Distributed System
Assignment 2 : Microservices

Fabian Denoodt, Benjamin Vandersmissen and José Oramas
{Fabian.Denoodt, Benjamin.Vandersmissen}@uantwerpen.be
University of Antwerp

2023-2024

1 Introduction

This document provides an overview of the requirements and background information for the second practical assignment of the Distributed Systems course. The goal of this assignment is to familiarize yourself with the microservice design pattern, decompose a problem in microservices and implement microservices using Docker/PodMan. You should solve the assignment **individually**.

2 Goal

The goal of this assignment is twofold:

- First, you should decompose a problem in microservices.
- Secondly, you should implement your design using the Docker/PodMan toolbox.

3 Requirements

3.1 Decomposing in Microservices

Given the following scenario, decompose in microservices in an intelligent way. This means grouping related functionality together in a single microservice and

making graceful failure possible. In doing so, think about how the services communicate with each other and what the impact is for the entire application when a single service goes down. You should also consider the scalability opportunities of your design. Your design decisions must be motivated in a report.

Scenario: Your friend has the bright new idea of merging Google Calendar and Facebook's event page together.

1. **Authentication:** A user can create a profile (Register) and sign in through username and password (Login).
2. **Events:** An event consists of a date, an organizer, a title, a description and is either public or private. Only the organizer can invite other users to his event. A user can see all the event invitations he has not yet responded to. Upon invite, he can respond by indicating whether he will participate, maybe participate or not participate in the event. The invite is then no longer visible. Given an event, all the users who are (maybe) participating should be displayed.

   In addition, if the event is public, anyone can participate without needing an invite from the organizer. This is achieved through a web-page displaying all future public events. Through this page, a user can indicate whether he will (maybe) participate or not.
3. **Calendar & Sharing:** Each user has a personal calendar, displaying the list of events the user will (maybe) participate in. Invites to which the user has not yet replied are not shown in his calendar. By default, a calendar is private, and thus, watching someone's calendar is not possible. However, if user X shares his calendar with another user Y, Y can now also see X's calendar. Sharing is not a symmetric operation (e.g.: X cannot necessarily see Y's calendar).
4. **Note:** There is no need to consider deleting or updating items. As such, when a calendar is shared, or an event is created, making changes should not be supported.

3.2 Implementing Microservices

Using the microservice decomposition you made in the previous section, implement the features using Docker images. Furthermore, you should implement the databases for the microservices you use in this part.

To make things easier for you, you can use separate Docker containers for the databases, however you still need to group databases together that are meant to be together.

For example, if you decomposed the problem in two microservices *microA* and *microB*, you can implement two additional Docker containers *microA-persistence* and *microB-persistence*, but they should only contain databases associated with *microA*, *microB* respectively.

Internal (from *microA* to *microA-db*) communications *with a database* can be done using python libraries (e.g. psycopg), while **external** (from *microA* to *microB*) communications between microservices need to be done via the REST protocol.

3.3 Frontend

You are given a code project which contains the GUI of the website. Use this project as a starting point for the assignment. As the frontend is readily implemented for you, you should only focus on the implementation of the backend logic. An overview of the web pages and their respective functionality which should be supported is presented at the end of this document.

4 Tools & Resources

The following tools and links might be helpful while implementing your solution.

- REST API in Python: implementing a REST API in python is actually pretty easy using a couple of libraries. The main libraries are Flask (https://flask.palletsprojects.com/en/2.0.x/quickstart/) and Flask-RESTful (https://flask-restful.readthedocs.io/en/latest/). You should already have some familiarity with Flask from Programming Project Databases.
- Docker: a tutorial that goes over the basic functionality of Docker can be found at https://docs.docker.com/language/python/build-images/. To run multiple microservices in a simple fashion, you can use docker-compose (https://docs.docker.com/compose/).
  In addition, the following resources are also recommended:
  1. https://www.youtube.com/watch?v=Gjnup-PuquQ
  2. https://www.youtube.com/watch?v=rIrNIzy6U_g
  3. https://www.youtube.com/watch?v=gAkwW2tuIqE
  4. An introductory presentation is provided on BlackBoard.
- Database: for the databases included in your microservices, you can use any database dialect you want as long as it works. However, most of

you will have experience with psql (https://www.postgresql.org/) from Programming Project Databases, so feel free to use that one. Interacting with a database from Python can be done with packages such as psycopg2 (https://psycopg.org) in the case of PostgreSQL.

5 Good to know

Here you will find some general tips that can save you some time and headaches.

- As the main focus of this project is on making you familiar with some tools and practices related to microservices, **you should not** focus on real-world problems out-of-scope. So **don't worry** about SQL injections, storing passwords securely, latency, load balancing and more.
- There are Docker images online that are pre-packaged with all kinds of tools. You can always take a look at https://hub.docker.com to see if you find anything useful.
- You can either use Docker or PodMan to run your solution, as both should work with the same files. However, PodMan has the advantage that it does not require root access, and as such your solution will always be tested using PodMan!

6 Requirements and Deliverables

The following are the minimum requirements and deliverables for a passing grade:

- You should include a way for us to easily run your solution with the inclusion of a `run.sh` that will automatically start your microservices and the user interface.
- You should also include a report. This needs to include the decomposed microservices, which features each microservice implements, which data it stores and which connections it has to other microservices. Provide an overview figure of the total microservice architecture. In addition, you should provide an explanation for the decomposition. (i.e., why do you group those features together in a microservice? what are the consequences if a service fails? why does your approach scale well for large user bases?)

The report also needs a small section for the implementation part of the assignment, where for each required Feature, you list the API endpoint that implemented it and document it, similar to Assignment 1.

- Make sure that your submitted solution is complete and has no missing files. Be sure that the Docker images you make will function on another computer.

7 Deadline and Submission

The deadline for this assignment is the **17th of May** at **23:59**. Late submissions will automatically have points deducted for tardiness.

Make sure that your submission has the following format and upload via Blackboard!

`DS-Assignment2-`*`Snumber-Lastname`*`.zip`

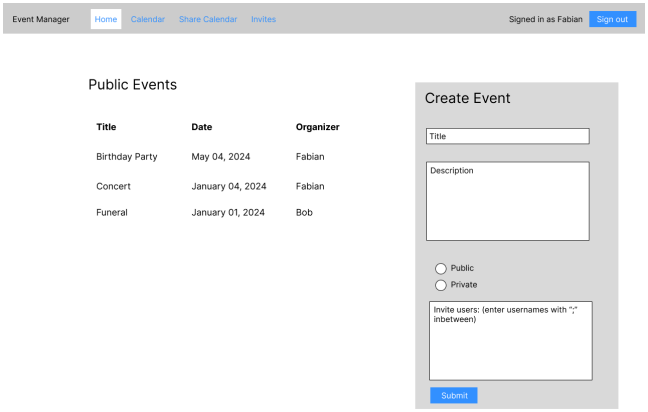

Figure 1: Home page displaying all public events (whether participating or not) and a region to create new events. Clicking on an event navigates to the event-details page, showing more information.

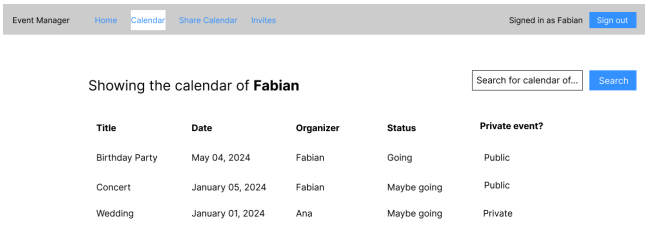

Figure 2: The calendar page. Navigating to this page by default shows the current user's calendar. However, by filling in another username, the calendar of that user can be seen. This should only work for users that have shared their calendar with you.

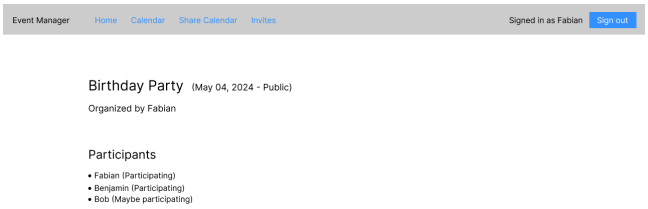

Figure 3: An example event-details page. This page can for instance be accessed by clicking on an event in the Home or Calendar page.

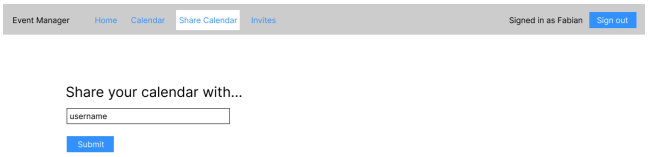

Figure 4: Sharing your personal calendar.

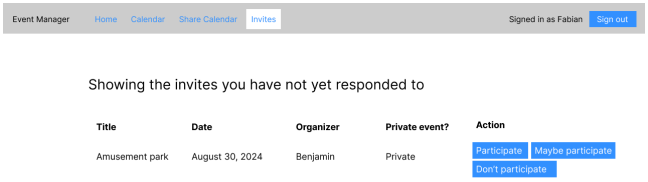

Figure 5: All invites are listed on the page. Upon a response, the invite should be removed.

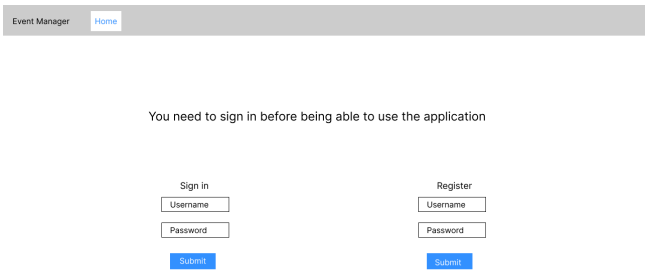

Figure 6: When a user is not yet signed in, he is shown this page where he can either register or sign in.