Springboot

To reload proj after you change smth in the pom.xml, have to right click the pom.xml file → Maven → reload project

If havent connected to db, remove the starter data jpa dependency first

With inversion of control, Spring basically scans your project for classes and creates instances of objects (beans) for every class that is annotated with bean-related annotations (such as "@Configuration", "@Repository", "@Service" "@Controller" etc..). All of these objects are stored in the Spring Context, which you can see basically as an array of beans.

This allows you to simply declare relationships between beans and Spring will go find in its context if a corresponding bean is available. If it is, it will inject it in your object without you needing to manually create an instance and pass it through to the constructor.

@Component vs @Service: not much diff, just semantics

Application.properties is smth like .env file

spring.datasource.url=jdbc:postgresql://localhost:5432/amigoscode

spring.datasource.username=

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.properties.hibernate.format\_sql=true

Create-drop means to have a clean slate everytime we start the app (empty db)

**Data access layer (functions that access database)**

Naming convention for the file: ClassNameRepository

Eg: StudentRepository

## Useful Annotations

**Component**

* class level
* When you want the class to be managed by the spring container
* Telling spring that this class is a bean → adds to the bean registry

**Service, Repository, Controller**

* Class level, stereotype annotations
* Similar to component, diff is that its providing metadata to spring
* Tells spring what kind of component is it
* Service: business logic
* Repository: data access objects, classes that interact with database
* Controller: web controller classes, HTTP requests, detected by Spring MVC framework -- do the job of being a web request handler

**Autowired**

* Activate the dependency injection

**Value**

* Property or setter annotation
* Inject external property value into bean
* Benefit: ask spring to set values from property files instead of hardcoded values
* @Value(“${propertyName}”) (eg database.url)
* @Value(“#{systemProperties[‘user.name’]}”)
* Can change the value without recompiling the code
* Class that contains the property has to be a spring bean

**Configuration**

* Code that is meant for configuring and initializing spring itself
* Class level
* Setup and wiring logic
* Allows for Autowired dependency injection

**ComponentScan**

* Can be used with @Configuration
* Auto detect beans across packages
* ???

**Bean**

* Method level
* For classes with @Configuration
* Wtv the method returns, take that and make it as a spring bean
* Method’s return object will be registered as a Bean
* Still not too sure whats the diff between using this and just doing stereotype annotation

**Transactional**

* Does the rollback??
* Does the transaction handling

**RequestMapping**

* Map HTTP requests in controller classes
* Class and Method level annotation
* Class: map all HTTP requests to that class for that URL
* Method: for fine grained method for that path

**RestController**

* Declares controller class that handles web requests and returns JSON response

**SpringBootApplication**

* Make an application a springboot application
* Combination of: Configuration, EnableAutoConfiguration, ComponentScan

# Postgres

Start the server in the postgres app first, then type into terminal

psql

**To list out all the databases**

\l

**To list out all roles**

\du

**To grant privileges to a role**

GRANT ALL PRIVILEGES ON DATABSE “databse\_name” TO “role\_name”;

**To create database**

CREATE DATABASE database\_name;

**To connect to database**

\c database\_name

Relations: \d