SEM5640 Individual Report

# Personal Evaluation

Each point of contribution (what I did) needs to state:

* How I worked as part of the team?
* Critically evaluate (reflect on it) the good and bad of my work?
* How would I do it differently?
* Module Registration
  + Dom was assigned to write module registration but there were no foreign keys and problems with it to the point where it was completely unfunctional although the logic he had written for the CSV upload and parsing endpoint did mostly work.
  + I rewrote the module registration to have alternate models that linked to a database using surrogate auto id primary keys.
  + I changed the code to have common conventions in naming, types used and comments.
  + I abstracted the db context access through use of an IDatareposioty as recommended in the practicals.
  + I fixed bugs in the controllers I found with my jmx tests.
  + I changed return status codes and url routes to be more sutiable.
  + I made controller and data repository access operations asynchronous.
  + I added logging to the file controller. Although no sure this was needed.
  + I refactored and debugged the CVS upload file controller actions.
* Frontend
  + <TODO>

## Process Contribution

At the start of the project I suggested we use sprint and product backlogs to track our tasks. I put together two Excel <REF> documents one for the Product Backlog and one for our weekly Sprint Backlogs. I was strongly pushing for using these documents because I know they are a key part of Scrum <REF> and I had read that failing to maintain a proper backlog was a serious sign of a lack of discipline in a Scrum project which often results in Scrum failing. I think my initial intention to be committed to as close a scrum-like process as possible was a good thing. Dom agreed to this but did not contribute to the design of the backlog. This backlog was not used after the first two weeks because it was done with uncertainty and the team did not engage with it. How to break down tasks and what exactly to include in the backlog was unclear. I think we really needed a way to plan and track tasks but only something that would work for this team and this project; not something from a text book that was to be abandoned after two weeks.

I suggested we record our end of week meetings. These were intended to be a combination of Scrum’s Sprint Review and Sprint Retrospective rolled into one. I drew up the format for these minutes but their use did not last more than a week. When recording the meetings both of us gave ideas and I wrote them down in my tabular format which was part of a dynamic Word <REF> document. I did not commit to this enough although the original idea would have been a good way to maintain a project review history. These were abandoned because the team found no value in them. I think we didn’t really know what to do if the result of the review was “We didn’t finish the work” or “The work is currently in progress”. Maybe committing to an honest critical review of the work completed so far would have been a better option than not reviewing at all.

Throughout the project I used Gitlab for version control and my use of it was successful. I think using branches to always keep a working master would be an improvement. I suggested we use different Gitlab group projects for each microservice which worked well for both of us.

## Documentation Contribution

I suggested we use a uniform formatting to make our deliverable documents more professional and easier to read. I took the template for this format from the 2nd year group project module so it was no work on the team’s part to create the format. This idea was a good contribution because it was an easy solution to documentation that required no investment of effort with the resulting template easy to use and fit for purpose. If doing the project again I would use this format but perhaps make the document visually nicer using latex to reflect my own taste.

I wrote the Project Plan deliverable. This document was unfinished and a poor piece of work. It identifies a few good things but does not expand enough. I think it was supposed to be revisited but got forgotten as other things took priority. Taking this document seriously would be an improvement if redoing the project. – remove this its just negative & WORD COUNT???

I spent a bit of time reading into example test strategies for small scrum project. I shared the resources I found with Dominic because he was assigned the test plan deliverable as a task. I should have probably not done this because this research is overlap with the work Dominic took on in the Test Plan deliverable.

We started doing design for the project early. I pushed for us to focus on Entity Relationship (ER) diagrams and API design first. I prioritised these artefacts of design because I thought detailing the data responsibilities of each microservice and how they exchange data would help us the most in mapping out our understanding of the whole system. The ER diagrams were in the form of UML ORM diagrams without the object mapped classes half of the diagram. The API design was a table of REST endpoint URLs along with a description and status codes for each HTTP verb (GET, POST, PUT, DELETE). Dom agreed with the ER diagrams but did not fully understand the API design because at the time he was unfamiliar with REST and interoperable web services. Latter towards the end of the project he was familiar and was responsible for the updating of the API design document to reflect our final system changes. However, at the time during the start of the project both ER and API design were made by me. The ER part of the ORM diagrams was a well done and important diagram. The API was lacking in perhaps explanation and an exhaustive list of possible return status codes. Focusing more on the API design as a team would be something to improve for next time.

It was recommended we draw up a UML Component diagram early on and this task was assigned to Dominic. The diagram was not complete properly so I remade it in the next sprint. We met as a team to discuss system component interaction from the very start so there was no excuse for not doing this. I believe he was under confident with UML and I know he had no familiarity with the Visual Paradigm Software. I did show him how to use the software but his lack of effort to read into how to make a component diagram was the reason this had to be done by me. If doing this again I would try show my team member example diagrams for them to learn from, although this is difficult with if they are not motivated to learn.

I researched Solr and wrote the Solr Technical Report deliverable. The research was brief because the benefits of Solr are clearly advertised on its website. I spent most of the time on this task walking through the official tutorial and playing with the technology. I believe this appropriate use of time given my confidence in its fit to our system. Although the report is short and could be more professional with slightly more expanded content.

I drew a Use Case diagram for the project in our Visual Paradigm project. The diagram was made, retrospectively towards the end of the project, by both me and Dominic at a single computer. This diagram should be made at the start of the project design to map requirements, if doing the project again I would prioritise this more at the start.

I wanted to produce full class diagrams for all the microservices because I believe that is a minimum of expected design. However due to time constrains they were not completed. I did add the model classes to complete the ORM diagrams and I finished a full class diagram of the Module Registration but did not include it in the report because the others were not done to go with it. If doing the project again I would be refactoring class diagram design while refactoring code and I would push for the whole team to do the same. At the very end of the project I found a plugin <REF> for Visual Studio <REF> that allows drag and drop auto creation of class diagrams; this tool is fantastic and would do this for the team.

## Testing Contribution

I wrote unit tests for the Module Registration microservice. I completed unit testing of ModuleController.cs, StaffController.cs, StudentController.cs but did not finish FileController.cs. In these tests I made use of Moq <REF> to mock the functionality of the external database. The code for this was good quality although more tests and a minimum coverage of FileController.cs would have been ideal. I had to move of this because of time constraints. If doing the project again I hope we focus on tests from the beginning thus allowing us enough time for them. I would also hope I do not have to write tests for someone else’s microservice.

Test were not written for the other microservices due to lack of time. Unit testing should have been throughout the project from the start of development, with time spent at the start to get working examples of testing for each technology. I spent time trying to get Junit <RERF> working for Java EE early in the project but failed because I couldn’t get an embeddable EJB container into unit tests which is required for any meaningful tests.

I used JMeter tests to test message store, notification and module registration endpoints having a single .jmx file for each microservice. These worked well for what they were testing although there could have been more test coverage. To test the Module registration CSV upload I used BASH scripts with a single CURL <REF> command because of problems with using JMeter. This was done well and I would use this approach again. Both me and Dominic made use of this type of testing.

## Deployment Contribution

I added a Dockerfile to each microservice along with a runner.bs script to take down, rebuild and start each project in a Docker <REF> container. This simplified deployment to just pushing code to the remote repository, pulling it on the Docker machine and running the runner script. This method worked well for both me and Dominic.

## Implementation Contribution

### Message Store Microservice

I wrote the Message Store microservice in Java EE. There were difficulties with getting this application further into development because for four weeks I couldn’t get basic functionality working this was due to multiple problems with JSON (de)serialisation, the main problem being non-functional JAXB annotations. After updating the version of Glassfish <REF> I was using from 4.1 to 5 the main problem was gone, I could then find and eliminate the other bugs.

The Java EE microservice was involved I used JPA to ORM the database, this included managing a self-referential entity, of my design, for the messages. I had issues with serialisation of a many-to-many relationship in the model causing an infinite loop and eventual stack overflow error, I made alterations on the database to avoid this. I achieved JSON serialisation through use of EclipseLink Moxy <REF> as an implementation of JAXB. The JAXB annotations being added to model classes to specify serialisation behaviour. Custom message body writers and reader were also used for serialisation.

I used the Glassfish <REF> Java EE server to serve the application and managed the server through NetBeans <REF> the Glassfish domain admin console. I connected my applications to both a local PostgreSQL <REF> database and a live one deployed on the university’s Docker <REF> machine throughout development. The API service of the message store was built using the Jersey <REF> implementation of JAX-RS <REF>.

A branch of the message store was created to contain the addition of Solr <REF> search. The branch was never remerged due to lack of time but most of the search requirements were met. The addition to the project was a search controller that used a single JAX-RS endpoint posted key-value pairs required for searching. The controller code made use of the SolrJ <Ref> Java API to index and query the message data with Solr. The result was then returned as custom JSON.

The work on the Message Store microservice was done completely by me. Having another team member with some familiarity of the Java EE technology would have been a big help. Perhaps next time ensuring each member has some tasks in each technology would better the team’s flexibility. I got stuck on the JAXB/JSON bug for a long time and did not move to something else. If doing this again I would move onto implementing other high priority features to make the best use of time available.

### Notifications Microservice

I wrote the Notifications microservice in ASP.NET <REF>.

The Quartz <REF> package was used for running daily and hourly email jobs on repeat.

User setting were stored in a PostgreSQL database and accessed via mapped ORM model classes using Entity Framework Core <REF>. Database access was abstracted using an interface to allow for future ease of unit testing with a mocking framework.

An email service was written using the provided SmptClient class. This was injected into the jobs to send email summaries. HTML formatting was included in this email service to visually improve the emails. The email formatting of was very basic with not all message data included. The plan was to revisit it and redesign it later but this was not done due to time constraints.

### Module Registration Microservice

### Front End Microservice

# Group Evaluation

How well group worked?

I felt I had all the work to do and no one to help me. Dom did not understand things properly which is fine because we here to learn but the problem is he took no care or want to learn for himself. Everything he did had to be redone by me. I feel I was stretched too thin because I couldn’t properly focus on one thing given. I had to fix his work or jump prematurely between different things. It was tiring because I had to initiate every pro-active thought or decision.

Dom would not draw design UML or do any testing of his own back. I introduced JMeter as a way to test endpoints and he did take that on properly but that is all and only after pushed him to do so. I did very little testing myself I really wanted to focus on it from the start but delays in the project and no encouragement or help from my group on the testing part meant any testing problems I had were always put on the back burner.

Managed working with docker well. Dom had a knack for it learnt things of his own back and showed me some stuff. Direct connections of our local running code to the actual databases meant we could all check what was going on for each other and in a way being able to inspect the database your team member is modifying, in my opinion, made us more of a team

DOM COMPLETELY AVOIDED LEARNING ANY JAVA EE

The component diagram was not listed to be done because we felt that the diagram provided in the project requirements was enough so not as to need an official component UML before development.

How successful was the development methodology? Would use dev method again?

Evaluate group performance and output? What could be improved?