Dentistimo

Assignment for DIT355 2021

Mini Project: Distributed Systems

This document contains the main assignment for the course, which has to be cleared in a team-effort. The project requirements are going to **change over time** in order to exercise the team's **agility** addressing the learning outcomes of working in teams, planning, and taking responsibility. Note though that adjustments will only be required when **explicitly noticed** by TAs. They will apply to all Teams equally, and will be communicated well in advance.

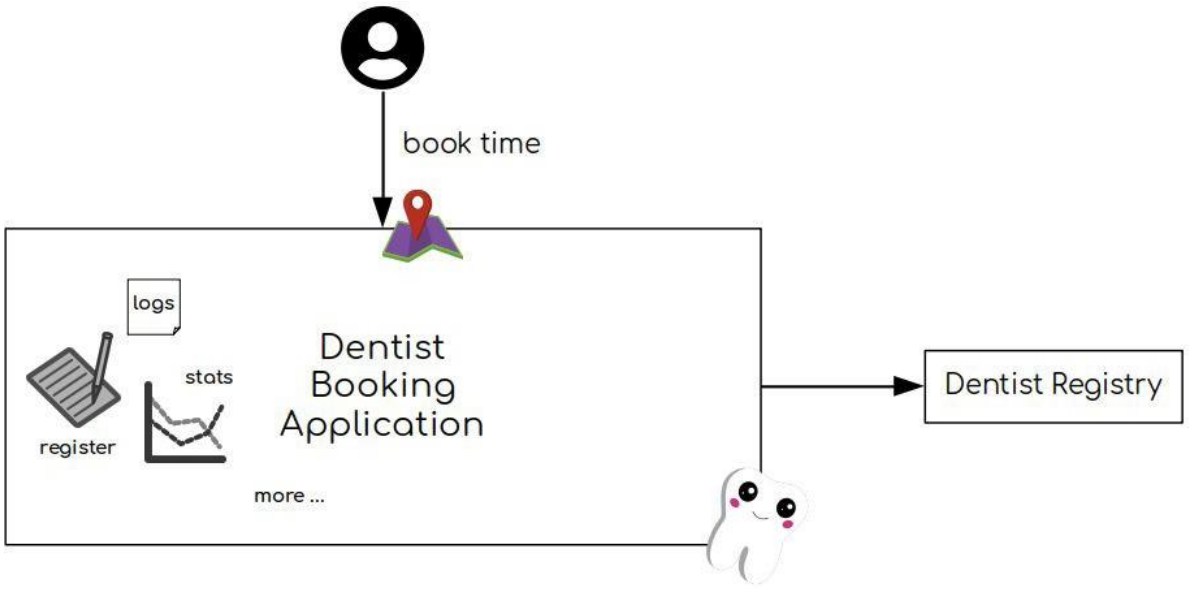
Throughout the course, knowledge about tools and methods will be communicated to extend the teams toolbox for addressing challenges of distribution. The most notable hurdles will have to be described, and solutions shall be explained. **Demonstrating the conscious choices in a distributed architecture for a realistic project is an important part of this course.** Try to strike a good balance between creating a meaningful software architecture and investigating new technology. And have fun.

Introduction

Our teeth are one of our most precious tools. We often don't appreciate them enough until we lose them, for instance due to bad dental care. Doing annual checks with a dentist is therefore recommended. For someone moving to Sweden, or within Sweden, it is not always easy to get a dentist appointment as many practices are working on full capacity already and will reject you as a new care-taker. As a consequence, as of today, you are required to do a manual search online, and additional time-taking calls with little chance of success.

The Task

You will create a service that allows residents of Gothenburg to book dentist appointments. Through a graphical user interface, a user shall be able to find available times in user-specified time-windows. Your solution will be based on a distributed system that combines various architectural styles. You will keep track of the availability of free time-slots for a number of fictitious dentists (Dentist Repository, found here) which you graphically signal to the user. A user is able to book appointments and receives a confirmation/rejection through the system. Find a conceptual overview of the system below.



The task exercises the learning outcomes of the course, i.e. to develop a product based on a distributed system in a team using common collaboration practices while applying knowledge about software architecture.

See the creation of the system as your main task, with the practical application as a driver forward and a means to demonstrate your acquired knowledge. However, it should be mentioned that this task came about in discussion with real dentists, explaining a real need of such a system in Sweden. Thus, if a Team is creating exceptional work and is willing to continue the project, there are good chances of taking this even further, for instance in collaboration with the Innovation Office at Gothenburg University. Feel free to inform the lecturers if this sounds interesting.

Requirements

General

- A middleware based on the MQ Telemetry Transport (MQTT) protocol must be used.
- It must be clear that distributed communication takes place.
- Components communicating via middleware have no knowledge about the physical location of other components, and don't depend on other components residing at a specific physical location. (Distribution Transparency)
- All members of the team shall understand all components; this is particularly important in the demonstrations and assessments.
- The toleration of failures will play a role in the final hand-in, and what exactly will be required will be part of the requirement updates later during the course. To be considered from the get-go:

    - All components must be capable of appropriately handling standard failures, such as wrongly formatted data inputs or out of bounds inputs for the defined interfaces.
    - Resource-handling shall be mindful, e.g. stopped components must unsubscribe from the MQTT broker fulfilling the contract. See L-4.

Input

The dentist registry database contains all dentist related information needed to handle bookings of the service. For simplicity, we assume that appointments are each 30 minutes long, starting every half or full hour, requiring one dentist. A dentist has a lunch break of one hour a day, and one Swedish Fika break of 30 minutes. Booking requests and responses are to be handled through MQTT. A booking request follows the format from this example, and a response from the system is exemplified here.

Visualization

- A user interface shall
    - contain a map-view over Gothenburg that can be navigated.
    - visualize the supply/available slots for appointments (individual or grouped availability using e.g. color coding or symbols, possibly changing at different zoom levels...).
    - allow for user requests.
    - delegate incoming user requests.
    - react to responses with appropriate messaging to the user.
    - react to simultaneous bookings by making changes in availability visible to the user (preferably without requiring an active refreshing of the interface by the user).

Architectural Constraints

The distributed system must consist of **at least 4 distributed components**, each clearly contributing to the purpose of the application and leading to an overall meaningful architecture. Architectural styles such as Pipe-and-Filter, Publish/subscribe and Client/server are supposed to be **combined**.

Software Distribution

The task shall be solved through a Distributed System architecture. The Distributed System should execute using a collection of nodes, so that each process is mapped onto a node. Nodes can be either separate hardware nodes, or separate operating system processes. During the **final demo**, the system must be demonstrated to run on **clearly independent processes**.[1]

[1] e.g. by starting the programs from different terminal windows on the same machin

Technology

To start with, a broker (centralized P/S component) must be set up and components for the emulation of requests as well as their visualization shall be developed to form the distributed system. **The majority of components** shall communicate through messages delegated by the broker (C/S is allowed if it can be meaningfully integrated into the architecture).

The solution is expected to use the Message Queue Telemetry Transport (MQTT) protocol as a common broker for **most inter-componen**t communication. The Eclipse foundation offers open-source implementations for the broker itself, called Eclipse Mosquitto, as well as client integration libraries for many high-level languages, going under the name Eclipse Paho.[2] However, these are merely options, and the choice of language and tools is entirely up to the team.

As this year the course is a distance course, we align the requirements of distribution in such a way that each student must be able to establish a MQTT broker on their local machines, and run all or most of the system components locally in different processes. It may therefore be **preferable to use a web-based front-end and not a mobile based front-end** as a user-interface, which otherwise requires all team-members to be able to run, say, an Android emulator locally (**make sure to agree on this** early in the Team). We discourage you from creating multiple UI's, simply because of the time limitations of the course, and the course goals being challenging as they are.

Collaboration and Submission Guidelines

The team is responsible to give the teachers access to all project resources at all times. The process documentation shall be created using Trello, following the Kanban style. Teams will be assigned a Team number (X), and the Trello board of the course has to follow the following naming scheme when inviting the other teachers:

"DIT-355 2021 Team X"

The documentation (see PMR below) includes the decisions for roles within the team and how members change roles over time. Role changes are encouraged to leverage learning through seeing the challenges from different angles, but it is up to the team.

All code shall be made available through the Chalmers|GU GitLab server , which shall be the main integration platform for the collaboration regarding source code throughout the course, also for the purpose of traceability.

Each independent component shall have it's own project/repository. An updated version of a high level description of the code shall be available in *all* of the teams repositories Readme(*.txt* or *.md*) files.

[2] MQTT is a standard, so if you want to substitute the broker or client by any other implementation/library, that is fine. Mosca, for instance, is another implementation of a MQTT broker.

No code/documentation is to be submitted through Canvas, instead, **all projects on GitLab** are expected to be **tagged with versions "v1", "v2", "v3", and "final"** no later than the respective deadlines for the four milestones you find on Canvas. The code is expected to reflect the software architecture and function according to the documentation[3]. Every second week, in accordance with the dates for version tagging above, retrospective reports have to be submitted through Canvas. This will add up to a total of four retrospective reports, two on the team level, and two on the individual level. The questions to be addressed can be found on Canvas in the submission folder (they might differ slightly from submission to submission). Final demonstrations **with all team members required online[4]** are held in January 5th.

Each Team has an assigned teaching assistant (TA), who they meet for roughly 30 minutes, including a 15 minutes Checkpoint meeting which the teaching assistants both try to support and assess[5]. The TA is your first point of contact regarding guidance in the project. An agile working process (Scrum) is expected to be followed, with progress being reported in a Kanban style using Trello. The further the course progresses, the more stringent the expectation on the application of agile collaboration and familiarity with the agile jargon.

To the best of the team's abilities, source code shall be documented, where not self-explanatory. Comments that **explain the why behind the code** are often more valuablethan comments that focus on the what *(*which can be derived from the code itself*).*

Passing (G)

The passing requirements for the course are separated into two categories, Team and individual, as presented and explained in the course kickoff meeting and slides. In short, the Team requirement entails the completion of this assignment. Software and Documentation must be in good sync, in particular for final submission. The individual requirements are

- active participation in all weekly checkpoint-meetings.
- participating in the presentation for milestones
- Traceable contribution in all assessment: Process, Software, Documentation.
- for passing with distinction (VG), read below.

[3] A third party should be able to run the distributed system following the instructions.
[4] Get in touch with the lecturer *before* the meeting in the exceptional case of non-availability.
[5] In particular in the beginning you will get help from the teaching assistant regarding scope.

Passing with Distinction (VG)

The course plan dictates the following:

*"To be awarded Pass with Distinction (VG), students must in addition **fulfill their role with excellence**, manifesting this by **choosing suitable tools and/or techniques**, and **supporting this with good documentation**."*

This is an individual criterion which will be assessed on an individual basis. You as an individual are therefore responsible yourself to keep track of your **individual contributions** in terms of tools and techniques introduced and realized throughout the project. In the final individual retrospective report you will therefore justify those contributions in a free-text form.

Good overall documentation of the software is a Team-requirement. The *good documentatio*n reference here means that your individual contributions must be **well justified** and **traceable** in your **Team's process documentation** to ensure that there is a team consensus about this really being your contribution. In your reflection report, make sure to focus on **explaining the impact that specific contributions had on the team's success**. Team-work is vital for the project to succeed, but justifications such as "X and I did all the work on suggesting/introducing and implementing tool/technique Y together" do not help the teachers making the individual assessment.

One example of reporting your key contributions is to **concisely** (i) describe **what** contribution you made, and (ii) provide links to traceable **artefacts** in your trello board (e.g., card discussions, mentions) and/or GitLab (e.g., commit messages). The impressions obtained by teachers and expressed by teammates in the retrospectives will also weigh into the decision.

Examination

The course applies **continuous examination**, with milestones for demonstrations that can be found in the course syllabus on Canvas. There will be final assessments in the examination week, but bear in mind that the bulk of achievement is expected to be **delivered throughout the course** before that, and known by the examiner/course responsible at that point in time. This means that the final examinations are merely a confirmation of the structured assessment that took place by the teacher team before that, based on the rich team data available on Trello, GitLab and the retrospective reports.

Distributed System Documentation Repository

The overall system documentation shall be stored in a single git repository (GitLab project assigned to you by teacher) with name **Documentation**. The documentation shall live up to the same traceability as the source code (through Gitlab) and the process (through Trello). All documentation shall be assembled in the Readme*.md* of the repository root using the

Markdown style, including all diagrams.[6] Over time, architectural diagrams shall be created and updated in accordance with the code representing the actual code well (mapping of code and documentation).[7] Business case and/or cost/revenue of the proposed application, which was recommended in your previous course on architecture, are not part of this course. Technical depth is not to be reported on either. Instead, **pay attention to architecture drivers/forces** because they are essential in shaping (justifying) your architectural design.

We understand that final Video presentation, Trello and GitLab will provide all details on your project. Therefore, the final written report on your project should be very brief and we recommend including:

1- Program Management brief
   - Describe the project management practices used
   - Report on important project management decisions regarding schedule and scope

2- Strategy & design decision
   - Provide your overall strategy to manage and execute the project
   - Identify your Architectural Significant Requirement (ASR) that impacted your design decisions

3- Architecture, styles, and tactics
   - Include a clear description of the conceptual design of the architecture, including architectural styles
   - Include a section that explains how the conceptual design is mapped onto implementation/technologies.

4- Lessons learned & Conclusion
   - Identify most important learning experience for your team in this project
   - Identify most significant challenges and how you mitigate these challenges
   - For a project similar to this in the future, what would be the things that you wish available in order to deliver best results.