Retrospective model

Sprint 1 – patient-service

What went well:

- I used the REST API specification template to create a REST implementation for accessing patients.

- I managed to communicate with the MySQL SQL database.

- I tested my implementation using Postman as well as with integration and unit tests.

- I managed to set up the Docker microservices infrastructure according to the requirements.

- The user interface has been created, allowing entry and modification of basic patient information.

What didn't go well:

- Docker and MySQL. Exporting sql-file into a docker container for automatic database and tables creation and then filling them out.

  Solution: Created a correct Dockerfile for MySql image creation with environment variables (MYSQL_DATABASE, MYSQL_ROOT_PASSWORD) and indicating the docker entry point.

What could have been better:

- I could have better estimated the time needed to complete certain tasks, which would have allowed for better planning and time management.

What I would like to do differently:

- Establish more accurate time estimates for next iterations to improve sprint planning.

Sprint 2 – history-service

What went well:

- I used the REST API specification template to create a REST implementation to access the notes.

- I managed to communicate with the NoSQL MongoDB database.

- I tested my implementation using Postman as well as with integration and unit tests.

- I managed to set up the Docker microservices infrastructure according to the requirements.

What didn't go well:

- Docker and NoSQL. Exporting history_service_note.json into a docker container for automatic database creation and then filling it out.

  Solution: Exposing the 27017 port linked to container port 28000 for outside access, the user manually sets data inside the container.

What could have been better:

- Create a Mongo image with the DB preloaded

What I would like to do differently:

- Establish more accurate time estimates for next iterations to improve sprint planning.

Sprint 3 – report-service

What went well:

- I used the REST API specification template to create a REST implementation to generate a diabetes risk report.

- Set up the rules for determining risk levels based on the patient's demographics and the triggers present in the practitioner's notes.

- The user interface has been created, to display of a patient's diabetes report.

- I tested my implementation using Postman as well as with integration and unit tests.

- I set up the Docker microservices infrastructure according to the requirements.

- I used Docker Compose to automate the deployment of all Docker containers in the project.

- I have set up the rules for determining risk levels based on the patient's demographics and the triggers present in the practitioner's notes.

What could have been better:

- Triggers location.

What I would like to do differently:

-Would create a separate table with triggers and do extraction inside the code when needed instead of hardcoded. Also, this approach would make it possible to work with triggers more flexibly (adding new triggers, modifying, deleting them) without the need to change them in the code.