Portfolio 2: Showtime

DATA2410, Spring 2021

- This is a **group** assignment.
- **Deadline: Sunday May 23'd, 23:59.**
- There will be **no extensions** for any reason. If you're not done by then, hand in what you have.
- The assignment is a part of your portfolio and counts towards your final grade.
- Allowed programming languages are python3, javascript, java and C++.
- You must submit your source files (.py, .java, .js, etc.), AND include a PDF report with a description of your implementation with screenshots. The PDF file must contain **detailed instructions** on how to run your program and documentation of all the implemented functionality with screenshots showing what to expect when running.
- You can upload a .zip file that contains all relevant program files if needed, but keep the PDF report separate as it then shows up directly in canvas. For practical reasons it is a good idea to add your student numbers to the top of the PDF report.

Contents:

Introduction

You are to choose ONE of the two tasks given. Both tasks have some options that you can choose from (e.g. which game to implement and some technology choices), and some "stretch goals". The intention is that both tasks should give you equal opportunity to get a good grade. The stretch goals are things you do not have to do if you have low or average ambitions or if you're a very small group (1-2 people), but things you should consider if you have high ambitions, are a big group and really want your solution to stand out.

Task 1 is the safer choice in that it's more straightforward how to implement it using very well known tools and techniques. As the service itself is slightly simpler to implement than the game in Task 2 it has a few more requirements on the deployment side and will give you a little more experience with devops related technologies.

Task 2 is more playful and allows for more creativity and more pure programming, but will likely imply that you do some more research on your own e.g. for graphics. Remember to keep a strong focus on networking and cloud computing. Again; it should be possible to achieve good grades with both.

The tasks are expressed as simple user stories, e.g. things a user should be able to do. This is a common way to express software features.

Don't panic

If you can't get all the user stories working - remember: **Keep It Simple Stupid**. Something simple and stripped down that works is better than something complicated that doesn't. Don't introduce complexity to your system (like e.g. threading) unless you know why you have to.

Go for gold. Both of these exercises could become something worth showing a future employer. Use that opportunity. At the same time remember that the main objective is to show what you've learned about networking and cloud computing. While it's great to make a fun and impressive product don't lose that focus. Good luck and have fun!

Task 1: A web shop with a REST API

The task is to implement a basic web shop without a payment solution using a RESTful API (except that). You don't have to adhere to all the HATEOAS requirements, specifically your JSON results do not have to contain links.

Note that this course is not a web design course and especially for smaller groups you're not expected to make anything that looks fancy. There is no need for good looking CSS - simple default HTML look & feel is ok. That said, you're free to use CSS frameworks like bootstrap and javascript frameworks such as react or angular if you prefer.

User stories

As a user:

- I can see the store front page with a list of products, each with pictures or icons. (You can borrow the pictures or make your own.)
- I can click on a product to get more information about it and possibly bigger and more pictures.
- I can see what each product costs
- It's easy to see where to click if I want to buy the product. If I click there I can see that the number of products in my "shopping cart" increased
- I can easily see how to proceed to checkout or to complete my purchases.
- When I get to the checkout I'll be shown a list of all the products I've bought and a total price at the bottom. And then, by pure luck, I'll see that a special bonus applies to me right now that deducts the whole amount allowing me to get all those products for free.

As an administrator:

If you're a small group (1 or 2) the admin functionality can be considered a stretch goal.

- If I authenticate myself with a special administrator username and password provided in the documentation I can see that I get logged in with admin rights
- As an administrator I can add new products. Each product has at least a name, a short description and a longer description and a photo.
- If I don't add a photo a generic placeholder photo will be used instead.

Technology choices

Backend

1. **Python3** and Flask. Tutorial here.
2. **Node.js** and the express framework. Tutorial here.
3. **Java** with spring.

Feel free to use swagger to specify and document your REST API and to generate your stubs. Note however that the generated code can be harder to read and may require some time to figure out how to use.

Database

1. **MySQL**. Probably the most deployed database in the world. Recommended because it should be familiar to most students already.
2. **MongoDB**. NoSQL database effectively storing JSON directly as "objects" instead of rows and columns (it's really a binary json format). Instead of a custom query language it uses javascript for queries. Recommended especially for node.js setups because it gives you javascript/JSON all the way.
3. **PostgreSQL** is allowed, and possibly others (ask first), but it may be harder for us to help you out as we haven't used all of them.

Frontend

When building against a REST API you have to use javascript to fetch resources. While CSS is not required it will make your life easier. Bootstrap

1. MDBootstrap and plain javascript, based on fetch.
2. Plain Javascript using fetch and plain HTML.
3. MDBootstrap and jquery, using ajax.
4. React.js
5. Angular.js

Helper tools

- The Postman client for testing your API using a GUI while developing it
- HTTPie for testing your API using the terminal while developing it

Stretch goals

Devops

- Secure all API endpoints with TLS (sometimes referred to as SSL/TLS).
- Add monitoring using Prometheus to track the resource usage of your service
- Add proper authentication using OAuth2 and/or a third party authentication service such as Google Sign-in

Extended functionality

- I can register with my username and email and then log in.
- If I'm logged in when I make a purchase, I can see a list of my orders on my user page.

Deployment with Docker

There must be at least two dockerfiles - one for the database and one for your backend. You should use docker compose to integrate the two. For smaller groups and groups with lower ambitions using docker compose can be considered a stretch goal.

Task 2: A multiplayer game

The task is to implement the classic Snake, as a multiplayer game. Anything from a very simple game with a small screen and two players is allowed. You are allowed to implement another game instead, but ask first.

User stories

- I can start the program and I easily get to start playing alone even if no other players are connected.
- I can move the snake around the board using the keyboard to avoid obstacles (walls or players) or steer towards rewards (e.g. food - optionally dead players can turn into food).
    - The game tells me which keys to use so that I don't have to refer to the documentation.
- If I run into walls (if there are any) or other players it will kill me. If other players run into me it will kill them.
    - I clearly see an indication that I've died if I run into another player or a wall
    - If there are no outer walls in the game (e.g. like slither.io) will scroll with the direction of my snake when I move it.
- I can clearly see if other players connect (including bots if any), in a list of connected users. A minimum of 2 players must be supported.
    - There may be other players operated by the software (e.g. return of the bots - as snakes!) but this is not a requirement. If there are bots they behave like any other player and I won't be able to tell the difference (except maybe if they play really badly or superhumanly well)
- I can get the snake to grow by running into "food". The food can be just colored squares but I can see that they're different from walls if there are any.
- I get points for getting a bigger snake and I can see how many points I have.

Requirements and implementation options

- A minimum of 2 players must be supported
- You can implement more games than one if you want. You can also choose another game, but make sure to ask the teacher for advice if you do.

Technology choices

You can choose between the technology stacks listed below. Note that the stacks that are **recommended order** only recommended because we have more experience

1. **Python 3** with **tkinter** or **pyQt/QML** for graphics and **gRPC** for networking
2. **Node.js** backend with **html/css/javascript** for graphics and **socket.io** for networking
3. **Java** with **gRPC** for networking, graphics tbd.
4. **C++** with **gRPC** for networking and **Qt/QML** for graphics.

Other technologies can be allowed, but ask first and make sure to provide good arguments for how those choices make the skills you learned in this course shine.

Deployment with Docker

There must be a dockerfile for the server that allows you to start a game server using docker build / docker run commands. It should then be possible to deploy your game server in a public cloud making it available for players across the internet.

Stretch goals

- Secure all communication with TLS. Look at https://www.grpc.io/docs/guides/auth/#examples for examples in many languages.
- Add monitoring using Prometheus to track the resource usage of your game server. Document how the resource usage changes when many players are connected.
- Allow for many (>10) or unlimited players. This will require you to manage a large grid and you probably have to make the game board itself scroll around the snake than to make the snakes move around the game board.
- Add bots. This is a good way to test the scalability of your service by pushing it to the limit.
- Add a persistent high-score list with a database backend. Note that high-score lists can be fairly hard to keep from being hacked (I've tried!). Some intelligent notes on why that is and a possible solution would be an interesting read.