Lecture 2

Databases, ORMs, Migrations, Entity relations, Filters and Reports

Databases

- Every web app needs a database to store data in
- But working with databases can be tedious
- Many REST Frameworks offer an abstraction to databases that is sufficient in many cases
  - Sometimes we might still have to resort to working directly with databases for efficiency reasons
  - This abstraction makes use of the fact that tables and classes are quite similar conceptually
  - Those that don’t offer it can easily be integrated with dedicated ORM libraries

Object Relation Mapping (ORM)

- This is the set of techniques used to abstract (mostly relational) databases behind classes
- It lets us with plain objects whose fields and methods are mapped to databases tables and queries behind the scenes
- Advantages
    - Significantly reduces the amount of code we have to write
- Disadvantages
    - Generated schemas and queries can be suboptimal or outright inefficient
    - High level of abstraction can make things hard to understand and debug
- We will soon start asking that you use ORMs and not write and raw SQL

Migrations

- The domain can change, and the database needs to keep up
- This is done through **migrations**: a migration analyses the changes in your data model relative to the current database structure and executes the needed queries to make the database match your data model
- Your ORM framework should support migrations
- You need to create a migration every time you make changes to your models
- Be careful not to cause data loss or introduce incompatible changes: for example, making a field that has repeating data unique will cause an error
    - Sometimes you might need to deal directly with the database to fix such errors

Entity Relations

- Related entities will generally be handled through class fields: for example, a class Author might have a **List<Book> books** field
- These are referred to as **navigation properties** or **related fields**
    - This lets us work with them in a more natural way than the manual coordination you've seen in FP / OOP
- For many to many relations, we will generally still need to work through a third class, but navigation properties will help here as well
- Care must be taken regarding how these related fields are fetched, as it might introduce significant slowdowns

A common pitfall

The **N+1 selects problem** is a common issue when working with ORMs. It won’t matter much in your labs since you will have little data, but you will soon need to avoid it. Some keywords and links for this:

- https://stackoverflow.com/questions/97197/what-is-the-n1-selects-problem-in-orm-object-relational-mapping
- Lazy and eager loading

Filters and reports

- ORMs generally allow us to filter our data by simply doing the filtering as we normally would on a collection
- We are able to pass in lambda functions or framework-specific functions to get any results we want
- These are then converted to SQL for us
- Care must be taken regarding the generated SQL, as it might not be very efficient
    - If you do the processing in multiple steps, that will most likely generate multiple SQL queries
    - If you introduce nested loops, that will most likely generate nested SQL queries
    - If all else fails, you can write the SQL manually
    - There are usually operations that bring data into memory: these should generally only be used at the end

Let's look over some examples

- In Django, Spring Boot and Web API