Lecture 3

Unit testing, Integration Testing, Deployment to a Server

Testing

- Unit and integration tests are a good thing to have, as they prevent developers from introducing bugs
- You will have to write tests for your application
- Unit tests generally test a single aspect or component of your application
- Integration tests generally test if multiple components work well together
- These applications are already quite complex: we have a database, repositories, services, controllers, domain models etc.

Testing

- We will write tests for everything not handled by the framework, such as:
    - Validations / configurations in data models
    - View / Controller logic
    - Services
- We will generally write two types of tests:
    - Those that test specific classes using asserts - you might need a mocking library for this
    - Those that test the actual REST Endpoints by using a REST client

Deployment to a server

Our application needs to be available to anyone with the URL to it. For this, we must deploy it to a server.

- You have free tier access to AWS EC2 Instances: https://aws.amazon.com/free/
- You have 300$ in credits for Google Cloud Platform: https://cloud.google.com/free
- This is enough for running small virtual machines to deploy our apps on and to present them during the labs
- Make an account on one of these and prepare to deploy your lab assignments

Deployment to a server

This is a rough overview of the steps you will need to perform for a basic deployment. It does not yet cover performance and security aspects - those will come later:

- Set up your VM so that it allows HTTP / HTTPS access
- Set up SSH access into your virtual machine: you can use the basic terminal for this or a client like Putty, Kitty, Termius etc.
- Install the necessary apps and tools on the server that are required to run your application (note that you will need to use the command line): Python 3, Java, node etc., also git. They might come preinstalled, so check first.

Deployment to a server

- Configure CORS for your project: you can just allow access to everyone for now
- Git commit & push the CORS configuration
- Git clone your project on the server
- If you use a database server, such as Postgres, you need to install and set it up on the server
- If you have migrations, run them
- Start your app from the command line and it should work
    - Make sure you run it listening on 0.0.0.0 and any port, not 127.0.0.1
- Navigate to your VM's URL on another machine (like your phone, or ask a friend to check it) and see if it works

Deployment to a server

Some things you might want to research yourself:

- Spring and Django embed their own web / app servers: they’re not very good as they are. Look into installing the web server separately for Spring and Gunicorn for Django
- Look into putting the Spring / Django web / app server behind Nginx
- Look into generating an SSL certificate using Letsencrypt
- Look into starting your app automatically as a service when the webserver starts
- Look into what settings Nginx and Gunicorn / other servers have that lets you control performance and security

Web server, app server, reverse proxy server

- A web server:
  - Accepts HTTP requests and delivers static content: HTML, images, other static files
  - The client is usually a web browser
  - Example: Nginx, Apache, Tomcat
- An application server:
  - Provides access to business logic through apps that generate dynamic content
  - Its clients are usually other applications, often a reverse proxy server
  - The communication protocols can vary: it might not be HTTP
  - Example: Gunicorn
- A reverse proxy server:
  - Sits in front of one or more web / app servers and distributes requests to them appropriately
  - Can act as a load balancer, security mechanism, caching mechanism
  - Example: Nginx