University of Vienna
Faculty of Computer Science
Research Group Software Architecture

Semester Project

Event Management System

2023S – 053020 VU – Advanced Software Engineering

May 24, 2023

- This document covers the assignment of the **Semester Project**.
- You can achieve a total of **47 points** for this project.
- The mid-term project submission (**DESIGN**) deadline is on **24.04.2023 at 11:00 AM**.
- The final project submission (**FINAL**) deadline is on **08.06.2023 at 11:00 AM**.
- Templates for the **DESIGN** and **FINAL** reports that outline the required report structure will be available in time on Moodle.
- This is a **team assignment**. You shall collaborate within your team to solve this assignment. All team members must contribute approximately equally to each of the design, the implementation and the documentation activities.
- In case you need assistance:
  - If you encounter problems related directly to this assignment document, please post in the **Moodle**[1] **Discussion Forum - Practical Assignment**, as it is probably also of interest to other colleagues.
  - For implementation/coding or technology-related questions, you can contact our tutors via ase.tutor@swa.univie.ac.at.
  - For specific questions regarding your solution or team issues, please make use of the **Gitlab Issue Tracker** in your team project and mention the handle of your team's supervisor (announced in Moodle).
  - You can discuss your inquiries regarding the practical part with the supervisors during the **online project question/answer sessions** in Moodle. To manage the sessions effectively, please write your questions in a **Gitlab Issue** and mention the handle of your team's supervisor **at the latest two hours before** the session.
  - As a **last resort**, you can contact the course email: ase@swa.univie.ac.at.

[1] https://moodle.univie.ac.at/course/view.php?id=390662

1 Git[Lab Submission] System

All files required by this assignment have to be submitted to the faculty GitLab system https://git01lab.cs.univie.ac.at/ into your GitLab Team Project (repository) located in the Gitlab subgroup: **VU Advanced Software Engineering / Students / 2023S**

Be aware to push your solution intended for submission and grading to the **master** branch. At the respective deadline the **master** branch is archived to a special branch for each of the DESIGN and FINAL submissions.

2 Plagiarism

If you copy code or other elements from sources other than the lecture slides, you have to provide a reference to them in a comment above the corresponding entry. All content submitted by students can be checked for plagiarism and/or cheating using automatic tools and individual inquiries. This can be done on concrete suspicion or on a random basis. Any (partial) performance obtained by fraud leads to an 'X' in your transcript, meaning that you have been caught cheating or plagiarizing. There are similar systems to this task on the Internet. It is not forbidden to be inspired by them; however, we run plagiarism checking tools. Any verbose copying of code, models or documentation counts as plagiarism.

3 Semester Project: Event Management System

The aim of this project is to methodically model and develop a system for Event Management, which allowes users depending on their role to organize or attend events and maintain the system. Main user roles are: **event organizer**, **attendee** and **administrator**. For reasons of data protection and privacy (see GDPR[2]) a user is not allowed to access any part of the system that is not directly pertinent to their role or *use case*. use cases that shall be supported by the system are:

- organizers can *create* events, specifying details e.g., date, time, and capacity.
- organizers can *manage* their *event inventory* for their events.
- attendees can give *feedback* for events.
- organizers and attendees can see *analytics* about the events (see SRs).
- attendees can export event to *calenders.*
- attendees can *bookmark* events that they are interested in attending.
- attendees can *receive notifications* about their upcoming events and changes to events they either *attend* or *bookmarked.*
- administrators *maintain* the services (i.e. the event management system) provides a mechanism to check the health of each service (e.g. the Health Check API pattern[3] or the HeartBeat pattern[4]).

[2]https://en.wikipedia.org/wiki/General_Data_Protection_Regulation
[3]https://microservices.io/patterns/observability/health-check-api.html
[4]https://martinfowler.com/articles/patterns-of-distributed-systems/heartbeat.html

3.1 Assignment

Your assignment is the **detailed modeling** and subsequent **implementation** of a system based on **your own exploration of the given domain**, and completing the **tasks** described in this section. You are advised to **start early** and take the time to carefully examine your domain, intentions, and design options at the outset, so that you avoid unnecessary effort when handling possibly unforeseen problems later on. Since this assignment is a team project, establishing good cooperation, arranging a suitable division of labour according to the particular skills of each team member, and making use of collaboration and development tools (e.g. issue tracking[5], CI/CD) is necessary. You will work **together** on the first submission (i.e. DESIGN) to provide the model of your application. Afterwards, each team member accepts a responsibility and works on his/her own tasks **separately** for FINAL. There are also smaller tasks in FINAL that need to be worked on together.

For DESIGN, the focus is on modelling the **domain model** and **architectural views** for the event management system and your services to be developed. For FINAL, we then expect you to **refine** this design using the *architecture patterns* and *microservices architectures* from Lectures 4-7. Thus for DESIGN, we assume you have a basic understanding of *Web Services*, *Service-oriented Architectures*, and *microservice basics*. If you lack this knowledge, the first part of Lecture 6 discusses them from a microservice perspective. The following Web tutorials give an overview of the basics:

- `https://www.ibm.com/cloud/learn/soa`
- `https://microservices.io`
- `https://martinfowler.com/articles/microservices.html`
- `https://microservice-api-patterns.org/introduction`

In addition, it is important to get hands-on experiences with programming of services. If you haven't done so before, we suggest to realize a few simple services each with their own database that communicate with each other, before working on DESIGN or FINAL. Please study online tutorials and examples for the set of technologies you intend to use in your FINAL submission. The language of the semester project is **Java and related technologies**. We recommend the Spring Framework[6]. If you would like to use another langauge, it must be approved by your supervisor.

Task 1: Domain Model and Architectural Views (14 Points)

Construct a **4+1 Views Model** of the event management system that consists of appropriate UML diagrams and text **describing all 5 views** in suitable detail. From the given domain introduction and use cases, identify functional and non-functional requirements, prioritize them, and iteratively refine your **4+1 Views Model** accordingly, while you explore the domain **realistically** and **in-depth** (e.g. by Internet search and possibly contacting domain experts). Decide on the boundary of your system (i.e. make realistic assumptions about the functionalities your system provides), and the functionalities your system depends on externally or as mocked.

[5] `https://docs.gitlab.com/ee/user/project/issues/`
[6] `https://spring.io/`

Exercise your judgment as to how the system is best organized, but at least in the domain layer employ **Domain-Driven Design** and its **building blocks** (entity, value object, service, aggregate, factory, repository) explicitly (i.e. use UML features such as comments, custom stereotypes etc.) to document the applied building blocks in your UML diagrams. The UML class diagram (logical view) of your domain layer must

- consist of at least 10 classes and additionally 2 classes per team member, and
- every DDD building block must be used at least once.

If your domain layer does not meet this requirement, identify more functional requirements, explore the domain more deeply, and/or consider additional use cases in agreement with your supervisor. Design your system as a **modular monolith** and have in mind that the design will be refined and implemented in accordance with the **microservices architecture** later. Design the **bounded contexts** according to the Scope Requirements (SRs) of the system (see Task 2 description and consider the SRs when designing the system).

Afterwards, as a first step to demonstrate the supported use cases and enable everyone in Task 2 to work at their own pace, implement appropriate interfaces and/or simulated mockup functionality:

- For each main module to be realized in Task 2, implement a distributed communication mockup. Wikipedia defines a mockup in this context as: "Service virtualization and API mocks and simulators are examples of implementations of mockups or so called over-the-wire test doubles in software systems that are modelling dependent components or microservices in SOA environments."[7] Please note that there are many tools for creating mockup services. You can either create them by programming them yourself or using an existing mockup tool (such as SOAPUI, Mockable, mockAPI, Mocky, and many more).

For example, if based on REST[8] and HTTP[9] client/server communication, the email notification subsystem mockup would deliver a few predefined emails. The results of this task will be delivered in the DESIGN submission of the project.

Task 2: Microservices Implementation (16 Points)

Realize the implementation of the system in accordance with the **microservices architecture** that meets the following system Scope Requirements (SRs).

SR1$_{\textbf{joint}}$ All students are responsible for updating the design of the system frequently to reflect well the services realized by the individual students, as well as their interaction. In addition to the per-student tasks, the system should provide an API Gateway, as well as integration or end-to-end tests (testing the integrated functionality of the system). You are together responsible that the API Gateway works and that they can be used to demonstrate the major functionality of the system. Also you are together responsible that the integration or end-to-end tests test the integrated functionality of the system.

SR2$_{\textbf{joint}}$ The system (i.e. all services) must support event-driven communication (e.g. via the publish/subscribe pattern). You can use existing technologies for this requirement.

[7] https://en.wikipedia.org/wiki/Mockup
[8] https://en.wikipedia.org/wiki/Representational_state_transfer
[9] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

SR3 The system should support a login service, where role-based authentication is managed. Users log in to the system and receive an access token[10]. You can provide your own access token implementation or use an OAuth2 service[11]. The system should support a maintenance service (see this use case on page 2).

SR4 The system must support an event inventory service: organizers manage their events via their inventory. They can update all the event-related information, at least *type, capacity, date, description.*

SR5 The system should support a search service: organizers and attendees can search for events by different criteria.

SR6 The system must support a bookmarking and tagging service: attendees can mark/unmark events they are interested in. The attendees can tag/untag different events, e.g., sports, food, education.

SR7 The system must support a recommender service: attendees get notified about new events based on their bookmarks **and** other reasonable criteria, e.g., *type, capacity, date, description.*

SR8 The system should support a feedback service: attendees can leave feedback and rate the events they participated in. There should be different criteria for feedback, e.g., suitable location or accuracy of description.

SR9 The system should support a calender export service: there should be at least three different formats, e.g., JSON, XML, and one standard calender format, for exporting bookmarked or attending events. Event tags should be considered (see SR6).

SR10 The system should support an analytic and reporting service: organizers can generate reports for events, e.g., how many people registered or attended, the number of bookmarks. The attendees should see analytics about feedback.

SR11 The system should support an attendance service: The organizers can specify the maximum number of attendance. The attendees can attend an event if there are vacancies. The organizers can send message to attendees.

SR12 The system must support a notification service: attendees are informed about any changes in events they are attending or bookmarked.

In addition to the $\text{SR1}_{\text{joint}}$ and $\text{SR2}_{\text{joint}}$ tasks, which are to be implemented **jointly by the team**, **each student** is responsible for **two SRs**. If teammates leave the project during the semester, you can realize their scope requirements as mockups only (or use the mockups provided during DESIGN). Please inform your group supervisor about such a decision. **$\text{SR1}_{\text{joint}}$ and $\text{SR2}_{\text{joint}}$ cannot be replaced by mockups only**. Please note **any data should be persistently stored using CRUD operations[12].**

To allow for user interaction with the system implement a **simple** user interface with emphasis on the functionality instead of eye-candy (e.g. a simple web front-end). The results of this task will be delivered in the FINAL submission of the project.

[10]https://microservices.io/patterns/security/access-token.html
[11]https://oauth.net/2/
[12]https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

Task 3: Containerization, Deployment and Testing (17 Points)

Each microservice needs to be containerized using Docker technology[13], deployed on a Continous Integration / Continous Deployment (CI/CD) pipeline using GitLab[14], and tested seperately at different levels (unit, integration, end-to-end). **Each student is responsible to do this task seperately for the microservices they implement.** Additionally, all students of the team are **together responsible** for deploying the API Gateway and the overall system integration or end-to-end tests. For this joint responsibility, you can reuse the pipeline realized by one of your teammates for the individual services. The results of this task will be delivered in the FINAL submission of the project.

3.2 Submission and Grading Criteria

Only material that has been submitted **in time** (i.e. pushed to your GitLab Team Project in the **master** branch following the **exact naming**) will be taken into consideration.

For grading we rely in large parts on your provided documentation. Hence, we suggest to ensure that all documentation describes your system design and implementation concisely and includes the requested information. For the per-student tasks, each student is responsible for providing the documentation of their respective parts.

3.3 Mid-Term Project Submission (DESIGN)

Deadline 24.04.2023 at 11:00 AM

The DESIGN submission shall include the following artifacts in the root directory:

`model` A directory containing a **set of UML diagrams** showing the **4+1 Views Model** of the system that you have planned with additionally full Use Case Description templates (aka. tables) according to the given **system Scope Requirements (SRs)**. Focus on the **domain layer**, but also provide initial modeling details about the other layers of your developed system. Design your system as a **modular monolith** and define the **bounded contexts**.

All UML diagrams must be **semantically** and **syntactically** correct. Use a drawing application (e.g. draw.io) to draw the UML diagrams. Incorrect, unreadable, or hand-drawn diagrams will not be graded! Either a high-resolution bitmap-type or a vector-type (e.g. `.svg`) export is required.

`report_design.pdf` A single **PDF** file report (with the exact name) on your current project status, with the following **compulsory** contents:

- Documentation and explanation of **design approach**, **assumptions**, **thought process**, and the resulting **design decisions** made while working through the assignment. Include a description of the planned **development stack** (e.g. Maven build, GitLab CI/CD) and **technology stack** (e.g. Spring Boot, Eclipse Paho).

[13] `https://www.docker.com`
[14] `https://docs.gitlab.com/ee/ci/`

- Identified **requirements** of the system, their implications on the implementation (e.g. limitations, constraints), and how you intend to verify the requirements (e.g. test cases, manual inspection).
- The **4+1 Views Model** incl. all UML diagrams with the appropriate description; everything strictly synchronized with the contents of the `model` directory!
- A planned distribution of tasks among team-members, which, in case of approval by your supervisor, will be worked on **individually** for FINAL. Please indicate in your DESIGN deliverable which SRs will be realized by **each student** for FINAL.
- **Team contribution** that contains a **listing of contributions** and **time effort** for **each team member**.
- A **HowTo** documenting the functionality of each mockup (e.g. its expected output, and how to run the mockups).

`implementation` A directory containing your implementation of the mockup, skeleton-like of the system. **Do not** bundle your code in archives like `.zip`, `.7z`, `.rar` etc.

- Students should always be able to go to mockups of the other services implemented by their teammates and not be reliant on each other after DESIGN.
- **For each subsystem** you should implemet a mockup functionality, as well as define and provide all HTTP endpoints.
- **Each student** must implement the mockup of the subsystem, for which they will be responsible in the FINAL submission.

The DESIGN submission will be graded with up to **14 points**. You have to submit **at least** a complete **4+1 Views Model** (set of UML diagrams) to be graded! We will use the mockup implementation by each student, as well as the individual performances during PRES (see Section 3.5 Examination), to differentiate the DESIGN grade between students.

3.4 Final Project Submission (FINAL)

Deadline 08.06.2023 at 11:00 AM

The FINAL submission shall include the following artifacts in the root directory:

`model` A directory containing refined UML diagrams according to the **microservices architecture**.

`report_final.pdf` A single **PDF** file report on your **final** project status with the **updated** content from the `report_design.pdf` file and the following **compulsory** contents:

- Updated system design (i.e. UML diagrams with appropriate additional textual descriptions; everything strictly synchronized with the contents of the `model` directory).
- Document your two most involved **architectural design decisions** regarding the overall developed system according to the **minimal decision template** presented in the lecture.

- Updated **requirements** of the system, their implications on the implementation (e.g. limitations, constraints), and how you verified the requirements (e.g. test cases, manual inspection). Include the status of the verification: is the implementation completely meeting the requirements?
- A **changelog** summarizing what has changed compared to DESIGN, including how the recommendations from the DESIGN feedback have been implemented.
- Updated **Team contribution** that contains a **listing of contributions** and **time effort** for **each team member**.
- A **HowTo** documenting how the implemented system is to be installed, initialized, launched, and tested. Please provide instructions to easily run your Dockerized application (e.g. using **Docker Compose** commands). **Each team-member** must include the documentation of Task 3 separately for each microservice (i.e. containerization, CI/CD deployment pipeline and testing).

`implementation` A directory containing your implementation (including test cases etc.). **Do not** bundle your code in archives like `.zip`, `.7z`, `.rar` etc. The implementation for **FINAL** shall adhere to **all** of the following:

- Well organized implementation (i.e. one directory per each microservice).
- Use of current development methods and tools (e.g. use of build tool like Maven or Gradle, good coding practices incl. common code style, unit tests, Javadoc documentation, object-oriented programming) to construct high quality software.
- Provide the containerized microservice **Docker images**, as well as the configuration files (e.g. Dockerfile, docker-compose.yml etc.) to easily run the application following the **HowTo** in `report_final.pdf`.

`video` A folder containing either a video file, or a text file with a link to the video. We recommend that you upload the video to a common video platform (e.g. YouTube, and provide a corresponding link in the report and the repository).

- Run the application (e.g. using `docker-compose up` and showcase the use cases).
- For each microservice apply the **maintainer use case** (i.e. deploy it on a CI/CD pipeline - showing the stages - and showcase that each microservice works as expected by checking its health or using the HeartBeat pattern - see the use case on page 2).

The final submission will be graded with up to **33**. If you achieve **less than half** of the points on a practical part, the related documentation part is graded with **0 points**.

Up to **12 points** for the implementation of the Event Manager system and up to **4 points** for the related documentation part.

Up to **6 points** for the containerization and deployment of the microservices and up to **3 points** for the related documentation part.

Up to **6 points** for the testing of the system (unit, integration, end-to-end) and up to **2 points** for the related documentation part (individual and team work, see Task 3).

3.5 Examination

- Each team has to present their solution at the end of the semester during the **Presentation Talk** slots (**PRES**). The assignment of teams to each slot will be announced in due course.
- The Presentation Talk is **essential** for the final grading. The grading of the final submissions (i.e. DESIGN and FINAL) are **preliminary** and can be revised upwards or downwards depending on the team and individual performance in the Presentation Talk, as well as the individual contributions to the project.
- In addition to their own specific tasks, each team member should have knowledge of the **joint parts** and be able to explain the team's design process, decisions, and the rationale behind them.
- **No** elaborate presentation material (PowerPoint slides etc.) is required apart from the material you submitted as your solution.