UNIVERSITY OF GOTHENBURG

MiniProject: Distributed Systems Software Architecture Document

Group 8

SAD Report

Munatsir Adam

Mila Mehrvarz

Leila Bencheikh

Ramzi Abu Zahra

Maab Mohammedali

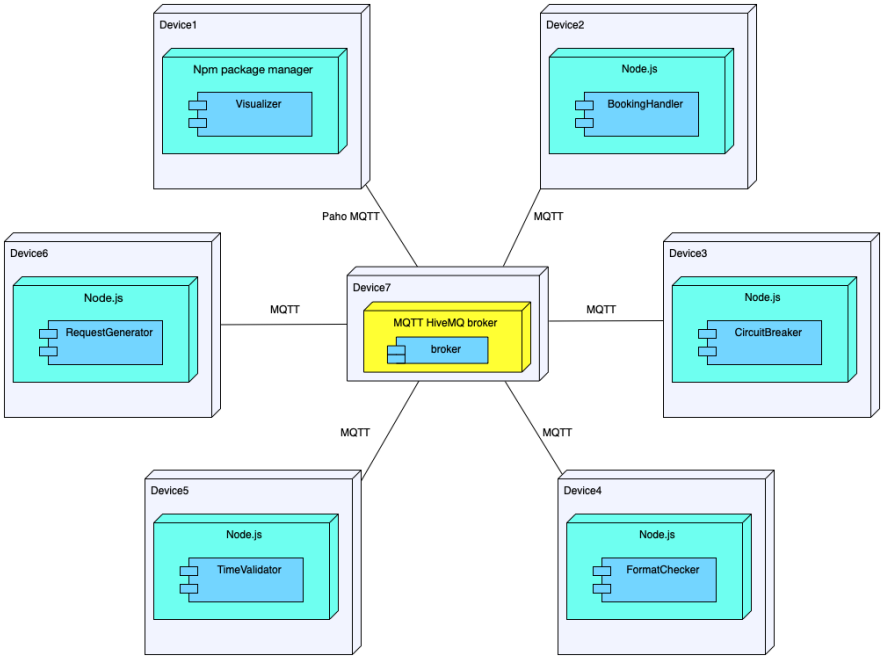
2021

Conceptual Design of Software Architecture

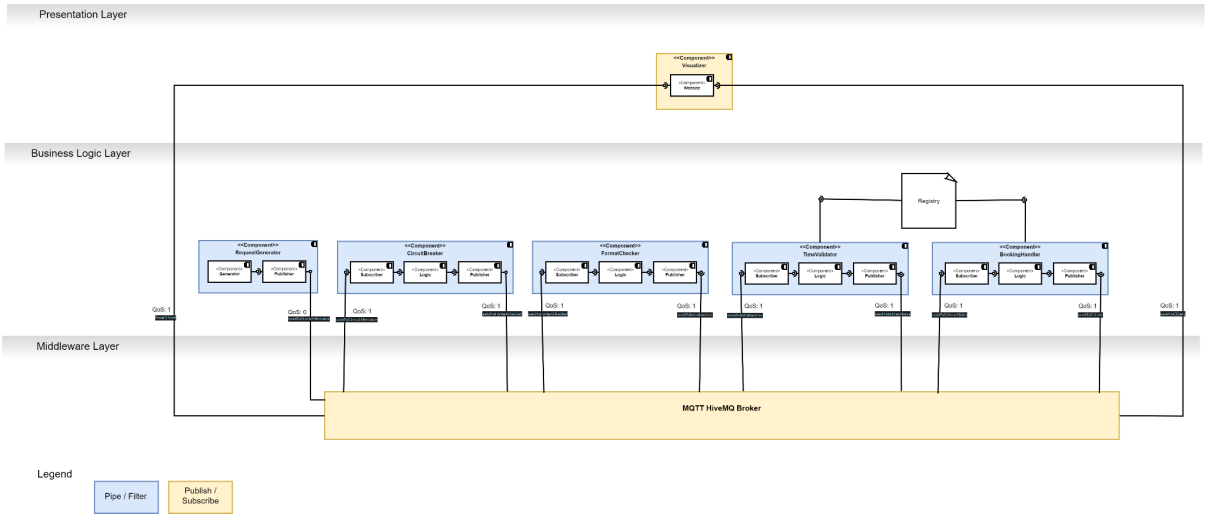
The driving forces around our early and later architectural choices were centered around two main architectural styles in our Dentistimo Booking System, those are Publish-Subscribe architecture and Pipe-Filter. The reason for choosing the Pipe-Filter style is to give our system the ability to further expand when more components are needed to be installed for further filtering or for adding more features to the system. This has proven useful especially at the end of the project when we needed to make sure the system has the ability to manage fault tolerance and stop the flow of requests when a certain threshold is reached. Which explains the additional components that were added later on, namely the circuitBreaker and RequestGenerator. Creating the Pipe and Filter system meant for us at the beginning that a component with the name Pipe should be created. At the beginning of the project the component named Pipe was initially created to manage and delegate the messages between components, but was later removed since we found out that the Mqtt broker delivers the same result thus omitting the need for redundant components like the Pipe.

Some components took the role of being a double filter such as formatChecker which did not just check the format received when a date is chosen by the client, but also the booking of a clinic was also checked within the same component in order to pass the data to the next components with validation. In order to do that in one component we decided to divide the messages sent to the same component according to the size of the message, since the booking message contains more information than the date message(which had a fixed maximum size of 27 bytes) we knew that the same filter could have more than one behaviour while receiving the same topic from the previous component. Another reusable element in our system was the Registry file, which is used by both the dataHandler and the extractData, since redundancy of data is not optimal for a system that depends on updatable data, we opted for making the registry available for both components simultaneously by keeping it in the root folder of the system. In addition to that the data that the Registry depended on was collected using a url based JSON file, allowing the system to be more scalable and manageable.

Mapping Conceptual Design with our Design Choices:

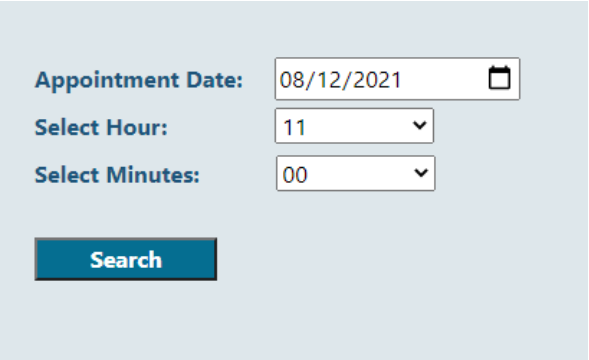


The whole system is centered around the Mqtt protocol, which allows each component to be run independently on a different device, or even multiple devices, where the broker remains the same. Such design choices will allow scalability of the system.



The component diagram above demonstrates how the Presentation Layer is both the starting and ending points of the data flow within our system. Through close observation of

the diagram one might notice that the broker handles all data except for the Registry which is crucial when it comes to the availability of data, and avoiding delays caused by API calls, since the number one priority is to provide the data to the user in the most efficient manner.



The client's ability to customize the date of choice will trigger the TimeValidator Component that directly depends on the registry to iterate and fetch the needed data, without the need for callback functions. The instantaneous results are then delivered to the BookingHandler to notify the client of his success or failure in booking the clinic according to the components own implementation which requires accessing the Registry.