Exam assignment – PG3402 Microservices

The exam for PG3402 is a portfolio exam where you should demonstrate your mastery of Microservices and the techniques and tools relevant to them, as well as your understanding and reasoning about them.

The exam is a portfolio exam composed of 2 sections:

The project: The task is to develop the project idea that you previously had approved in the arbeidskrav document. The project will use microservice architecture to deliver the functionality mentioned in the arbeidskrav document, along with any other functionality that you think is interesting and relevant for the idea being proposed.

The reflections document: In addition to the practical project, you will be expected to have a short document (no more than 2 pages) that will contain your reflections during development.

The project

The project is the practical demonstration of your ability to implement the project idea you selected using microservices and related tools and technologies.

You will deliver an archive (**.zip**) that will contain **all** the code and relevant configuration and scripts to allow the examiner to **build** and **run** your project.

The project should contain a **README.md** document that contains the overview of the project as well as the steps necessary to build, run, and interact with your project. Commands to maven, docker, or other technologies used should be included. Double check that the steps you propose do, in fact, build and run the project as expected.

NOTE: It is essential that your project does, in fact, **build** and **run** as described in the README.md file. This is fundamental to delivering software in general, so make sure to double check that you have all the necessary components, and that the steps you describe are up to date with the version of the project you are submitting.

Individual reflections document

As discussed in class, microservices, the relevant tools and architecture choices, come with tradeoffs. The purpose of this document is to show what choices have been made in the development of the project, the reasoning behind those choices, and what the tradeoffs were.

For example, if a different tool or technology is used than the ones discussed in class, this choice could be motivated and discussed in the document.

The document should be no longer than 2 pages, and focus on the most interesting and relevant choices. The document should focus on issues specific to your project and your experience. This is not a theory section, just a discussion of what individual experiences were.

Grading

The project is assessed as a complete package, with microservice specific and project specific requirements all playing a part in determining the grade.

The evaluator **must** be able to build the project from the .**zip archive** and get all the services up and running. This is essential to grading the project.

The evaluator **must** be able to test all the functionality mentioned in the arbeidskrav and in the README.md file.

If any credentials are needed (for example, login credentials, database credentials, and so on) it is your responsibility to provide test credentials for the examiner to use.

If particular steps need to be taken in order to see certain functionality, this should also be described in the README.md file.

If some functionality can only be assessed by direct calls to the APIs, make sure to provide Postman calls to be sent, along with the expected responses from your system.

An examiner cannot assess functionality if that functionality is not accessible.

The **README.md** file should contain:
- [ ] **instructions** on how to **build**, **start**, and **run** the project
    - if you have more than one means of running the project (for example, docker containers run locally, docker containers running on dockerhub, or running the individual services locally), make sure you have clear instructions for each of these.

- [ ] an **overview** of the project (you can use the one from the arbeidskrav as a basis). The overview should accurately describe the project implementation.

- [ ] a **list of user stories** that allow an examiner to assess the functionality developed in the project (think of scenarios that the examiner can run to see what functionality you have implemented)

- [ ] a **diagram** showing the **architecture** of your system. This should show what services the project contains and what type of communication they have between them (synchronous or asynchronous).

- [ ] if you collaborated with other students on a project, the README.md file should also contain a discussion of the contribution and responsibilities of each team member.

During development, you may have to make a number of architecture decisions, assumptions about the domain you are working with, or simplifications to how such a project would work in reality. Document these decisions, assumptions, and simplifications in the README.md file.

Microservices specific requirements

Required (but **not sufficient**) for **E**
[ ] 1. Use multiple services, that fulfill different functionality and communicate with each other

Required (but **not sufficient**) for **D**
[ ] 2. At least two of the services communicate using **synchronous** communication (for example, direct REST calls between two services).

[ ] 3. At least two of the services communicate using **asynchronous** communication (for example, using Message Queue). This will be done in accordance with event-driven architecture, as discussed in class.

Required (but **not sufficient**) for **C**
[ ] 4. The project uses a unique access point, that handled calls and routes them to appropriate services – Gateway

[ ] 5. The project uses a unique access point that, in addition to routing calls, also does load balancing

Required (but **not sufficient**) for **B**
[ ] 6. The project has a means of centrally controlling the health of running services – health check

[ ] 7. The project has a means of centrally controlling configurations for the services – for example, using Consul

Required (but **not sufficient**) for **A**
[ ] 8. The project has a means of containerization – building container images from the existing services and getting such containers running and interacting with each other.

Task specific requirements

The functionality will be assessed based on the arbeidskrav, so the implementation should be for the same project. Changes are, of course, required, as the development process will provide additional information.

If the scope of the exam is reduced from the arbeidskrav (that is to say, if you choose to only implement some of the user stories) you should discuss the reason behind this reduction, and how you decided which user stories to remove.

You are, of course, free to add functionality in addition to what was in the arbeidskrav. That also should be discussed in the reflections document.

Note that the project should still have interesting and unique functionality, developed specifically for this exam. The implemented functionality should be consistent with 200 hours of work spent on the project (including software engineering activities around the project like requirement writing and prioritization, and other necessary activities).

The grading of task specific requirements will be based on the implementation of task specific user stories.

Some hints:
- implement the most important (highest priority) user stories first
- prioritize those user stories that are the most interesting for your project and the most important for the functionality that you intend to offer
- focus on user stories that involve functionality that is developed specifically for your project. Off the shelf functionality is not as interesting to work on and it is not as useful for you in showing your ability to work with microservices in particular, or to develop software in general.
- make sure to try out the process of building, deploying, and running your system before delivery
- make sure the project runs as expected, and that the evaluator can see all the functionality that you present in the README.md file
- make sure your delivery includes all the additional information needed to build and run your project: setup scripts, configuration files, seeding the database with data, docker-compose and anything else you might need.

Lykke til and Happy Programming!