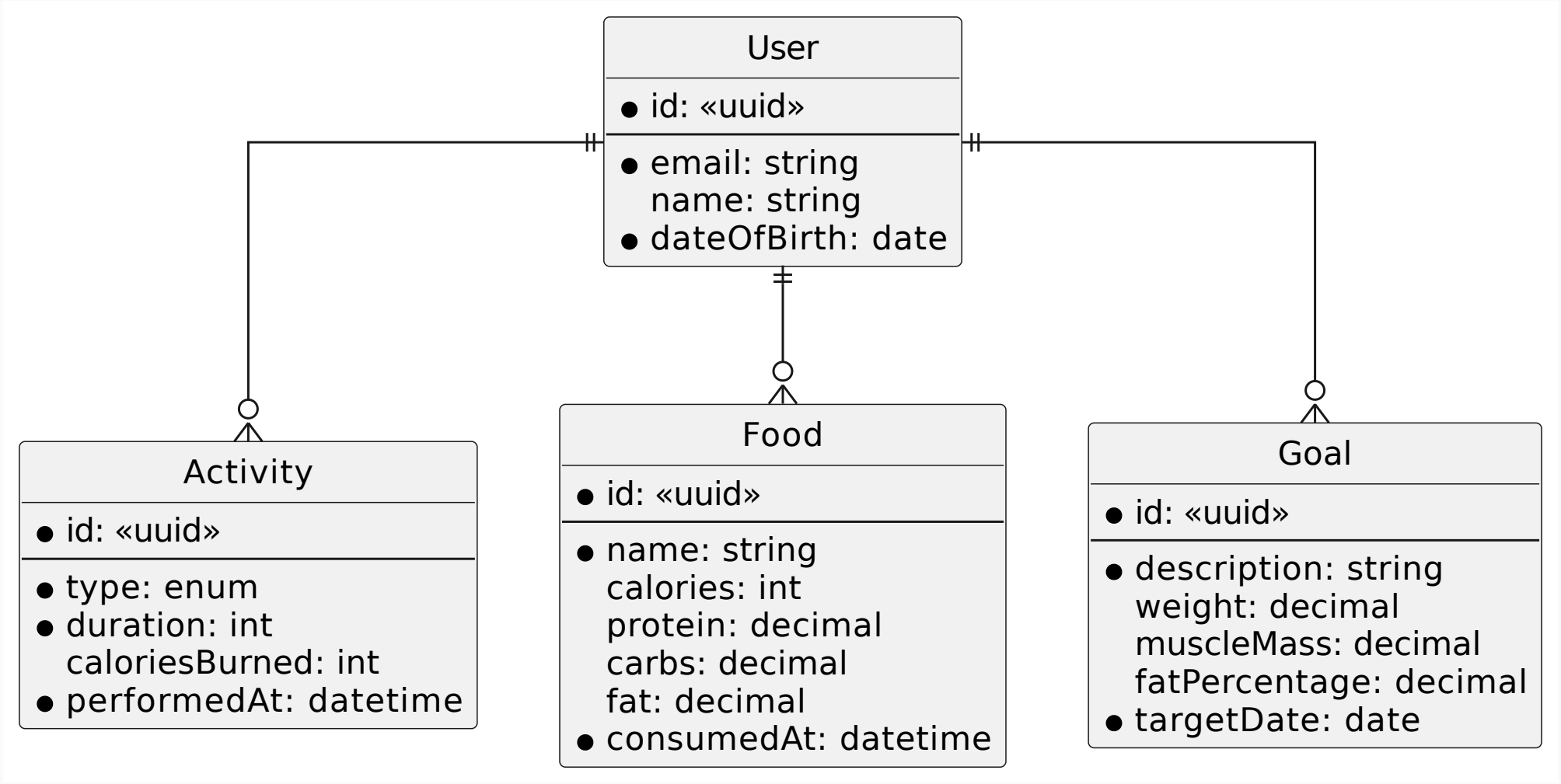
Design a database model for a fitness tracker application using PlantUML. This application aims to help users monitor their fitness routines, dietary intake, and progress towards their health goals. Your database should include the following entities, at a minimum:

- **User:** Represents the users of the fitness tracker app.
- **Activity:** Records various activities that users can perform, such as running, swimming, cycling, etc.
- **Food:** Details the dietary intake of the users - meals and their nutritional values.
- **Goal:** Sets fitness or dietary goals that users aim to achieve - weight, muscle mass, etc.

Think about

- What types of relationships will you use? (one-to-one, one-to-many, many-to-many)
- What attributes will the entities have?
- Which of the attributes will be required?

Example solution



Discussion

- **Tracking Progress Over Time:** Suppose the application needs to support tracking users' progress over time, such as weight changes or improvements in activity performance. How would you adjust the diagram to accommodate this functionality? Would you add new entities, attributes, or relationships?

- **Reward System:** Suppose the application wants to motivate users by implementing a reward system based on achieving goals. What adjustments or additions to the diagram would be needed to support this?

Database modeling: Level 2

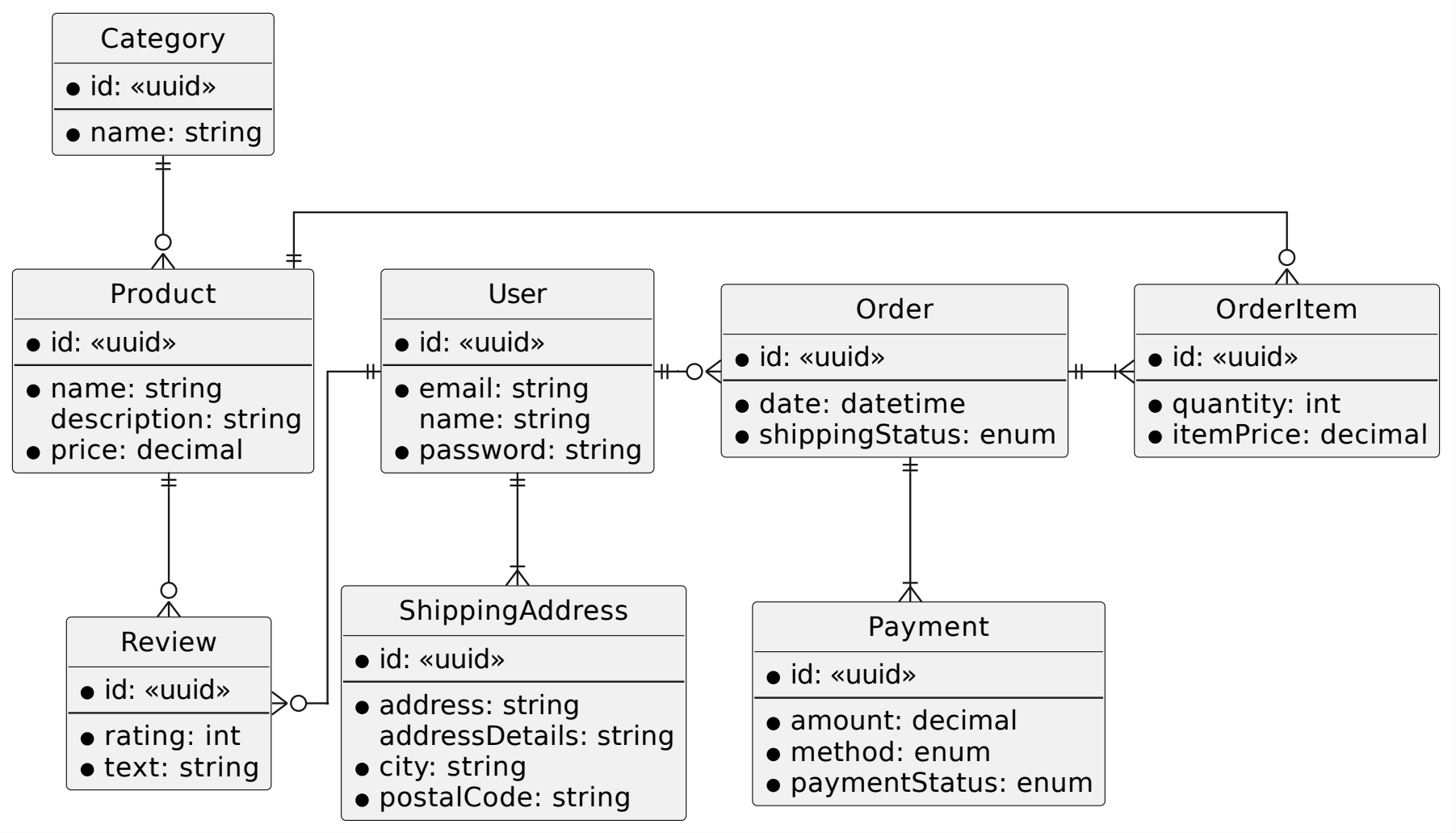
Let's raise the complexity a bit. Design a database model for an e-commerce platform. Your model should facilitate core e-commerce functionalities, including product listings, user accounts, orders, and payment processing. Your database should include the following entities, at a minimum:

- **User:** Customer accounts on the e-shop platform.
- **Product:** Items available for purchase.
- **Order:** Records of purchases made by users.
- **OrderItem:** Individual items within an order.
- **Payment:** Payment information and status.
- **Category:** Product categorization.
- **Review:** User-submitted product reviews.
- **ShippingAddress:** Information about shipping addresses.

Think about

- What types of relationships will you use? (one-to-one, one-to-many, many-to-many)
- What attributes will the entities have?
- Which of the attributes will be required?
- What data types will you use for the attributes?
- What cardinality should the relationships have?

Example solution



Discussion

- **Product price:** The price of the product is stored both in the `Product` and `OrderItem` entities. Is this a good design choice? Why or why not?
- **Modeling Hierarchical Categories:**
  Currently, our model does not explicitly support a hierarchical category structure, such as subcategories within categories. How could you adjust the `Category` entity to allow for this hierarchy?

Converting your diagram into a Prisma schema

Firstly, initialize Prisma & TypeScript in a blank Node.js project

```
npm i -D typescript prisma @types/node tsx
```

Then initialize TypeScript & Prisma in your project

```
npx tsc init
npx prisma init
```

Converting your diagram into a Prisma schema

Set up a script in the `package.json` file which will start up our code file with these arguments:

```
{
    // ...
    "scripts": {
        "start": "tsx seed/seed.ts",
    },
    // rest of the file ...
}
```

- **Now, convert your ERD into a Prisma schema**. If you're stuck, you can look into the documentation here.

Run your database and migrations

First, let's start up the database

```
docker run --detach -e POSTGRES_USER=user -e POSTGRES_PASSWORD=password \
  -e POSTGRES_DB=database -p 5432:5432 --name pb138-seminar-07-database postgres:latest
```

Now, set the correct credentials in your `.env` file

```
DATABASE_URL="postgresql://user:password@localhost:5432/database"
```

Make sure your Prisma schema is ready for migration! Afterward, you can run your migration with:

```
npx prisma migrate dev --name 'name of your migration'
```

And you can look into your database with:

```
npx prisma studio
```

Create a simple seeding script

- Create a folder `seed/` and in it, create the file `seed.ts`
- Paste the following code block into it:

```
import { PrismaClient } from '@prisma/client'
const client = new PrismaClient();
async function seed() {
// ... you will write your seeding logic here
}

seed()
  .then(async () => {
      await client.$disconnect()
  })
  .catch(async (e) => {
      console.error(e)
      await client.$disconnect()
      process.exit(1)
  });
```

- Create the seeding logic. But... Where do you obtain data?

Now for the seeding part

- Seed the database with `@faker-js/faker` package!

```
npm i -D @faker-js/faker
```

- Look into the documentation to see how to use this library.
- Seed the database with simple Prisma queries you should be familiar with from the lecture (nested `create` or `createMany`). If not, also look in the Prisma documentation - create & createMany.

Stopping the postgres container

To stop the `postgres` container, run the following:

```
docker stop pb138-seminar-07-database

# To remove:
docker rm pb138-seminar-07-database
```

That's it for today!