Final Submission

1. User stories

Title	User story	Acceptance criteria
Filter time Slots	As a patient, I want to search time slots for a specific time period, so that I can book time slots more conveniently.	● Provide start time and end time filter box for users to select. ● If the user makes a filter selection, only clinics, dentists and time slots that meet the criteria are displayed. ● Filter the retrieve data by using pipeline architecture.
Authenticate dentist requests	As a dentist, I want to use services without the chance of being impersonated by someone else.	● Once logged in, the dentist shall not be required to re-enter login details for further requests.
Cancellation of booked time slots	As a patient, I want to cancel the time slot of my appointment with my dentist so that I can reschedule the appointment based on my updated schedule.	● If a logged in patient has booked an appointment for the selected time slot, the UI should display a Cancel Booking button so that the patient can cancel his/her appointment from the UI. ● When the patient clicks the Cancel Booking button, the client publishes an mqtt unbook topic to which the Booking Service subscribes and updates the status of the booking for the specific time period. ● Cancelled slots will be turned green for patients to rebook.
Improve real time response	As a user, I want to view the page that responds in real time so that I can see the updated information without refreshing the page.	● React to simultaneous bookings by making changes in availability visible to the user (without requiring an active refreshing of the interface by the user). ● Timeslots and their associated status, colors, and buttons are updated in real-time.

2. Key features and tasks

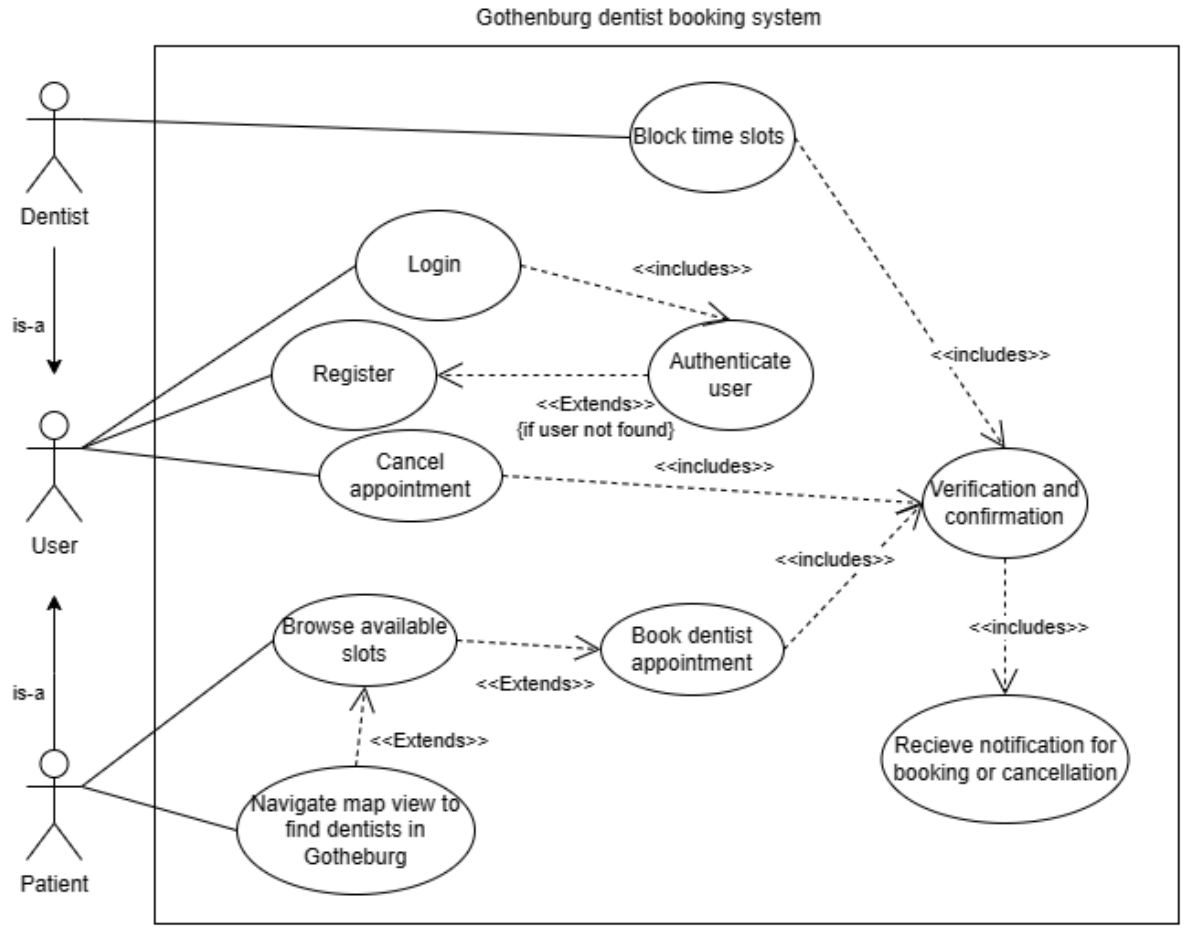
- Filter search function:
    - Provide start time and end time filter boxes for users to select.
    - Filter time slots, dentists, and clinics by selecting time periods.
- Authenticate dentist:
    - Ensure secure login for dentists.
    - Implement a session management system to avoid repeated login for subsequent requests.
- Cancellation of booked time slots:
    - Display a Cancel Booking button in the UI for logged-in patients who have booked an appointment.
    - When the Cancel Booking button is clicked, publish an MQTT unbook topic.
    - Update the booking status for the specific time period in the Booking Service.
    - Change the color of cancelled slots to green for patients to rebook.
- Improve real time response
    - Implement real-time updates for the page without requiring manual refresh.
    - Update time slots and their associated status, colors, and buttons in real-time.

- Implement monitor tool
    - Using CLI to display details about the user service load.
    - Using CLI to display details about the booking service load.

3. Architecture

4+1 View model:

Use-case diagram:

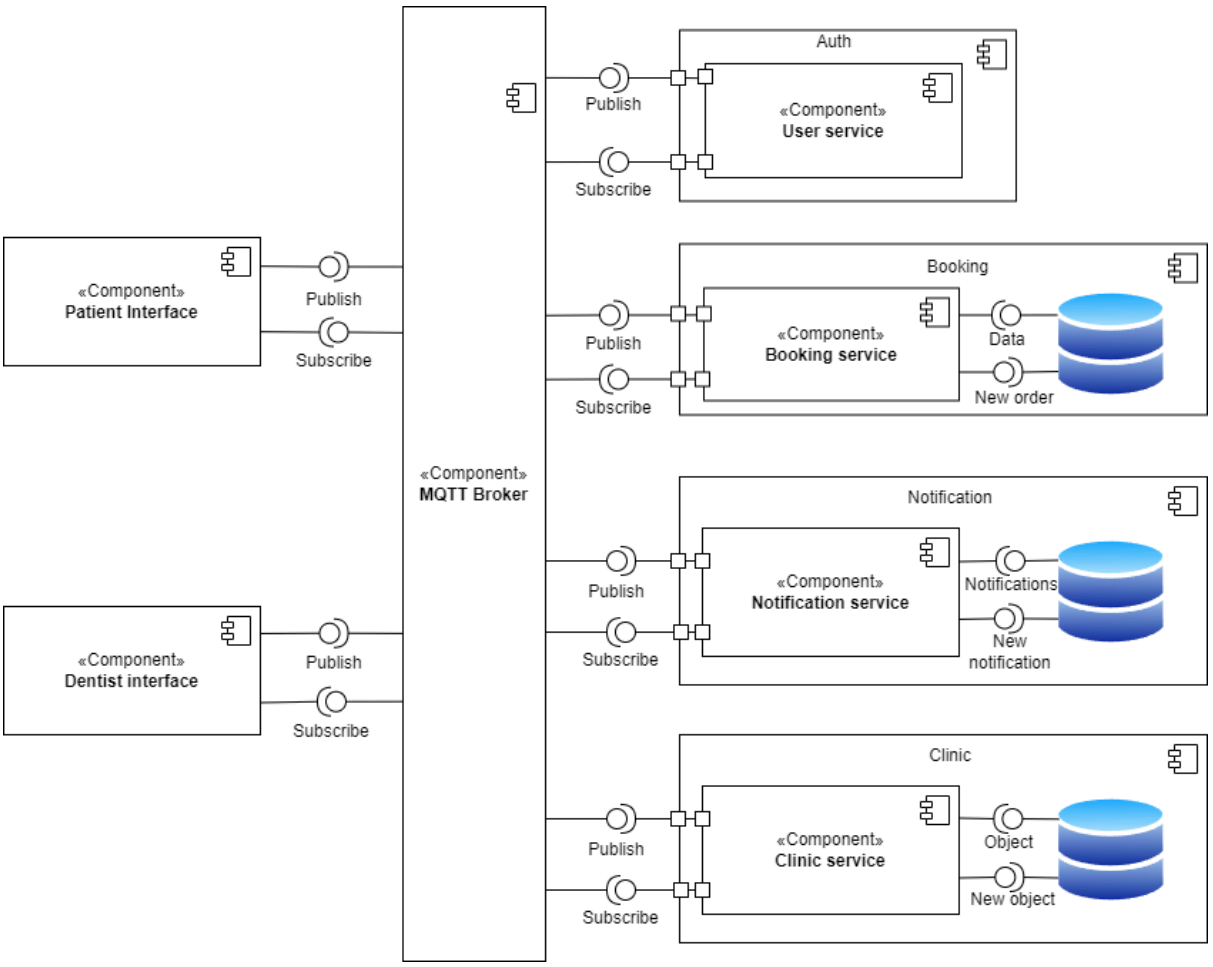


https://drive.google.com/file/d/1QdV1vNX04qkpw-O42jC6JpIMYzj4IeFO/view?usp=sharing

Description:

(UNCHANGED) A use case diagram depicts the actors, the use cases and the relationships between them. The system includes users, dentists, and an administrator. Users can perform login, registration, find dentists in the system, browse available dentists, schedule or cancel appointments, get a navigation map to find dentists, get a notification for scheduling or cancellation, and find dentists in the system. Dentists can handle login, logout, get available slots, cancel appointments, accept or reject appointments, get notifications, and get slots from the administrator. The administrator can handle block time, authenticate users, cancel users if not found, accept or reject users, and get available slots.

Component diagram:

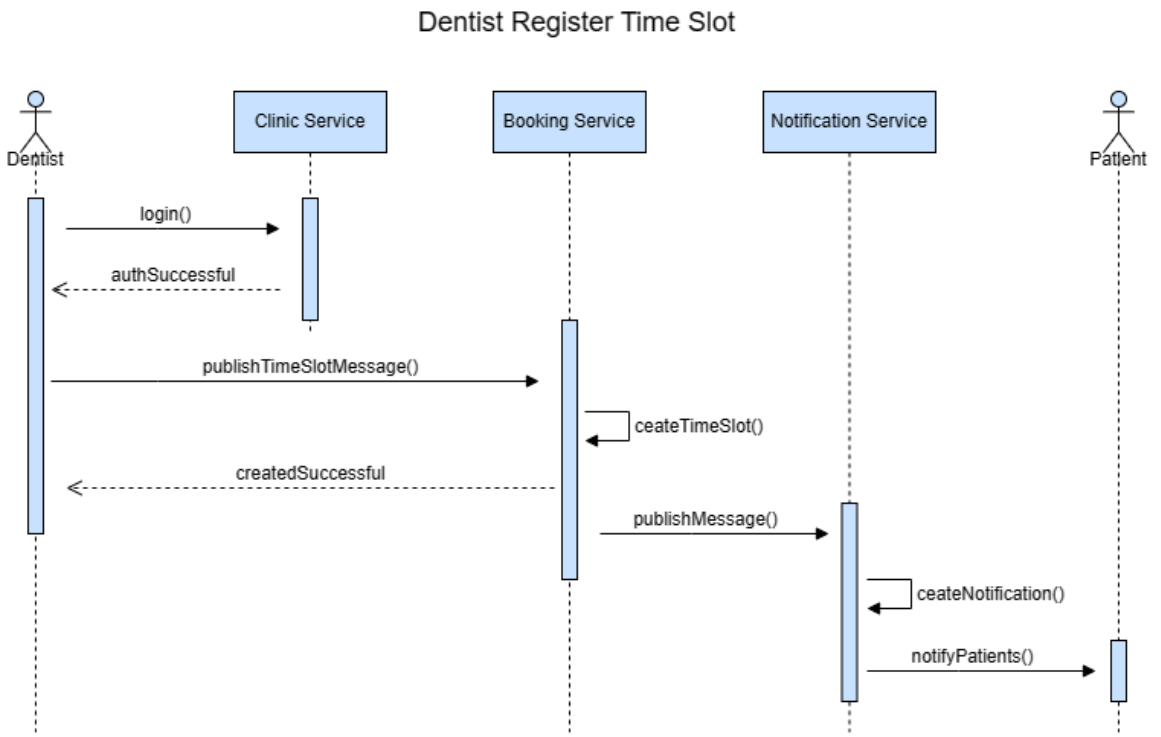


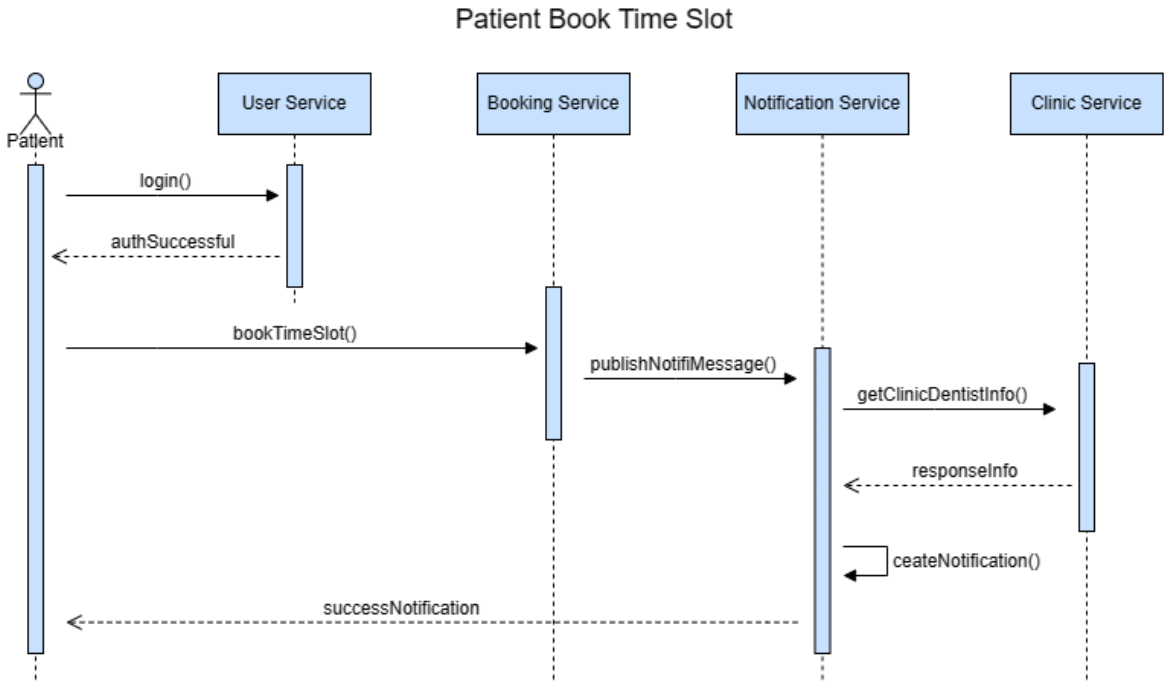
https://drive.google.com/file/d/1nWVawNji-KZ_ncC8Uuzcllu0LJmymyjb/view?usp=sharing

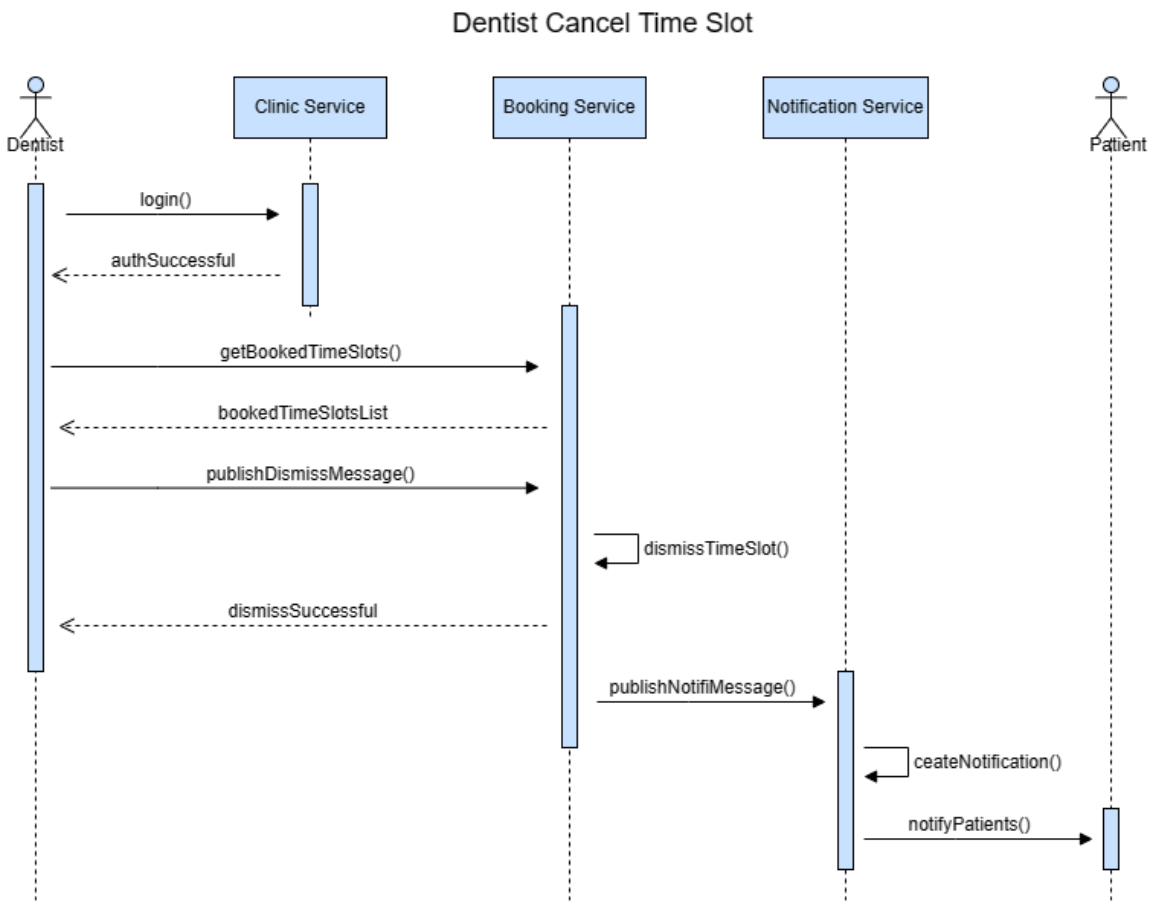
Description:

Changes include the removal of the user service database as it is not required as there is nothing to store due to login through BankID.

Sequence diagram:





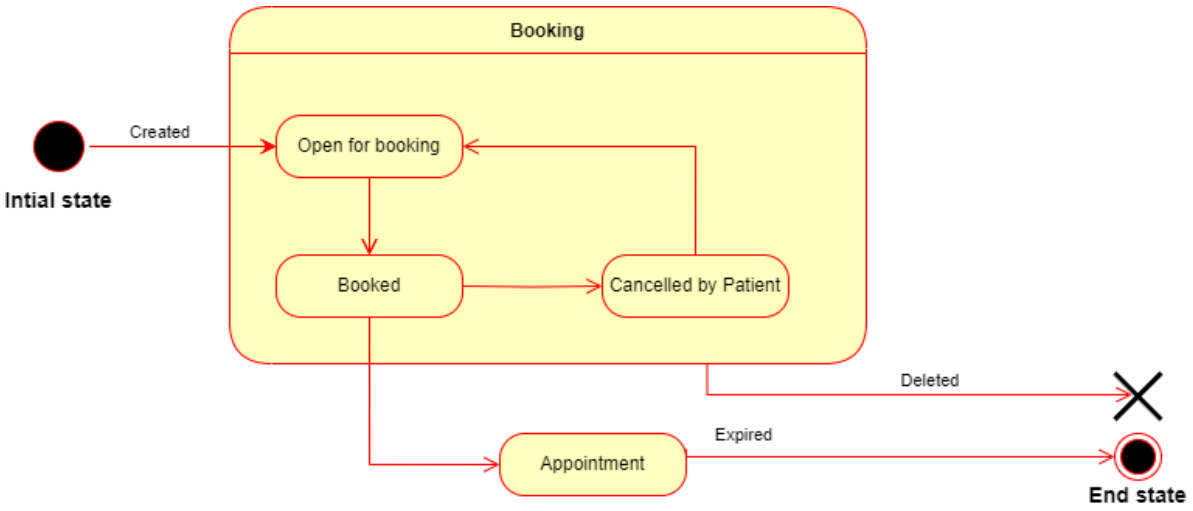


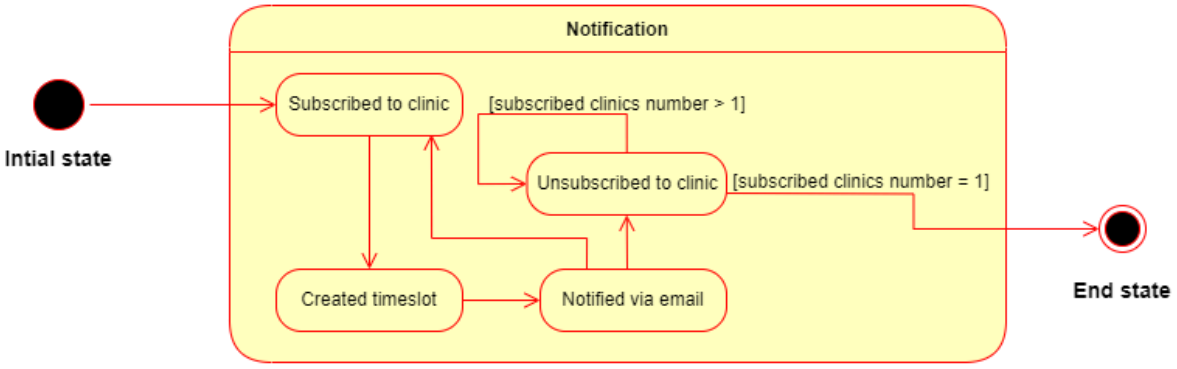
https://drive.google.com/file/d/1AtmZG-_LN1Fs6Nno2QWCgT0S86wHhrhR/view?usp=sharing

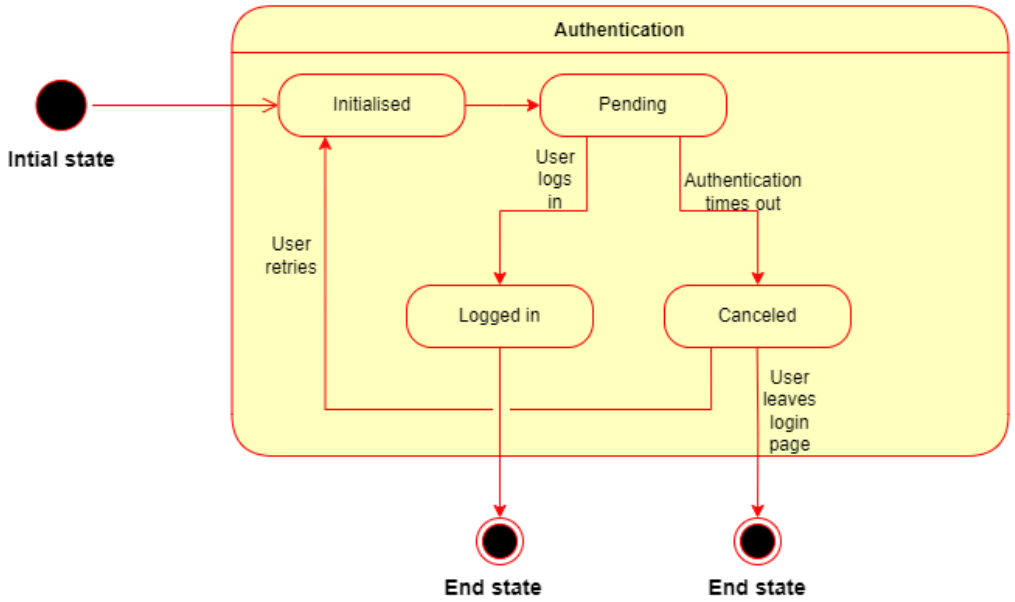
Description:

The diagram is broken down to show specific interactions of dentist registering timeslots, patient booking timeslot, and dentist cancelling timeslot. The diagram shows the process of login, dentist registers time slot, patient books time slot, dentist cancels time slot and notification functions between different services in the distributed project. The User Service is responsible for the login authentication of patients. The booking service is responsible for booking and the business logic related to time slots. The notification service is responsible for handling the notification service. Clinic service is responsible for the business related to the clinic and dentist. All services communicate through MQTT and work together to implement the business process.

State diagram:





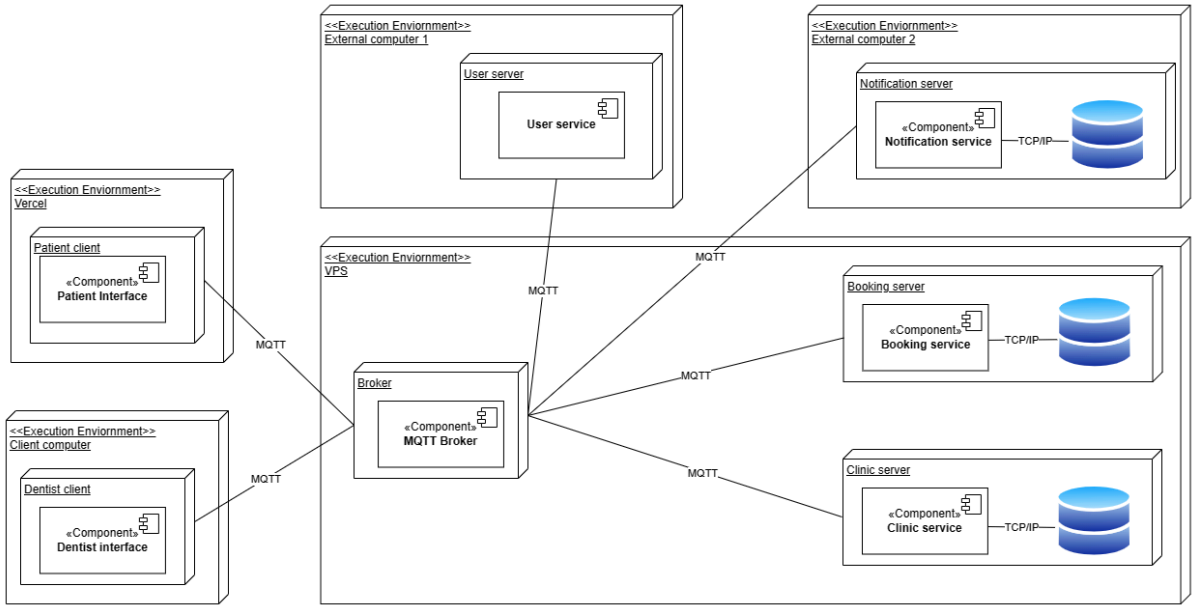


https://drive.google.com/file/d/1OfPDtCoQS4mSxTeZ7m8nH5A5ndyk-H-k/view?usp=sharing

Description:

The diagram has been changed from class diagram to state diagram that represents the logical view. It depicts the booking state, notification and authentication.

Deployment diagram:



https://drive.google.com/file/d/1qjY1a3jqriqDBDPF3byFc-hm3eBVxh87/view?usp=sharing

Description:

Changes include the removal of the user service database as it is not required as there is nothing to store due to login through BankID. Furthermore, distributing the services into multiple devices. MQTT is used directly for communication between the clients and the servers. This simplifies the project structure and reduces the transmission latency that can be introduced by the API Gateway layer. Increases the decoupled nature of the system even further.

4. Development Process

The decided plan is to follow a similar structure to scrum where we have daily standup meetings which will be on discord, though if everyone is available can be held in-person too (During weekdays, upon need will occur on weekends). The duration will range from 10-30 mins at max. Furthermore, a fixed meeting will be held on Monday which can be a couple of hours in length regarding retrospectives and planning for each week (which in turn will help manage each milestone which is alternative weeks). Friday will be kept as backup or an addition to Monday whenever required. To aid and keep a log of the problems, Wednesdays will be used to have a brief meeting to discuss the outcome of the checkpoint with the TA and/or teachers. Certain meeting rules:

- Daily meeting will be online for short duration (unless otherwise stated)
- Mondays and Wednesdays meeting will be in-person (unless otherwise stated)
- Unnotified delay of max 5 mins is allowed, the team must be informed if you are running late
- If a member can't attend a meeting, they must let the group know a day beforehand (unless something sudden occurs)
    - In replacement, they can write a message to the group via Discord of what they did and plans of action with queries they might have

Code contribution

Gitlab rules:

- Code reviews: Merge request must be assigned to a person and there must also be a reviewer who cannot be the assignee, if merging and pushing to the main
- Commit message must follow this structure:
    - **Format:** #issue no. This is commit message
    - **Example:** #999 Delete everything
- Each issue has a user story and acceptance criteria and is connected to a specific milestone
- When creating an issue, create a branch from the issue and when needed create merge request from that issue (when merging to main)
- Each requirement that is made by the group will be a gitlab milestone
- The milestones provided by the teacher will act as sprints which we will be labels in gitlab to sort of issues
- Gitlab's kanban board will be used to track opened, on-going (which will be a label), and closed issues
- Code documentation: write comments for each file to give general info, add additional comments where required if the code is difficult to understand

Testing:

- MQTTX and/or HiveMQ will be used for testing MQTT
- We use JEST for unit testing, vitest for patient webclient
    - Unit test includes testing the encryption, and login
    - Unit tests are applied where necessary, suitable, and feasible; for instance, not for stateful parts of the application like database operations. Code interfacing with databases proved impractical to unit test, and is therefore only covered by integration testing
- We use JEST integration testing for testing mqtt connection with the database in the Notification service

- Test ESLint so there are no linting problems
- Apache JMeter + MQTT JMeter library to stress test connection, pub/sub certain topics
    - Booking test: Pub: booking/timeslot/book, sub: bookings/createBooking
    - Clinic test: Pub: clinic/clinics/get/all, sub: clinic/clinics/get/all/response
    - Dentists test: Pub: clinic/dentists/get/all, sub: clinic/dentists/get/all/response
    - Timeslots test: Pub: booking/timeslot/get/byDentist, sub: clients/PatientUI/booking/timeslot/get

Code style:

- We use Prettier and ESLint for code formatting

Roles:

Jitish Padhya: Scrum master and documentation responsibility of development process and architecture, Meetings, project developer
Henrik Lagrosen: Project developer, architecture designer, document editor
János Litkei: Project developer, architecture designer, document editor
Shiyao Xin: Project developer, architecture designer, document editor
Yingchao Ji: Project developer, architecture designer, document editor

Gitlab tracing explanation:

To begin with, tracking requirements to issues begins with the main DENTIGO repo which has each requirement as a sub-wiki page and has an associated high-level user story (issue) linked to it which acts as an epic (collection of issues) in the DENTIGO repository. Using the linked items, branch specific and in depth issues are grouped together in the main repo. Following on the completion of all linked items will lead to the completion of the main high-level issue. The specific issues in the linked item had an associated branch starting with the issue number along with every commit message including issue number at the start for tracking all the way from a commit to the requirement and vice versa.

5. Justification for grade 5

Encryption

To ensure data integrity and privacy, we encrypt all sensitive information sent over MQTT using both asymmetric and symmetric encryption algorithms. We also use similar asymmetric algorithms, making it possible to provide secure authentication while minimising the risk of leaks of private credentials.

The client-server communication follows the following communication pattern. The data is encrypted for transit using RSA-OAEP and AES-256-CTR. The client initially generates an AES key. When a request to a service is to be made, the client starts by downloading the public key for it from a secure HTTPS server. The client encrypts the payload with the AES key, then encrypts the AES key with the downloaded public key. The client then sends the encrypted AES key, the encrypted payload, and the iv (initialization vector) to the server. The response will be encrypted with the same AES key.

The response shall be the encrypted response and the IV. The client decrypts the response using the AES key and the IV.

BankID

The use of BankID was used as a replacement for the ordinary login/signup method of username or email with password. This firstly gives a more authentic sign-in as it is most commonly seen across Swedish applications. It also allows for secure identification of users, as personal details will be available to the system. This is an extension to what was required, which was to be able to just distinguish between the different users.

The webclient will initialise the communication by sending a MQTT message to one of our servers, which will initialise the connection with the BankID API. The server will repeatedly query the API for updates and information used to generate QR codes. It will send parts of that back to the client that initialised the login process. When logged in, it will take the social security number from the BankID API. It will generate a hash from it, then sign that hash with the private RSA signing key. The signature and hash will be sent to the client, along with other data from the BankID API.

Authentication

The servers authenticate the users using the UserID and signature, which are generated by the User Service and sent from the web client. It will verify the validity of these by downloading a public key for the User Service.

Timeslot filter

The patient interface allows the user to filter dates from and/or to that they desire the appointment to be. After filtering, only clinics, dentists, and time slots that meet the conditions will be displayed, and patients can be browsed and booked more conveniently. This is simply done through having a date picker and checking in the database for the relevant database. Though, goes further than the requirements that is just to show the time slots.

6. Sprint Retrospective

Name	Reflection
Jitish Padhya	All tasks were completed in time. Mainly worked with JMeter stress testing and documentation of the final report
Henrik Lagrosen	Everything went well. Worked mainly on the monitor tool, encryption for bookings, race condition prevention, and various fixes
János Litkei	Everything went well. Worked on the time slot cancellation feature in the Dentist CLI and the Booking Service, as well as the containerization and automatic deployment of the Booking Service and the Clinic Service.
Shiyao Xin	Tasks finished on time. This sprint went well. Faced some challenge on integration test for CI, tackled the problem with teammate's support. The rest of the issues that I worked on for this sprint went smoothly.

Yingchao Ji	All tasks went well. Mainly response for the filter search function and the real-time response function. Also involved in the JMeter stress test and improving responsive pages.

What went well?

We finished all tasks in time.

What will be improved for next sprints?
N/A

What decisions did we make about software architecture and distributed systems?
(UNCHANGED)

What decisions did we make about technologies e.g. for distributed systems or user interfaces?

We added the use of Apache JMeter for stress testing and added its MQTT JMeter library to test the functionality from connection to the broker, publishing and subscribing to topics and disconnection.

Based on our knowledge of software development processes, what relevant concepts or methods did we apply?

- **Agile Methodology:** We embraced Agile principles to foster collaboration, adaptability, and feedback of teacher and TA throughout the development lifecycle. Regular sprint planning, daily meetings, and retrospective sessions.
- **Scrum Framework:** We implemented Scrum as a framework within the Agile methodology, organizing work into sprints. Roles such as Scrum Master, Product Owner, and development team members.
- **Gitlab Version Control:** Utilizing version control systems like Git enabled us to track changes, collaborate seamlessly among team members, and roll back to previous versions if needed.
- **Code Reviews:** Regular code reviews were conducted to promote knowledge sharing, identify potential issues early, and maintain coding standards.
- **Continuous Integration and Continuous Deployment (CI/CD):** Automation of building, testing, and deployment processes through CI/CD pipelines ensured rapid and reliable delivery of software updates. This approach helped in maintaining a stable codebase and reducing manual errors.

6. Project conclusion

Gantt charts for each milestone (for MVP)

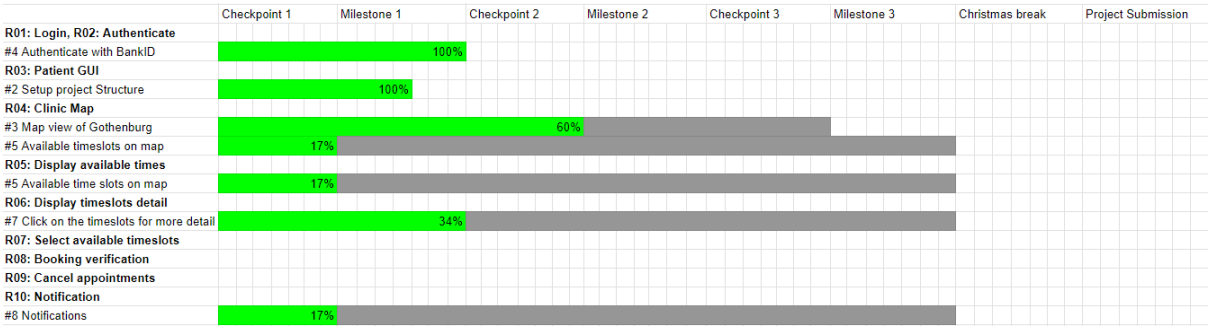


Figure 1. Milestone 1 gantt chart (estimated values gantt chart produced from Milestone 2)

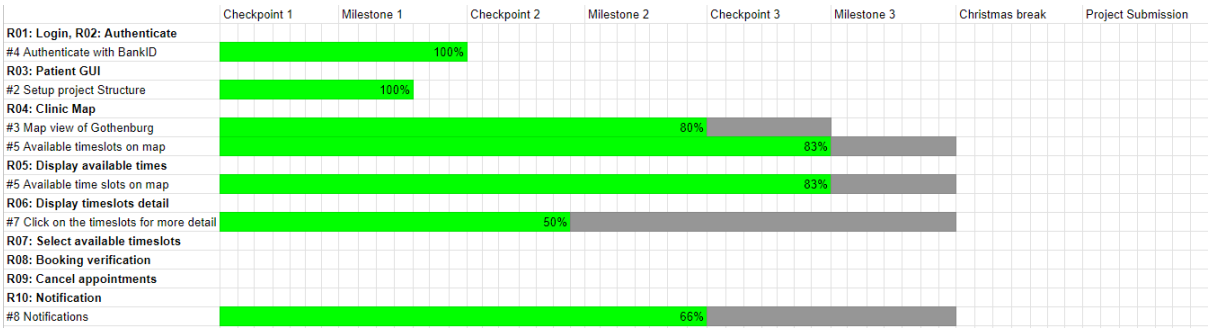


Figure 2. Milestone 2 gantt chart

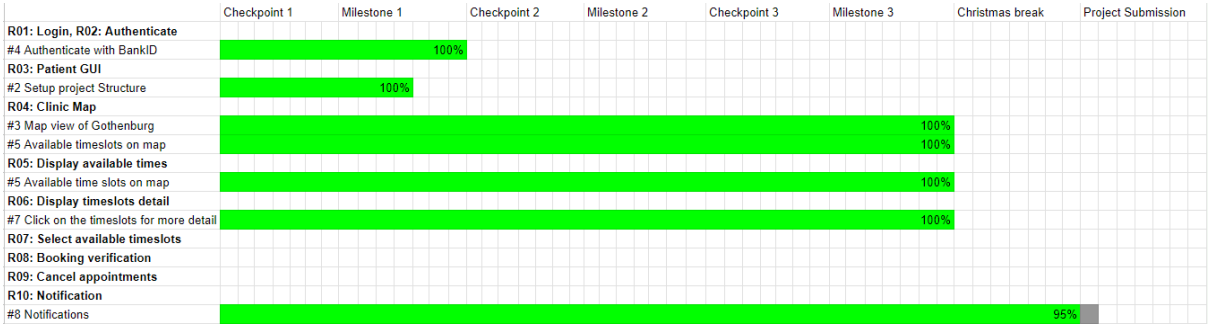


Figure 3. Milestone 3 gantt chart

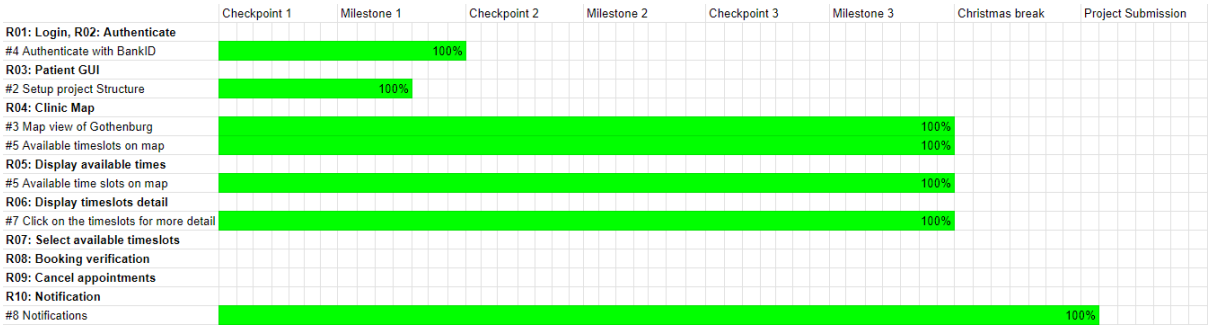


Figure 4. Final submission milestone gantt chart

System analysis

The architecture as described is microservices that communicates directly through MQTT across from the users to the microservices. A single point of failure arises through our MQTT broker based on Mosquitto where during stress testing we discovered that exceeding 800 concurrent users lead to error in connection to the broker of several users.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %
MQTT Connect	800	2913	0	6597	1940.84	0.00%
MQTT Pub Sam...	800	219	0	540	77.01	0.00%
MQTT Sub Sam...	22460	0	0	0	0.00	0.00%
MQTT DisConn...	800	78	0	333	28.60	0.00%
TOTAL	24860	103	0	6597	620.99	0.00%

Figure 5. Stress test table for 800 concurrent visits

Label	# Samples	Average	Min	Max	Std. Dev.	Error %
MQTT Connect	1000	4031	0	7018	2295.26	19.20%
MQTT Pub Sam...	1000	147	0	491	82.54	19.20%
MQTT Sub Sam...	20661	0	0	0	0.00	0.93%
MQTT DisConn...	1000	58	0	373	36.85	19.20%
TOTAL	23661	179	0	7018	937.58	3.25%

Figure 6. Stress test table for 1000 concurrent visits

As seen in figure 5, connection to disconnection from the topic and broker had no error in the process when having 800 concurrent users. While, in figure 6 pushing to 1000 users at a given time lead to rise in errors mainly due to failing to establish connection with the broker (error 502). An alternative that is compatible with Mosquitto is to use MQTT bridges to have multiple brokers so that there can be for example, a primary and a secondary broker or broker for specific services creating a cluster [1].

Secondly, the microservices uptime can be points of failure of features. For example, clinic service needs to be running to login dentists in their interface, user service to login to patient interface, no booking service cannot book/ cancel (as nothing will be visible). While these issues won't be crashes there will be timeouts, errors shown in display, etc.

Gitlab tag dates

MS1: Tag made on gitlab for all repositories (6th Nov 2023 - 15th Nov 2023)

MS2: 15th Nov 2023 - 28th Nov 2023

Repository	**Last commit**
DENTIGO	-
DENTIGO - Booking Service	b6e90fca5db214a62a4d8e5283146317567593d2
DENTIGO - Clinic Service	fc6ee13fdedbd6beca4db9eda332702cf63879a2
DENTIGO - Dentist Interface	-

[1] https://iot.stackexchange.com/questions/318/can-mosquitto-support-multiple-brokers

DENTIGO - Notification Service	fea4468c918972e76e5fd08b9f7c8f85e703fabf
DENTIGO - Patient interface	d82eea70f1006ade45c85e77ff13cdb4ec9a7cb6
DENTIGO - User Service	c54a2190375f4769bd2d1fc7cda427ee2495704c
Monitor tool	-

MS3: 28th Nov 2023 - 12th Dec 2023

Repository	**Last commit**
DENTIGO	-
DENTIGO - Booking Service	e9d9d7596245a51d21da6a618c302fc1f1491e98
DENTIGO - Clinic Service	f5711c22af7dd469953666de14c8b553ab338c5e
DENTIGO - Dentist Interface	aac51c104324bb63a0d5e58a8c1fb1910449af22
DENTIGO - Notification Service	235254deae882d05b6380298125a15d49cedfd70
DENTIGO - Patient interface	0e5bfb55cdce7b8110acf5fc93b39c3712c8ec22
DENTIGO - User Service	645797f663fd06e6de116cf2225fc2e1d4fe6479
Monitor tool	-

Final submission: 12th Dec 2023 - 9th Jan 2024

Repository	**Last commit**
DENTIGO	-
DENTIGO - Booking Service	d7f00981129dd98dedf802c2a63cf852bd47563b
DENTIGO - Clinic Service	5a5a99ccf5163aa5f9da97644f41c27444dae3bb
DENTIGO - Dentist Interface	5b15646ce14c25e7e9f39495b63e47239c8a2505
DENTIGO - Notification Service	fdffcc87d2ae00f016e924bb18a995363b472

	863
DENTIGO - Patient interface	d7e03f56d12087a9cfe9e0978661a2e47315136e
DENTIGO - User Service	645797f663fd06e6de116cf2225fc2e1d4fe6479
Monitor tool	28a1bcc31bfa1f33738e79a84e62938107bdc031