SEM5640 Group Project
Group Report

Author: Dominic Parr & Morgan Jones

Config Ref: SEM5640.2019.tp
Date: 17th January 2020
Version: 1.0
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. OVERVIEW

In this project, our task was to design the Siarad system specified Appendix E. The system as a whole was a messaging system to enable students and lecturers to discuss topics on modules in a modern interactive way, similar to Blackboard (Bb) [1].

The system needed to allow for a multitude of users which were able to access specific material relating to the modules they were registered on; to notify these users when there was activity in their respective modules, and to facilitate specific searching for messages amongst other things.

The key outputs from this project were: The technical work (the Siarad system), this Group project report, Test Plan, Project Plan, Solr [2] Technical Report, and the UML diagrams.

The technical work is broken down in to microservices and the design/implementation of this is described in the respective sections.

The group project report is broken down into the following sections:

- **Requirements** – In this section we discuss about the final set of requirements that were agreed upon and justify any requirements that may had been removed or added.
- **Development Methodology** – In this section we discuss about the methodology we followed throughout the project, the advantages and disadvantages this methodology yielded, the difficulties it entailed, and the rationale for selecting the methodology.
- **Design** – In this section we discuss the design for the system, presenting the initial designs that were created, we also talk about the appropriateness of these designs for the specific solution we pursued.
- **Implementation** – In this section we discuss how we developed the system, as well as the problems we encountered and how we overcame them. This section also talks about technical changes made to the design as a result of certain implementations.
- **Testing** – In this section we discuss the test plan that was made in the design stage of the project, as well as how it was enforced throughout the project, and the different styles of testing that were pursued as well as their appropriateness to each section of the project.
- **Status** – In this section we discuss where the functional state of the project at the end of development as well as any bugs that currently persist in the system.
- **Critical Evaluation** – In this section we evaluate the effectiveness and suitability of our development methodology and the outputs we produced. We also explore what we would had done differently if we started again, and what we would want to do given more time.

The test plan, project plan, Solr [2] Technical report, microservice endpoint list and requirements spec are included within the report as appendices and are referenced and evaluated at the relevant sections.

2. REQUIREMENTS

During the project, there was a list of requirements that the final system should employ, in this section we will review these requirements explaining how they were implemented, or if they have not been, why, along with the justification for it.

The system produced misses some of the functional requirements, it also misses some functionality that is not implicitly stated as a functional requirement, for example, it does not have SignalR [5] implemented to allow for live notifications; these issues are covered in detail in section 7 of this report. The reasoning for this primarily relates to the scope and lack of resources of this project and is covered in detail in section 8.

2.1. Messages

M-FR1 – Message Content
Messages are text-based content items within the system. Each message has the following information:

- User – the user id who wrote the message.
- Message body – the content of the message, written in Markdown [3] format

Previous, this requirement had another condition to do with emoticons, however, it was discussed with the client that the functionality involving emoticons was no longer required.

Markdown format was not looked into due to its lower priority and time constraints towards the end of the project.

M-FR2 – Referring to other users
A message can refer to other users by using @userid within the body of the message. That can generate a notification to alert the other user that they have been mentioned in a message.

This functionality is implemented through users getting emails if they are tagged in a message. Tags are parsed from messages in the message store microservice.

M-FR3 – Messages Groups
Messages are grouped by groups, within modules. A module has at least one group, which is the main group. Messages in a group are ordered by the time that the message was created.

This functionality is implemented with exception of the minimum one group. Currently modules can have zero groups.

M-FR4 – Creating Groups
Any user can create new groups. When the group is created, an announcement is made in the main group. The group will be displayed in a list of groups for the module. A group is available to all users within the module.

This functionality has been implemented with the exception of the main group announcement.

M-FR5 – One-to-one Messages
A user can send messages directly to another user.

Private groups are partly implemented.

M-FR6 – Replying to messages
A user can reply to any message in the group. Replies are shown in the order in which they are created.

Messages can be added but there is no way to associate a message as a reply despite full support for this in the message store because the front end UI is incomplete.

M-FR7 – Editing Messages
A user can edit a message that they wrote. When a message is edited, there is an indication to everyone that it has been edited.

This functionality is not implemented.

M-FR8 – Deleting Messages
Any message can be deleted by members of the module staff or by administrators.

Students can delete their own messages if there are no replies to the messages.

When a message is deleted, it will be removed from the group of messages shown to users.
A deleted message will not be removed from the Message Store; instead it will be marked as deleted so it is not shown in the group of messages.

This is implemented.

M-FR9 – New message Indicator
When new messages have been posted, the system will update the user interface to indicate that new messages are available within modules and groups.

This functionality has yet to be implemented.

2.2. Module Registration

MR-FR1 – Module Information
The system will allow administrators to create modules, with the following information:

- Module Code, e.g. SEM5640
- Title, e.g. Developing Advanced Internet-Based Applications.
- Academic Year, e.g. 2020 will represent the year from September 2019 to August 2020.
- Staff members, which is a list of user ids for the staff working on the module.

For this requirement, we allow administrators to create a module, in addition to the initial requirements, we require that the Campus Code is also provided (e.g. AB0 – Aberystwyth), this is not controlled by a dropdown menu in case new Campus Codes are created in the future, the administrator would need to create the module with this in mind.

We do not allow for a list of staff members to be added, instead we have the main coordinator of the module listed, administrators can check the module and register staff members on it via the interface. This change is then persisted through the database, this was a change in functionality due to the authentication not working as intended, and thus modified so the system could still employ authentication, the details surrounding this are covered in section AA-FR1.

MR-FR2 – Editing module information
The information about the modules can be edited by administrators.

This is implemented as expected, following the same data specified in MR-FR1. The data that is changed is persisted through to the database and the list displaying the modules is re-rendered with the appropriate changes.

MR-FR3 – Student List
A module will have a list of students who are registered on the module. The University's central student management system contains a list of students taking a module. A file can be exported from the system in a CSV format. This system will provide a facility for an administrator to upload that student list for each module.

This requirement has only been partially implemented. A list of students can be attained for a registered module, this is only accessible to administrators and can be viewed in the edit module section, this also allows an admin to remove and add students to a specific module. The system does not allow a list of students on a module to be exported.

The system does allow for a CSV formatted student/module lists to be uploaded correlating to the example CSV files provided by the client. The functionality to create the CSV file is within the code but has been commented out as it contained minor bugs and there wasn't enough time to fix it. This same functionality also applies to staff on a module, allowing the administrator to add or remove registered staff members, this is to fulfil the modified functionality of MR-FR1.

MR-FR4 – Updating the student list
If a CSV list is uploaded to a module where there are existing students for the academic year, the module list is updated.

The system allows for a CSV file to be uploaded; if a general module list or staff list is uploaded, it will truncate data from all tables, this is essentially a "clear all" from the system. When they enter a student module list, it will not truncate the data as this would cause the previously uploaded module list to be erased.

One of the challenges we faced when designing this functionality was the system claiming that a duplicate key was being added when we tried to create the module – student link, this eventually was fixed with further validation checks on the data.

MR-FR5 – Module Membership
The Module Registration part of the system will provide a resource to the rest of the system.
It will enable other parts of the system to find out which modules a user is a member of for a given academic year.

The system provides this functionality as expected, the users' specific information is displayed on their home page as they log in. The system by default will only pend the modules a student is on for the current year.

2.3. Search

S-FR1 – Search
A user can search for messages that match text entered by the user. The search is limited to any modules that the user is associated with.

This functionality has been implemented but not in the working system. Solr [2] search has not been integrated although working separate.

S-FR2 – Search Filters
The search can be filtered in the following ways:

- Across all groups in a module.
- Within a specific group.
- Across all modules in a given academic year.
- In all modules that a user is associated with.

This functionality has been implemented but not in the working system. Solr [2] search has not been integrated although working separate.

S-FR3 – Search Results
The search results will display the messages that match the search text. Where the message is a reply to another message, it should be possible to expand the result to show all messages that follow the searched message, it is not required to show messages previous to the searched message.

This requirement was appended with the client due to difficulties with displaying the previous messages of the searched message. This functionality has been implemented but not integrated to the front end.

2.4. Notifications

N-FR1 – Register for Notifications
A user can register for notifications from the system. The system will present a set of notification types, as specified in N-FR2. A user can change the registration at any time.

This functionality was implemented as expected, it is displayed as a settings tab on the navigation bar.

N-FR2 – Notification Types
The system will provide the following types of notifications:

- Daily summary – request a daily summary of messages within a specified group or module. The summary can indicate how many messages were created during the day and provide a link to view the messages in the system.
- Mentions – request a notification when the user is mentioned in a message. The notification will include the text of the message and a link to see the message in the system.
- Replies to a message – request a notification when a specific message is replied to.

The user is able to opt in to all these settings, and then specific the frequency in which they should occur (depict as hours) as specified in section N-FR3, all these settings can be found in the settings tab described in section N-FR1.

N-FR3 – Frequency of Notifications
The system will provide a mechanism to send notifications at set time intervals throughout the day. For example, once an hour, the system will check if any conditions have been met for the notifications. If they have, the notifications are sent to the relevant users.

This is implemented, the notifications are sent to the user via email, mentions and replies are also notified to the user on the front end server. The interval is a set integer corresponding to which hours the user receives emails. Example: If the user has an interval of three the user will get emails every three hours.

2.5. Authentication and Authorisation

AA-FR1 – Authentication
The system should be configured to use an LDAP [4] server within the Computer Science department to authenticate users. In addition to LDAP [4], the system should have an alternative authentication method for development and test purposes.

Although the LDAP [4] was configured for the application, using the LDAP [4] server to authenticate users did not work. Because of this the system was designed to incorporate the alternative method that was used for development and testing. Admin users were added to the system (stored within the Identity Server) and these admin users control the roles that all other users have (including other admin users). We also added an option for registration which would create a default user with the credentials specified, an admin would then have to authorise this user.

Unfortunately, this implementation means there is not a lot of automation for the admin user, this implementation is described in detail in Section 5.

AA-FR2 – Authorisation
Users may be identified in the LDAP [4] server as staff. This information can be used to distinguish between staff and student user categories.

The complications of this was specified in section AA-FR1, along with the alternative approach.

2.6. External Interface Requirements

EIR-1 – Appearance
The system should be developed as a set of microservices. There should be a suitable User Interface that allows access to the different facilities; there might be one user interface that interacts with the different services. User access to the service is through a web interface that can be accessed through modern web browsers.

This is implemented as expected.

EIR-2 – Internationalised Interface
The system should be internationalised so that the user interface can be available in different languages. English and Welsh should be supported in this version of the system.
For delivery, only English language localisation need be provided.

Internationalisation is not been implemented, however, naturally English is supported. The reason it was not implemented is due to the complexity required for a minor feature, as well as the time consumption. Since we did not have a lot of available time, this was prioritised as a lower priority.

3. DEVELOPMENT METHODOLOGY

We chose to use the Scrum [6] Agile development methodology for our engineering process because it is a well-known and generally successful approach to engineering dynamic projects, although the requirements of this project were original set, we were informed by the client that these requirements may change at any time. The reason we wanted to pursue an agile methodology is because there were a lot of unknowns within the project seeing as we did not have much experience in developing projects of this type and scope.

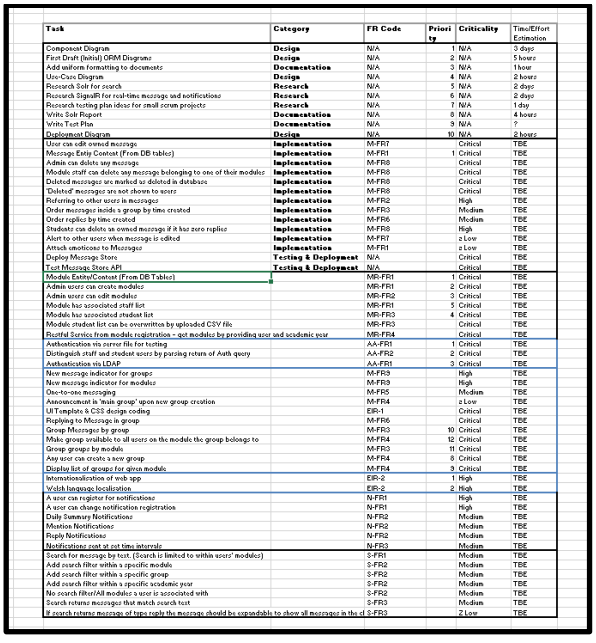
As well as Scrum [6], we adopted other software engineering practices, such as pair programming from the development methodology, Extreme Programming [7]. ->

Scrum would usually have a scrum master assigning tasks that would be undertaken during a sprint; because we were a group of two, we omitted the scrum master component and relied on our active communication and understand to come to accords on what tasks to undertake.

In this methodology we developed our application in sprints; each sprint was planned to be one or two weeks depending on the size of the task that was committed to. At the end of the sprint we would have a review and retrospective of how the week went, this would outline the positive and negative qualities and strategize on how to improve on the next sprint. We originally want to accurately estimate the amount of time and effort required for each task and would then use that to better determine the order of the tasks that we should commit to, in addition to this we rated each task by priority which helped us further determine the order in which tasks should be undertaken.

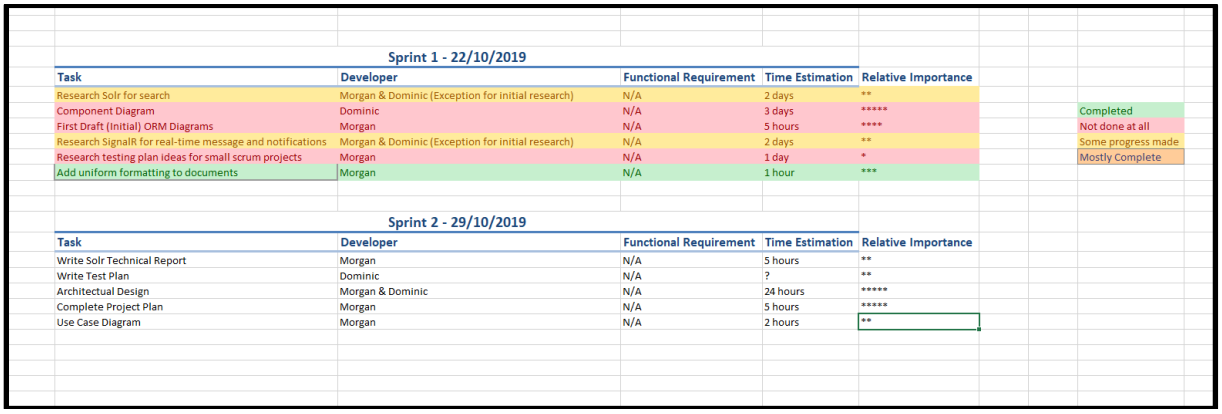
Toward the start of the project, we were maintaining a high-level product backlog of major tasks. We also formally documented our weekly review/retrospectives.

Since the size of our development team was limited to 2, we did not uphold the Scrum practice of daily stand-ups; we instead chose to meet a few times a week for collaborative work and have an official weekly meeting. Our weekly meeting was intended to be combination of classical Scrum sprint planning, review and retrospective all rolled in to one; the reason for this was to prevent the process from becoming too cumbersome.



Task	Category	FR Code	Priority	Criticality	Time/Effort Estimation
Composer Diagram	**Design**	N/A	1	N/A	3 days
First Draft (Initial) ORM Diagrams	**Design**	N/A	2	N/A	5 hours
Add uniform formatting to documents	**Documentation**	N/A	3	N/A	1 hour
Use-Case Diagram	**Design**	N/A	4	N/A	2 hours
Research Solr for search	**Research**	N/A	5	N/A	2 days
Research SignalR for real-time message and notifications	**Research**	N/A	6	N/A	2 days
Research testing plan ideas for small scrum projects	**Research**	N/A	7	N/A	1 day
Write Solr Report	**Documentation**	N/A	8	N/A	4 hours
Write Test Plan	**Documentation**	N/A	9	N/A	?
Deployment Diagram	**Design**	N/A	10	N/A	2 hours
User can edit owned message	**Implementation**	M-FR7		Critical	TBE
Message Entity Context (From DB tables)	**Implementation**	M-FR1	1	Critical	TBE
Admin can delete any message	**Implementation**	M-FR8		Critical	TBE
Module staff can delete any message belonging to one of their modules	**Implementation**	M-FR8		Critical	TBE
Deleted messages are marked as deleted in database	**Implementation**	M-FR8		Critical	TBE
'Deleted' messages are not shown to users	**Implementation**	M-FR8		Critical	TBE
Referring to other users in messages	**Implementation**	M-FR2		High	TBE
Order messages inside a group by time created	**Implementation**	M-FR3		Medium	TBE
Order replies by time created	**Implementation**	M-FR6		Medium	TBE
Students can delete an owned message if it has zero replies	**Implementation**	M-FR8		High	TBE
Alert to other users when message is edited	**Implementation**	M-FR7		2 Low	TBE
Attach emoticons to Messages	**Implementation**	M-FR1		2 Low	TBE
Deploy Message Store	**Testing & Deployment**	N/A		Critical	TBE
Test Message Store API	**Testing & Deployment**	N/A		Critical	TBE
Module EntityContext (From DB Tables)		MR-FR1	1	Critical	TBE
Admin users can create modules		MR-FR1	2	Critical	TBE
Admin users can edit modules		MR-FR2	3	Critical	TBE
Module has associated staff list		MR-FR1	5	Critical	TBE
Module has associated student list		MR-FR3	4	Critical	TBE
Module student list can be overwritten by uploaded CSV file		MR-FR3		Critical	TBE
Restful Service from models registration - get modules by providing user and academic year		MR-FR4		Critical	TBE
Authentication via server file for testing		AA-FR1	1	Critical	TBE
Distinguish staff and student users by parsing return of Auth query		AA-FR2	2	Critical	TBE
Authentication via LDAP		AA-FR1	3	Critical	TBE
New message indicator for groups		M-FR9		High	TBE
New message indicator for modules		M-FR9		High	TBE
One-to-one messaging		M-FR5		Medium	TBE
Announcement in 'main group' upon new group creation		M-FR4		2 Low	TBE
UI Template & CSS design coding		EIR-1		Critical	TBE
Replying to Message in group		M-FR6		Critical	TBE
Group Message by group		M-FR3	10	Critical	TBE
Make group available to all users on the module the group belongs to		M-FR4	12	Critical	TBE
Group groups by module		M-FR3	11	Critical	TBE
Any user can create a new group		M-FR4	6	Critical	TBE
Display list of groups for given module		M-FR4	9	Critical	TBE
Internationalisation of web app		EIR-2	1	High	TBE
Welsh language localisation		EIR-2	2	High	TBE
A user can register for notifications		N-FR1		High	TBE
A user can change notification registration		N-FR1		High	TBE
Daily Summary Notifications		N-FR2		Medium	TBE
Mention Notifications		N-FR2		Medium	TBE
Reply Notifications		N-FR2		Medium	TBE
Notifications sent at set time intervals		N-FR3		Medium	TBE
Search for message by text. (Search is limited to within users' modules)		S-FR1		Medium	TBE
Add search filter within a specific module		S-FR2		Medium	TBE
Add search filter within a specific group		S-FR2		Medium	TBE
Add search filter within a specific academic year		S-FR2		Medium	TBE
No search filter/All modules a user is associated with		S-FR2		Medium	TBE
Search returns messages that match search text		S-FR3		Medium	TBE
If search returns message of type reply the message should be expandable to show all messages in the cl		S-FR3		2 Low	TBE

Figure 1 - Example Initial Product Backlog



Sprint 1 - 22/10/2019

Task	Developer	Functional Requirement	Time Estimation	Relative Importance
Research Solr for search	Morgan & Dominic (Exception for initial research)	N/A	2 days	**
Component Diagram	Dominic	N/A	3 days	*****
First Draft (Initial) ORM Diagrams	Morgan	N/A	5 hours	****
Research SignalR for real-time message and notifications	Morgan & Dominic (Exception for initial research)	N/A	2 days	**
Research testing plan ideas for small scrum projects	Morgan	N/A	1 day	*
Add uniform formatting to documents	Morgan	N/A	1 hour	***

Completed
Not done at all
Some progress made
Mostly Complete

Sprint 2 - 29/10/2019

Task	Developer	Functional Requirement	Time Estimation	Relative Importance
Write Solr Technical Report	Morgan	N/A	5 hours	**
Write Test Plan	Dominic	N/A	?	**
Architectual Design	Morgan & Dominic	N/A	24 hours	*****
Complete Project Plan	Morgan	N/A	5 hours	*****
Use Case Diagram	Morgan	N/A	2 hours	**

Figure 2 - Example Initial Sprint Backlog

28/10/2019

What was good?	What was bad?	Changes & Improvements?
Extra work was added in the form of spike testing technologies: SignalR, JPA, Alternate use of eclipse IDE	Decisions were blocked (design decisions in UML diagrams)	Consider best options for communication: Email or fb or ...etc
Good reflection throughout sprint about process and possible improvements	Communication was poor and infrequent	Ensure we meet twice a week for project work in the same room (as set out in our project plan) even for non-coding tasks
Achieved set tasks for sprint	Estimations for backlog items was way-off (too high)	Add our project meetings to our calendar
	New items of work cropped yet we had no process for how to deal with them as a group	Choose better estimates that properly reflect amount of expected work time on each task
	Too bogged down on docs need to make practical work at least equal priority	Use issues on gitlab for highlighting new things/tasks that need to be done that are not already in the sprint or product backlogs. They can then be fixed or added to the product backlog for a following sprint
	Too few task set to trigger proper involvement (procrastination)	

Figure 3 - Example Initial Spring Retrospective Output

Ethics

In this project, the question of ethics is raised since we're storing user data (username, password, forename, surname) as well as any content they might send over the system (message data), in the scope of our system we only use test data with dummy users and therefore violate no ethical constraints. The system would allow for an admin to push real data to it, but we do not expect this to happen as it would potentially result in random students being emailed from the email store.

One ethical question that might come up is malicious attacks to attain the users' LDAP [4] credentials which would allow them to access other systems outside the scope of this project. This does not apply to our project as we do not use the LDAP authentication service, and instead use our own identity authentication; this was due to a complication within our system but it allows for less sensitive data with a similar level of security due to the end points for the identity server not being exposed.

4. DESIGN

The application is designed as four separate micro-services. The front end microservice that the user interacts with is a standard dynamic web application, the other three are microservices that expose REST APIs [8]. The microservices communicate through REST calls to these APIs, the data format used for the REST calls is mostly JSON [9], although the Module Registration microservice does take in data in the format a CSV file for the purpose of updating module, staff and student lists. The microservices are listed below along with a description of their function and a component diagram to outline their interaction.

Front End Application
Responsible for handling requests, our applications UI and authentication. The front end communicates with the other three applications. It updates user notification preferences on the notifications microservices, queries the module registration to get information about the users registered on different modules and queries and stores user messages from and to the message store. The front end also keeps a database describing all the message groups on the system. The instant messaging functionality of our application is provided through client-side code that is contained within this application. The client-side library used for this functionality it SignalR [5] provided by Microsoft.

This microservice is written in ASP.NET core 2.2 [10]. This technology was chosen because of the implementation for SignalR [5] requiring this platform.

Module Registration
The module registration is the source of authority on what users are registered on what modules. It provides CRUD operations on modules and CSV file upload operations to be used by admins.

This microservice is written in ASP.NET core 2.2 [10]. This technology was chosen because through the experience we had gained on the project up to this point, we were more comfortable using this technology.

Message Store
This microservice stores all messages in the system and is accessed by both the front end and the notifications. It provides CRUD operations on messages and exposes a search functionality to be used by the front end.

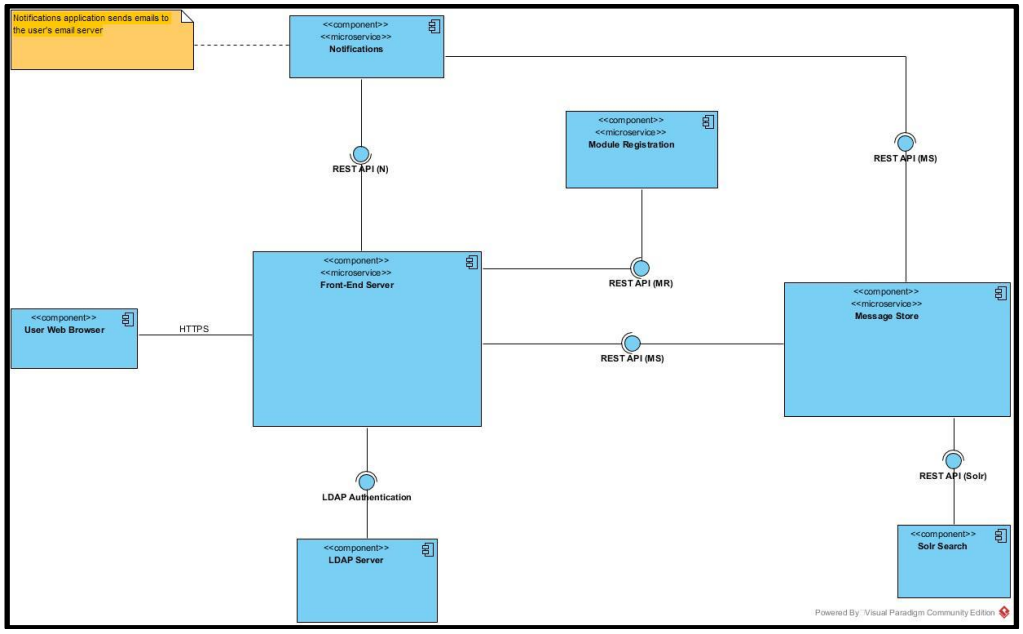
The search functionality of our site was not implemented by us. Instead we used a third-party application called Solr [2], this is an open source and comprehensive web application for searching that can accessed as a service from our own applications. More information is in the Solr [2] technical report included in the appendix B.

This microservice is written in Java EE [11] and deployed on a Glassfish 5 [12] server. This technology was chosen because of the design choice for the message store to communicate with Oracle's Solr [2] this being another service written in Java and with good java API support for interfacing to it.

Notifications
The notifications is a multi-threaded application that sends emails to users based off set preferences. The front end makes calls to its service to update user preferences stored in the application and then at certain points during the day the microservice queries the message store to gather the messages required for its email summaries to users.

This microservice is written in ASP.NET core 2.2 [10]. This technology was chosen because of the easy use of Quartz Package [13], for scheduling jobs to allow for the functionality of multi-threading.

4.1 Initial Design



We started the detailed design of the system by looking at the data each service would store and the operations each service would expose. We modelled the systems data in a set of entity relationship UML diagrams and the microservices operations as a descriptive list of REST API URLs and associated response codes. These diagrams were updated throughout the project as we understood more about the project and refactored our implementation.

Below are ORM diagrams for each of the microservices they describe the data stored in the database and the ORM classes responsible for mapping to those data relations. The outline of our services' REST APIs are included in Appendix A.

4.2 Low-Level Design

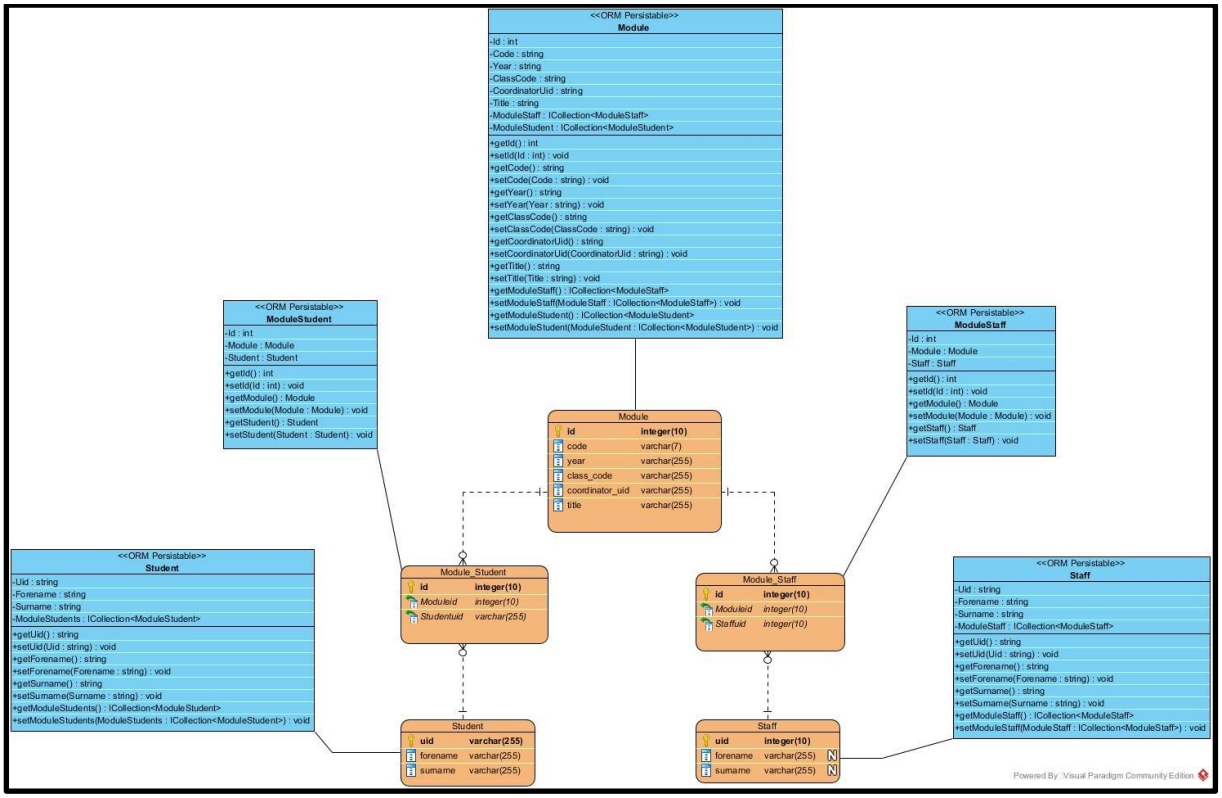


Figure 4 - Module Registration ORM Diagram

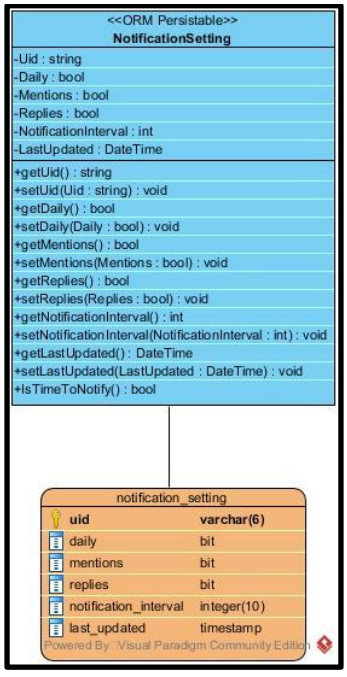


Figure 5 - Notification ORM Diagram

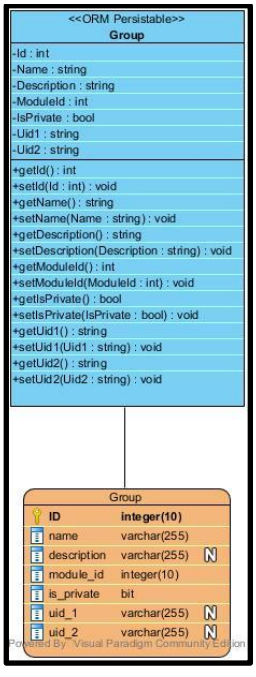


Figure 6 - Front End ORM Diagram

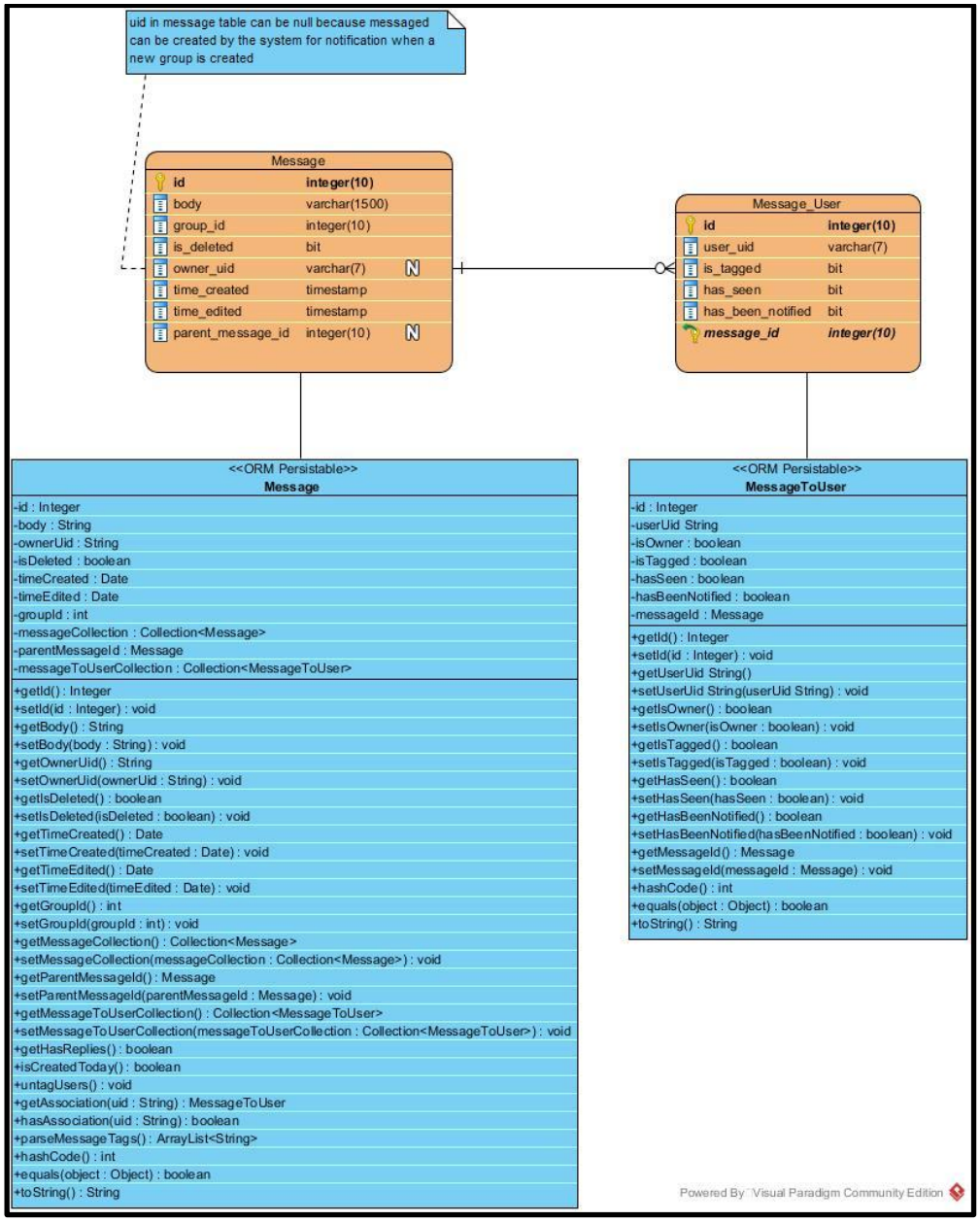


Figure 7 - Message Store ORM Diagram

5. IMPLEMENTATION

In this section, each sub-section is broken down in to two parts (unless state otherwise), these parts are the backend microservice implementation, and the front end integration implementation. All databases within this system use PostgreSQL. All databases use data models to create the tables, and data annotations or the model builder to control the data types and constraints of the data.

For a large portion of the front end interface design as well as the functionality behind the Identity database, the webseries guide presented by "Kudvenkat" [14] of Pragimtech [15] was followed. This includes view models, cshtml razor pages, and the controller classes used in his guides, these have been modified to accommodate our specific needs for the project.

5.1 Module Registration

When Module registration was first created, it supported a table for Student, Module and Staff. In its data model, it used composite primary keys to identify unique attributes, this caused a major issue when trying to create foreign key references that could never be resolved. Despite this we worked around a way to still get the correct data (although it was quite difficult to use and put stress on the front end), however, we agreed that this was not usable in its current state due to the foreign key restriction creating the inability to validate creation and deletion.

The final version of the module registration backend is broken down into several tables; the three main tables are: Student, Module, and Staff. Following this there is also a Module_Student and Module_Staff table that holds the data of which student and staff are registered to what modules respectively, it does this using a foreign key reference to the respective tables.

In the data model, a key difference between the final database and the previous database is that the primary key is controlled by an auto-created surrogate primary key, this allowed for foreign keys to be created as there was no composite restraint. Although the data could be identified from unique attributes, using a surrogate was decided as the best route to take.

Student, Module and Staff have CRUD operations supporting them so that, from the front end, it is possible to populate and edit data within these tables. Module_Student and Module_Staff only support create, get and delete as there is never a need to update this data.

The module registration microservice exposes three POST endpoints for uploading CSV files containing module, student and staff data. On upload of module and staff CSVs the previous data on the server is truncated. For students the previous data is not removed as there are individual files for each module's registered students. This may be a weakness of the implementation as student data will never be removed unless done explicitly by the CRUD operations available to an admin user.

One of the complications of this backend is the unnecessary duplication, this stems from the staff and student table being similar, in a retrospective we thought it would be better to merge these tables, but we knew that would cause a multitude of errors throughout connected systems and decided to keep the model as is. As an improvement to the system the staff and student tables/model would be merged into a single user table/model which would help reduce code duplication.

The front end implementation of module registration is only available to users with the authentication level of "Admin", this is because it performs operations that access the database. There is one exception to this, and that is retrieving the list of modules to display to the user or staff member.

When a user of authentication level "Student" or "Staff" logs in to their account, the front end will call a list of modules that, that user is registered on, it does this by accessing the Module_Student or Module_Staff data table respectively, this is allow the functionality to view the groups for that module. A user of authentication level "Admin" will receive a list of every module, this creates quite a long list, but the admin requires the view to see every module to perform administrative actions.

An admin is given an additional navigation option on their navigation bar titled “Manage”, this opens a drop down menu with the options to manage “Users”, “Roles”, and “Modules”, upon clicking one of the navigation options they will be redirected to that specific page.

On the “Users” page, an admin can view a list of every user within the system, from this section they can edit the user information (forename, surname), or choose to delete the user. On the edit page for a user, they will see a list of authentication roles that, that user has applied to them, from here they can manage the roles and assign whichever role they wish.

From the “Users” page, an admin can also create a user, it is worth noting that when a user is created, they will have no authentication roles applied to them, the admin would then have to go in to that users roles and manage them.

For the scope of this project, there was no need to include the functionality for a users’ password to be changed.

On the “Roles” page, we see a similar interface as the “Users” page, there is the option to create a new role, or edit an existing role. Although the functionality allows for this to happen, it is not advised for any admin to change this unless they are editing the entire system as these roles directly correlate to authentication tags throughout the system.

On the “Modules” page, similar to the other pages, we have the functionality to edit and delete a module, when editing a module we have the options to edit the specific information for the module, manage the students registered to that module, and manage the staff registered to that module, the functionality behind this operates the same as the other pages.

We also have the option to add a new module, or upload CSV files, the possible CSV files we can upload are the module file, staff file, or student module file. These inputs are modelled after the example CSV files, we received from the client. Some additional parameters are requested for Staff and Student Module file, both requiring the campus (i.e. Aberystwyth, Mauritius), and the Student Module file requiring the year it relates to. This allows the admin the mass-populate the database given the correct format of files.

5.2 Authentication and Authorisation

The backend support for authentication and authorisation is handled via Identity, a built in asp.net library. This created a database with auto-generated fields of type IdentityUser. We added some additional fields to IdentityUser so we could have greater control over what is happening within the system. The fields we added were; a Boolean variable for if the system display for the pages should be English, the default value of this is true, if the variable is set to false then we’d want to load up a Welsh version. We also hold a Forename and Surname cell, this is because we don’t want to constantly pull this data from the database, and instead use it locally.

Unlike the other databases, this database has no HTTP methods supported, this allows for a layer of security as the port is exposed to the system. The only way for a malicious user to edit this would be to attain the login credentials for an admin user or access the remote docker container with valid credentials.

As mentioned in section 5.1, when a user is created by either a new user or an admin user, they will initially have no authorisation to the system, this means they will only be able to view home page which will simply specify that they’re not registered on any modules (default view).

In an ideal solution we would use LDAP [4] to handle this for us, but due to technical difficulties we couldn’t get LDAP [4] to work (though it is still included in the project, commented out in the login section); more specifically, we could get an unauthorised connection to the LDAP [4] server, but this did not contain the information required, we could not establish an authorised connection to get the specific details of the connecting user. Despite this we still needed authentication and authorisation

within the system so although this requires more effort for an admin user, it creates the same level of result to validate the systems functionality.

Every page has a default level of authentication required (this is for the user to have an account of any role or no role), for certain views such as retrieving a student list, you would need to have the role of student else a default view would be returned. All admin pages have admin authentication required. The only pages that allow anonymous view is the register and login page, all unauthorised requests get redirected here.

5.3 Message Store

The Message Store microservice is built using JavaEE 7 [11] running on a Glassfish 5 [12] server. It uses the Jersey [16] implementation of the JAX-RS [17] java community standard for building RESTful web services in JavaEE [11]. The message store makes use of the Java Persistence API (JPA) for interfacing with a PostgreSQL [18] RDBMS via object relational mapped (ORM) model classes.

The messages are stored in a self-referential relation to capture the parent-child relationship of message replies. A message-user relation is used to describe the relationship between every user and every message. There were many issues with stack overflow during model serialisation and for this reason this association table is a single table not a link table, in the end this turned out suitable as the only piece of data for a user is the user id so there is not much data duplication despite the small lack of conventional database normalisation.

For serialising & de-serialising to or from JSON we made use of JAXB [19] annotations. Originally a set of annotations for XML marshalling, EclipseLink moxy allows the easy use of these standardised annotations for JSON also. Custom message body writers and readers had to be used to achieve the desired result with JSON serialising. A long time was spent on getting serialisation to work for the project due to a bug in the version of glassfish (4.1) we were using at the time. During this period GSON [20] was tried as an alternative for JSON serialisation.

The main code for the message store is in the controller that is responsible for the endpoints it exposes (MessageStoreREST.java). There are endpoints for CRUD on messages as used by the front end application and endpoints for retrieving message summaries for given users, made use of by the notifications microservice. There are also endpoints for creating associations between users and messages. This is to address a flaw with our design as not all users who should have associations with messages do until these endpoints are called by the front end.

5.4 SolR

The message store exposes a separate controller (MessageSearchREST.java) with a single POST action endpoint for search. This controller access Solr [2] for the indexing and querying of messages through a java API called SolrJ [21]. The search endpoint takes the following key-value pairs as criteria for the search: user id, filter type, filter & search text. Where the filter type is one of {GROUP, MODULE, YEAR, ANY}.

5.5 Notifications

The notification microservice is built using ASP.NET with two main functions in mind.

- To expose a CRUD REST service to the Front End application for updating of user email preferences.
- To routinely pull messages from the message store and email them to users based of these preferences.

The CRUD service is the standard .net implementation of a REST API relying on a single entity framework core model for representing a user's email settings. The notifications service connects to a

PostgreSQL database using Npgsql [22] package to achieve this. The settings for a user are three Boolean values describing if they want daily, reply & mention emails, along with a time interval describing when they are sent the reply & mention notification emails.

The multi-threading required for the routinely pulling is messages is achieved through use of the Quartz package [13] for .net core [10]. There are two quartz jobs running on two different schedules. A daily job that is responsible for sending users' daily message summaries that triggers every 24 hours. An hourly job that is responsible for sending users summaries of messages they have either been tagged in or messages that are replies to any of their own messages, this job triggers every hour and uses a value in user settings to determine if it is the hour for a user to receive an email. The message store keeps track of all notifications so no user will be spammed duplicate notifications.

The quartz jobs make use of two custom services one message client service class to get the messages for notification from the message store microservice and an email service class responsible for formatting and sending the messages over email via Simple Message Transfer Protocol (SMTP) [23] to registered users.

Standard quartz jobs are not able to use scoped services (such as DbContext). To get around this we used a class to wrap the quartz jobs to alter the scope of the service provider they use. The code for this class (QuartzJobRunner.cs) was taken from an online blog post by Andrew Lock [24], this is referenced in code as a comment.

On the front end implementation, the only thing we want for the user to be able to do is opt in to receive notifications, the types of notifications they wish to receive, and the intervals they want to receive this. The default for any given user is that they will not receive notifications. When they click on the Settings tab on the notification, we create the default notification values for them (there is no point in doing this sooner), at this stage the user can then choose to edit those values to start receiving notifications. To opt out of notifications, they would simply need to adjust their settings to be that of default values (specifying to receive no notifications).

6. TESTING

In this project we defined a very brief test plan that encompassed an overview of tests we'd apply to the entire system as we developed it, this was moderately brief as at the time it was created we had yet to start implementing the system and we did not have enough knowledge to accurate gauge what we'd need. Instead of updating this document as we progressed through the project, we instead internally decided the direction to pursue as with our limited resources we wanted to make sure that we were focusing on the most important and prioritised tasks. The test plan can be found in appendix D.

In the test plan we talk about the categories of testing we'd be performing, both manually and automatically; these were unit testing, integration testing, functional testing, and exploratory testing. We did not include acceptance testing or system testing as categories of testing, these tests were performed manually through exploratory and functional testing.

Within the project, unit tests were performed on the backend databases to validate some functionality, we decided that these tests were important to have here as it is the backbone of the front end server, if we erroneous operations here, it is safe to say the front end would not function as intended. It is also a measure to reduce debug time by finding these errors early on as cross-server debugging can prove quite challenging.

For unit testing .net apps we used MSTest along with Moq [25] for mocking depended external services. Our test coverage was small and we could have benefited from taking a more rigorous approach. We had problems getting any unit tests to run for Java EE. This was due to needing to inject an embeddable EJB container into the unit tests for the tests to run. We could not get embeddable EJB container into the tests without which there was little to test.

Integration tests were primarily applied through Apache JMeter [26] using JMX tests, these tests allowed us to test the HTTP requests that are exposed from the database, this is because the front end will retrieve data from the database via these RESTful connections, by validating that the functionality of these end points is as intended, we can assume that any errors that occur on the front end would be down to the implementation of the front end service, and not the database. This allows us to work on the front end service disjoint from the database by giving us confidence in the backend.

For functional testing we exploratory tested the system whilst ensuring it met the functional requirements of the system, we did this to catch out unexpected bugs to fix them as the functional requirements enforce the core requirements of the system. This helped us identify many minor bugs to do with data manipulation within the databases from the front end access. This replaced our system and acceptance tests to a manual system, if this functionality worked during testing then we could say that the functional requirement had been fulfilled, in combination with explorative testing we could ensure the robustness of this satisfaction.

As a whole, the system was exploratory tested with no real goal in mind other than to break the system, this allowed us to find bugs (especially related to erroneous user activity) which we could then fix, or document as bugs in the system.

7. STATUS

7.1 Implemented

Modules can be accessed and viewed by users on the front-end. User roles are implemented with authentication using module registration as a source of authority for users instead of LDAP. CSV files containing data for the module registration micro-service can be uploaded through the front-end by admins. Modules, students and staff can be individually CRUD by admins through the front end.

Groups can be Created, Edited and Deleted by the appropriate user roles. Both private and public groups are supported. Messages can be added to groups and deleted by the appropriate users. Messages are ordered by date of creation. Login and logout functionality is there.

Users can create and update their email settings through the front end for use on the notifications server. The notifications micro-service is working to pull messages from the message store and email users of their notifications based of the updated email preferences/settings. The notifications is completely implemented and given messages and user preferences will email users appropriate summaries both once a day and at each users chosen hourly interval.

The message store is implemented fully however not integrated fully to the front end thus limiting our system. Message CRUD endpoints, message-to-user associations, email user message summary endpoints and support for hierarchical message replies is all there although not made use of due to lack of time on the front end. There are a few bugs as revealed during attempted integration with the front end but these are detailed below bugs section.

There is code for a majority implementation of the search functionality using Solr. Solr is integrated to the message store through use of SolrJ and a working endpoint is made available for search with the required search criteria. This code is not deployed though due to lack of time so it is included as an alternative message store in the code submission.

7.2 Not Implemented

The application front end is unfinished. Its UI lacks support for private messaging and message replies. SignalR has not been made use of so instant message propagation is not available. There is no search functionality available. There is no internationalisation. Messages cannot be edited.

LDAP Authentication has not been used. CSV export has not been implemented.

7.3 Known Bugs

There is a bug with the update endpoint on the message store, lack of time to fix this means the edit message functionality on the front end is non-functional.

There is a bug when trying to create a message with a tag. A tag is a word starting with a '@' symbol this was the intended way to tag users in messages. The message store parses these tags to create associations between users and the messages they are tagged in, however there is a bug in the method that creates these associations on the message store server.

When a user tries to manually type a URL in (for example, from the home page to a group of a module view), they will get an error as specific pages require certain models to be passed into them.

There is not complete role authorisation for all the controller actions meaning some users can see groups they are not members of through simply navigating via a URL.

8. CRITICAL EVALUATION

8.1 Development Process

Our development process was weak despite having initially good intentions and aspirations. Meeting, planning and documenting at the start of the project was done for a while until we started to drift away from our routine process. Being unsure of what to focus on as a useful part of the process was another issue. If we could have had more accurate estimation that would have resulted in better commitment to the work we took on we would have functioned better as a team from the start.

We focused too much on the high-level design instead of getting involved with the development process and technology. Understanding that design is really important as the first part of the project but we were just unsure of what that meant for a project of this type. Perhaps if we went into more low-level design earlier we could both learn more about the technology and start the project with some more valuable design.

The design we did have was actually useful though. The API endpoint list and component diagram gave us a loose understanding on the project and these were evolved and referred to throughout the project. There were some flaws in design and communication that resulted in late additions of endpoints again suggesting we should have gone into more detailed design earlier in the project.

We gave tasks a ordering by priority based of the value we thought their completion would add to the system. Our priority list was high level and viewed microservices as a whole task instead of breaking down the parts of them that come together to form the most critical client valued items. Perhaps instead of developing all functionality at once we should have focused on developing just the critical parts of multiple microservices first and integrating them together better throughout the project. Instead of trying to integrate all at the end. This would see us delivering real end-to-end client value throughout the project.

When we starting coding we stopped our regular meetings because we couldn't commit properly to completing tasks. We couldn't forecast how long each task would take because they kept flowing over to the next week. There was also a blocker Morgan had in message store due to the wrong version of Glassfish recommended to be used. It took a month for a week long microservice to be implemented throwing off project progress.

Our process was not followed towards last half of the project because it was not working for us and the project was very behind. We needed a proper investment to reform the process to be of more value but we just didn't feel we had the time. Next time maybe we should make effort to reform the process before it gets to such a point. A take home message from this would be to not waste time if stuck or blocked on a task. We could instead move on to do something else that would push the project forward or meet together to solve an issue as team, perhaps using pair programming more often to that end.

8.2 Outputs

We feel that some of our code and support for features in the microservices is quite good. However, our actual demonstrable output is poor because of our lack of time for UI and integration with the incomplete front end.

The quality of the UI is poor because it is effectively a skeleton for demonstrating function, the style that is there is just a basic layout produced from using standard bootstrap themes and grid layout. We think we hit quite a few functional requirements even though there is so little in the UI although the requirement are only met at a basic level a more robust implementation would be desired in a final version of the system.

The module registration and notification microservices are working and hit all requirements as we understand them. Our submitted version of the module registration has changes to do with accommodating an alternative authentication which means there is more functionality there than

original in the requirements specification. The message store is completely implemented and although there have been a few bugs in endpoints discovered towards the end of the project during integration most is functioning well as shown by our JMeter tests.

Overall, we think the code in our backend (application tier) microservices is of good quality and addresses the requirements of each service. We have tried to follow good design principles and conventions of the technologies we learnt to use and believe we have achieved this. Design patterns and development principles such as abstraction, dependency injection and mock testing have been used to name a few.

There are some unused or redundant endpoints and pieces of code. Work effort could have been better directed to prevent writing code that does not result in client value despite being of good quality. This would had saved us time.

Quite a bit of time was invested into learning technologies that we did not end up implementing due to time constraints. SignalR, Solr and ASP.NET internationalisation were all looked into. Solr search being actually implemented but not used. More commitment on agreed priorities and less switching between tasks last minute would have prevent this time being wasted.

REFERENCES

[1] I. (. Blackboard, “Blackboard website,” [Online]. Available: https://uk.blackboard.com. [Accessed 10 January 2020].

[2] A. Solr, “Apache website,” [Online]. Available: https://lucene.apache.org/solr/. [Accessed 10 January 10].

[3] J. Gruber, “Markdown,” [Online]. Available: https://darlingfireball.net/projects/markdown/. [Accessed 16 January 2020].

[4] “LDAP,” [Online]. Available: https://ldap.com/ldap-related-rfcs/. [Accessed 17 January 2020].

[5] Microsoft, “SignalR ASP.NEt,” [Online]. Available: https://dotnet.microsoft.com/apps/aspnet/signalr. [Accessed 17 January 2020].

[6] “What is scrum,” Scrum, [Online]. Available: https://www.scrum.org/resources/what-is-scrum. [Accessed 17 January 2020].

[7] Wikipedia, “Extreme Programming,” [Online]. Available: https://en.wikipedia.org/wiki/Extreme_programming. [Accessed 17 January 2020].

[8] Wikipedia, “REST,” [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 17 January 2020].

[9] Wikipedia, “JSON,” [Online]. Available: https://en.wikipedia.org/wiki/JSON. [Accessed 17 January 2020].

[10] Wikipedia, “.Net Core,” [Online]. Available: https://en.wikipedia.org/wiki/.NET_Core. [Accessed 17 January 2020].

[11] Oracle, “Java ee glance,” [Online]. Available: https://www.oracle.com/java/technologies/java-ee-glance.html. [Accessed 17 January 2020].

[12] Java, “glassfish,” [Online]. Available: https://javaee.github.io/glassfish/. [Accessed 17 January 2020].

[13] M. Lahma, “Quartz,” [Online]. Available: https://www.nuget.org/packages/Quartz/. [Accessed 17 January 2020].

[14] Kudvenkat, “Asp.Net core tutorial,” YouTube, [Online]. Available: https://www.youtube.com/channel/UCCTVrRB5KpIiK6V2GGVsR1Q. [Accessed 16 January 2020].

[15] Kudvenkat, “Pragimtech,” [Online]. Available: https://www.pragimtech.com/. [Accessed 16 January 2020].

[16] Eclipse-ee, “Jersey,” [Online]. Available: https://eclipse-ee4j.github.io/jersey/. [Accessed 17 January 2020].

[17] Java, “jax-rs,” [Online]. Available: https://github.com/jax-rs. [Accessed 17 January 2020].

[18] PostgreSQL. [Online]. Available: https://www.postgresql.org/. [Accessed 17 January 2020].

[19] Java, “jaxb-v2,” [Online]. Available: https://javaee.github.io/jaxb-v2/. [Accessed 17 January 2020].

[20] Google, “gson,” [Online]. Available: https://github.com/google/gson. [Accessed 17 January 2020].

[21] A. Solrj, “Apache website,” [Online]. Available: https://lucene.apache.org/solr/guide/6_6/using-solrj.html. [Accessed 17 January 2020].

[22] PostgresQL, “npgsql,” [Online]. Available: https://www.npgsql.org/. [Accessed 17 January 2020].

[23] Wikipedia, “SMTP,” [Online]. Available: https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol. [Accessed 17 January 2020].

[24] A. Lock, “QuartzJobRunner,” [Online]. Available: https://andrewlock.net/using-scoped-services-inside-a-quartz-net-hosted-service-with-asp-net-core/. [Accessed 17 January 2020].

[25] “moq,” [Online]. Available: https://github.com/moq. [Accessed 17 January 2020].

[26] A. JMeter, “Apache website,” [Online]. Available: https://jmeter.apache.org/. [Accessed 16 January 2020].

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	09/01/2020	N/A – Initial Creation	DOP2
1.0	N/A	17/01/2020	Completion	MWJ7 & DOP2

APPENDIX

APPENDIX A

System API Design
Module Registration API

The job of **Module Registration** API is to take a user and an academic year and return a list of modules. Admin users can also CRUD modules.
Consider use of API key in Authorisation header (front-end server will have a hard-coded API key).
The default prefix path for all Module Registration resources is "/api/modules" unless stated differently with a different "/api" prefix.

Resource	**GET** read	**POST** create	**PUT** update	**DELETE**
	Return a list of modules	Create a new module	Method Not Allowed (405)	Method Not Allowed (405)
/{id}	Get a specific module	Method Not Allowed (405)	Update a specific module	Delete a specific module
/year/{year}/code/{code}	Returns a specific module given its module code and year.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/year/{year}/{uid}	Returns a list of modules from a given year that is associated with a specific user.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/user/{uid}	Gets all modules for a specific user.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/year/{year}	Get all modules for a specific year.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/{id}/students	Get the registered students for a specific module.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/{id}/staff	Gets the registered staff for a specified module.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/api/data/students/{class_code}	Method Not Allowed (405)	Adds to the current student and module_student list from CSV	Method Not Allowed (405)	Method Not Allowed (405)
/api/data/staff/{class_code}/{year}	Method Not Allowed (405)	Empty and reload the staff and module_staff list from CSV	Method Not Allowed (405)	Method Not Allowed (405)
/api/data/modules	Method Not Allowed (405)	Empty and reload the module list	Method Not	Method Not Allowed (405)

		from CSV	Allowed (405)	
/api/students	Returns all student entities.	Create a student entity.	Method Not Allowed (405)	Delete all student entities.
/api/students/{uid}	Method Not Allowed (405)	Method Not Allowed (405)	Update a specific student entity.	Delete a specific student entity.
/api/students/{uid}/{mid}	Get a list of module_student links for a specific module.	Add a student to a module_student link for a specific module.	Method Not Allowed (405)	Delete a student from a module_student link for a specific module.
/api/staff	Returns all staff entities.	Create a staff entity.	Method Not Allowed (405)	Delete all staff entities.
/api/staff/uids	Return a list of all staff user ID's.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/api/staff/{uid}	Method Not Allowed (405)	Method Not Allowed (405)	Update a specific staff entity.	Delete a specific staff entity.
/api/staff/{uid}/{mid}	Get a list of module_staff links for a specific module.	Add a staff member to a module_staff link for a specific module.	Method Not Allowed (405)	Delete a staff member from a module_staff link for a specific module.

Notifications API

The **Notifications** application will provide an API to update user notification settings. The front-end server will use this API. The notifications component will also have to access the Message Store's API to get messages to email.
Consider use of API key in Authorisation header (font-end server will have a hard-coded API key).
The default prefix path for all Notification resources is "/api/usernotifications" unless stated differently with a different "/api" prefix.

Resource	**GET** read	**POST** create	**PUT** update	**DELETE**
	Method Not Allowed (405)	Create a new notification settings entry.	Method Not Allowed (405)	Method Not Allowed (405)
/{uid}	Returns a specific users' notification settings.	Method Not Allowed (405)	Updates a specific user's notification settings.	Deletes a specific user's notification settings.

Message Store API

The **Message Store** provides an API to the front-end server and to the notifications application.
The default prefix path for all Message Store resources is "/MessageStore/api" unless stated differently with a different "/api" prefix.

Resource	**GET** read	**POST** create	**PUT** update	**DELETE**
/messages	Get all messages.	Create a new message.	Method Not Allowed (405)	Method Not Allowed (405)
/messages/{id}/{uid}	Method Not Allowed (405)	Create a new association between user and message.	Method Not Allowed (405)	Method Not Allowed (405)
/messages/{id}	Returns a specific message	Method Not Allowed (405)	Updates a specific message	Deletes a specific message
/messages/{id}/seen	Method Not Allowed (405)	Marks a message as having been seen by a user.	Method Not Allowed (405)	Method Not Allowed (405)
/messages/delete/{id}	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)	Marks a message as deleted.
/messages/user/{uid}	Get all of a user's messages.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/messages/replies/{uid}	Get the messages required for a given user's reply summary.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/messages/mentions/{uid}	Get the messages required for a given user's mentions summary.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/messages/daily/{uid}	Get the messages required for a given user's daily summary.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/messages/group/{group_id}	Get a list of all messages from a specific group.	Method Not Allowed (405)	Method Not Allowed (405)	Method Not Allowed (405)
/messages/search	Method Not Allowed (405)	Search the message store using SolR to return messages matching either a specific group module or year.	Method Not Allowed (405)	Method Not Allowed (405)

APPENDIX B

INTRODUCTION

Purpose of this Document

The purpose of this document is to provide a technical report for the usage of Apache Solr [1] within the 2019 SEM5640 masters' year group project.

Scope

This report aims to describe our research of Apache Solr [1] and evaluate its potential for use within the project as an alternative to a bespoke implementation of the system's required search functionality.

Objectives

The objectives of this document are:

- Outline Solr and its suitability for the required search functionality
- Evaluate the adjustments required for using Solr
- Evaluate Solr's interoperability with other system components/applications.
- Compare Solr use to bespoke search implementation

SEARCH WITH SOLR

Solr is an open-source enterprise-search platform, written in Java, from the Apache Lucene project. Solr is a powerful search tool used by many heavily-trafficked websites and applications. Solr's search capabilities are far greater than the search required for our project; therefore all search requirements can be met through use of Solr as alternative to bespoke search implementation.

Solr has a Java API called Solrj [2] that makes it easy for Java applications to interface with a Solr server. We are planning to build our system's message store microservice in Java EE therefore use of Solr for search would be an easy adjustment. Both Solr and Solrj have comprehensive online documentation and tutorials so usage should be fairly straightforward.

Solr would be running on the same machine (Docker container) as the message store microservice but on a different port. The message store microservice would then make use of Solrj to make calls to the Solr collection's REST API. These calls would involve both indexing and querying on the Solr server. Where indexing would provide Solr with the input documents for it to search through and querying would provide Solr with the details of the search itself. Results would then be returned to the HttpSolrClient in the message store application.

Overall we believe use of Solr to be a good choice for this project. Solr is a professional way to provide search functionality to our web-based system. Integrating Solr with our current choice of technology is easy. Use of Solr will be able to meet our search related functional requirements and save us time in not having to code and test our own search implementation.

APPENDIX C

INTRODUCTION

Purpose of this Document

The purpose of this document is to describe our choice of engineering methodology and our initial plan for the project.

Scope

This document describes our adopted version of the Scrum agile engineering methodology and outlines an initial high-level roadmap for the project. The document also states which development practices and tools we aim to use throughout the project.

Objectives

The objectives of this document are to:

- Outline Scrum and our reasons for choosing it.
- Define our Scrum process.
- List our tools and development practices.
- Present an initial project calendar.

CHOSEN METHODOLOGY

We have chosen to follow the Scrum engineering methodology for this project. As it is a well-known agile process for developing software projects.

SPRINT PROCES

- Weekly planning
    - Tasks are chosen by team members to be worked on during the sprint. Tasks are selected based on priority in the product backlog.
    - Estimate the time for tasks before we take them and only take as many as we can commit to for that sprint/week.
- **No Daily Stand ups**
    - Alternatively, we will meet a few times a week for joint project work. During these sessions' issues can be resolved. If problems need resolving outside these sessions the team will communicate the issues over email.
- Weekly review
    - To review code, tests and diagrams too if appropriate.
    - Small retrospective (good? bad? improvements/changes?)
    - Tasks are not complete unless code is properly tested.

TOOLS AND PRACTICES

Some the proposed microsystems that comprise the project will be written in Java EE and ASP.NET. We will use the NetBeans 8 IDE for developing Java EE. We will use Microsoft's Visual Studio IDE for developing in ASP.NET. Junit and MS Test will be used for testing Java and C# code respectively.

We are using docker to deploy our applications on a university machine (m56-docker1.dcs.aber.ac.uk).

PROJECT OUTLINE

SPRINT	TUE (Planning)	WED	THUR	FRI	SAT	SUN	MON (Review)
Zero 15th-21st		Project Handout (16/10/2019)	Research & Spike Testing				
First 22nd-28th	Research & Spike Testing						
Second 29th-4th	Write Documentation						
Third 5th-11th	Document Milestone						
	Code Module Registration						
	Code Message Store						
Fourth 12th-18th	Test file Authentication				Integration Testing & Fixes		
	Code Front-End Server						
Fifth 19th-25th	Code Front-End Server						
Sixth 26th-	Code Notifications						
	Add LDAP Auth						
Seventh	Add Solr Search						
Eighth	Full system integration and extra low priority work						
Ninth	Integration Testing, Deployemnt & Fixes			Project End (13/12/2019)			

APPENDIX D

INTRODUCTION

Purpose of this Document

The purpose of this document is to describe the groups' plan for testing the functionality elements of the system and to ensure a certain standard is maintained across all systems.

Scope

This test plan will apply to all systems within the application, some sections may be more relevant than others.

Objectives

The objectives of this document are:

- To identify the types of testing that will be explored.
- To state a conformity that all tests will follow.

Testing

There are two categories of testing that will be applied; this is manual testing – the practice of a user or users exploring a system to identify bugs by chance via exploratory testing or conforming to a set test table to ensure functionality of the system is as intended. In most cases, both are applied. There is also automated testing – the practice of creating tests using tools such as MSTest (Asp.Net), JUnit (Java EE), Moq.
In this project there will be a higher focus to automated testing, this is primarily because the group size of the project is too small to pour large quantities of time into manual testing of a large system on a regular occasion. Manual testing will be performed at certain milestones within the project. The four types of testing that will be implemented are:

- Unit testing – These tests will enforce consistent functionality across all levels within the project, utilising Moq testing for system behaviour, and other tests to enforce functionality.

- Integration Testing – These tests will involve a mix of manual, and automated testing. The manual testing will consist of a graphical check to ensure that the design remains consistent. The unit tests that had been compiled from all components will be refactored (if necessary) and ran to confirm proper functionality.

- Functional Testing – This will primarily be covered by unit tests that will be written as the code is developed, it will test a range of things with a high focus on system dependencies, such as the databases. At this stage we would also confirm that the test table could be satisfied fully, and any discrepancies would be reported.

- Exploratory Testing – This is a testing exercise in which testers are assigned a loosely defined task to achieve using the software being tested [1]. It will only be performed on the system as a whole a few times throughout the project due to lack of resources; after integration of main components is successful a 'brute force' test will be performed on the system to try to identify obscure bugs.

APPENDIX E

INTRODUCTION
Purpose of this Document
This document describes the requirements for the Integrated Masters Group Project 2019-2020 for SEM5640.

Scope
This requirements specification describes a distributed computer-based messaging system to provide a discussion facility to support student learning in a university. The application will support the realtime distribution of messages, which are grouped by modules. There will be facilities to search the messages and configure when notifications are sent. These requirements describe new software systems that need to be developed.

Objectives
The objectives of this document are:

- To describe the background to the SEM5640 group project application (Siarad).
- To provide details of the criteria that the group project's product must meet.
- To describe the types of interaction with the system which must be supported.
- To describe the technologies that must be used to implement the system.

GENERAL DESCRIPTION

Product Perspective

It is proposed to develop Siarad, a new application to enable students and lecturers to discuss topics on modules in a modern interactive way that offers an alternative to using Blackboard (Bb)[1] [1] discussion forums.

Bb discussion forums are linked to each module. They offer a way for everyone on a module to interact by posting messages, known as Threads, and allowing anyone to respond to those messages. However, the interface relies on full-page refreshes and resembles older formats of discussion forums that are slow to use.

The Siarad system is to provide a central hub for conversation on the modules, making it easier to interact. There are some requirements for the format of messages, but there is scope to develop the features supported within a message to make them more engaging for the users of the system.

A system such as Slack[2] [2] is an example of a modern, interactive way to provide messaging in teams. Whilst the Slack system provides some inspiration, the Siarad system should focus on features that would support learning by students in all departments in a Higher Education environment.

The Siarad system will provide the following top-level features:

- Discussion areas for each module that a student is registered for.
- Ability to create extra discussion areas within a module.
- Ability to have one-to-one discussion between staff and students on a module.
- Search facility to find messages with certain search criteria applied.
- Summary statistics available to staff about the usage of the system.
- Notifications sent to users according to some criteria, e.g. when a user is mentioned directly in a post.
- Administrator access to any messages to investigate any claims of inappropriate messages being sent.

Product Operation

The application will be a set of server-based applications that store and manage messages and related data. These server applications will:

- manage the messages that are written by users and how those messages are linked together, e.g. a set of messages in a module and a set of replies to a specific message;
- provide real-time updates of messages to users who are connected to the system;
- manage the list of modules and users, organised by the academic year;
- search the messages using a set of criteria, e.g. text message or date range;
- manage user preferences for notifications;
- integrate with the department's LDAP service to provide authentication and authorisation

Technology

It has been decided that the application will be developed as a set of cooperating micro-services. This is an architecture style that is intended to allow for the set of services provided by Siarad to be upgraded or modified easily with least disruption to the overall service. The services will be written using .NET Core and Java EE.

Microsoft has developed SignalR [3], which provides real-time communication within an ASP.NET application. The group will use SignalR to provide the distribution of real-time messages within Siarad.

A management team within the University is interested in exploring the open source Apache Solr [4] tool to provide a search facility. It would like the project to investigate the tool as an early part of the work. The group should write a technical report on the feasibility of using Solr to provide the search facility. Following the

[1] 1 Blackboard and Bb are trademarks or registered trademarks of Blackboard, Inc. [2] Slack is a trademark or registered trademark of Slack Technologies, Inc.

investigation, the group will make a choice to either use Solr or produce an alternative mechanism to provide the search facility.

Figure 1 shows an example decomposition of the main elements in the system, which are described in this document.

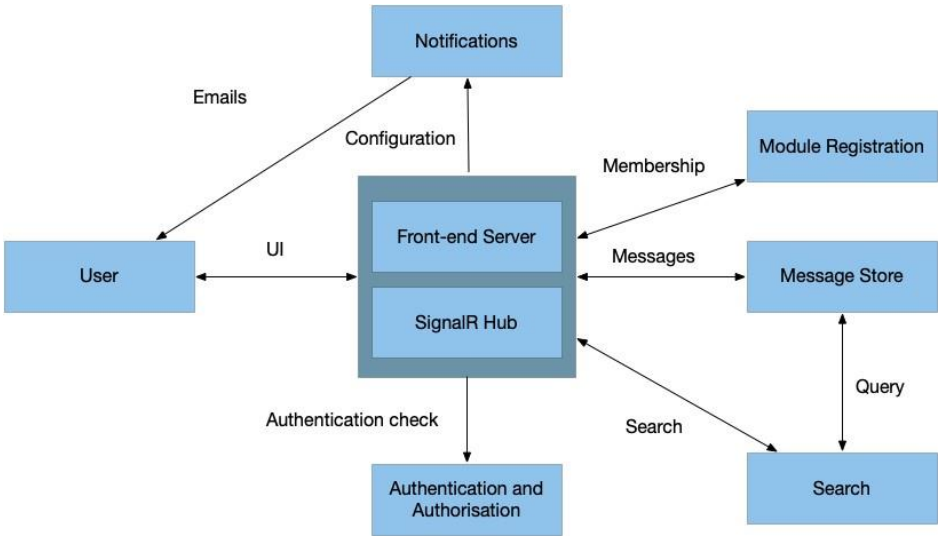


Figure 1: Proposed decomposition of the system

User Categories
The software will have the following categories of users.

- **Student** – This user can post messages – either creating new messages or responding to other messages.
- **Module Staff** – This user can post messages and has the ability to block a message from being viewed by other users.
- **Administrator** – This user is responsible for overall administration tasks, managing the list of modules and students. The user has the ability to block a message from being viewed by other users. The user also has the ability to see any message in the system, including those in a oneto-one communication.

SPECIFIC REQUIREMENTS
Functional Requirements
This section lists the key functional requirements for the systems.

Messages
The functional requirements for the messages are:

- **M-FR1 – Message Content**
  Messages are text-based content items within the system. Each message has the following information:
  - User – the user id who wrote the message.
  - Message Body – the content of the message, written in Markdown [5] format.
  - A list of emoticons that users have attached to the message. Each emoticon has an associated list of user ids to indicate who added that emoticon.
- **M-FR2 – Referring to other users**
  A message can refer to other users by using @userid within the body of the message. That can generate a notification to alert the other user that they have been mentioned in a message.
- **M-FR3 – Messages Groups**
  Messages are grouped by groups, within modules. A module has at least one group, which is the main group. Messages in a group are ordered by the time that the message was created.

- **M-FR4 – Creating Groups**
  Any user can create new groups. When the group is created, an announcement is made in the main group. The group will be displayed in a list of groups for the module. A group is available to all users within the module.
- **M-FR5 – One-to-one Messages**
  A user can send messages directly to another user.
- **M-FR6 – Replying to messages**
  A user can reply to any message in the group. Replies are shown in the order in which they were created.
- **M-FR7 – Editing Messages**
  A user can edit a message that they wrote. When a message is edited, there is an indication to everyone that it has been edited.
- **M-FR8 – Deleting Messages**
  Any message can be deleted by members of the module staff or by administrators.

  Students can delete their own messages if there are no replies to the messages.

  When a message is deleted, it will be removed from the groups of messages shown to users. A deleted message will not be removed from the Message Store; instead it will be marked as deleted so that it is not be shown in the groups of messages.
- **M-FR9 – New Message Indicator**
  When new messages have been posted, the system will update the user interface to indicate that new messages are available within modules and groups.

Module Registration
The system will manage a list of module information. The information will be used to determine which modules are presented to the different staff and students. This section lists the requirements for that module management.

- **MR-FR1 – Module Information**
  The system will allow administrators to create modules, with the following information:
  - Module Code, e.g. SEM5640.
  - Title, e.g. Developing Advanced Internet-based applications.
  - Academic Year, e.g. 2020 will represent the year from September 2019 to August 2020.
  - Staff Members, which is a list of user ids for the staff working on the module.
- **MR-FR2 – Editing module information**
  The information about modules can be edited by administrators.
- **MR-FR2 – Student List**
  A module will have a list of students who are registered on the module. The University's central student management system contains a list of students taking a module. A file can be exported from that system in a CSV format. This system will provide a facility for an administrator to upload that student list for each module.
- **MR-FR3 – Updating the student list**
  If a CSV list is uploaded to a module where there are existing students for the academic year, the module list is updated.
- **MR-FR3 – Module Membership**
  The Module Registration part of the system will provide a resource to the rest of the system. It will enable other parts of the system to find out which modules a user is a member of for a given academic year.

Search
The system will provide a search facility. This section describes the requirements for that facility.

- **S-FR1 – Search**
  A user can search for messages that match text entered by the user. The search is limited to any modules that the user is associated with.
- **S-FR2 – Search Filters**
  The search can be filtered in the following ways:
  - across all groups in a module,
  - within a specific group,
  - across all modules in a given academic year, or
  - in all modules that a user is associated with.
- **S-FR3 – Search Results**
  The search results will display the messages that match the search text. Where the message is a reply to another message, it should be possible to expand the result to show all messages: the original message and all replies, while clearly identifying the message found in the search.

Notifications
The system will provide the option for notifications. The notifications will be sent as emails. This section describes the requirements:

- **N-FR1 - Register for Notifications**
  A user can register for notifications from the system. The system will present a set of notification types, as specified in N-FR2. A user can change the registration at any time.
- **N-FR2 – Notification Types**
  The system will provide the following types of notifications:
  - Daily summary – request a daily summary of messages within a specific group or module. The summary can indicate how many messages were created during the day and provide a link to view the messages in the system.
  - Mentions – request a notification when the user is mentioned in a message. The notification will include the text of the message and a link to see the message in the system.
  - Replies to a message – request a notification when a specific message is replied to.
- **N-FR2 – Frequency of Notifications**
  The system will provide a mechanism to send notifications at set time intervals throughout the day. For example, once an hour, the system will check if any conditions have been met for the notifications. If they have, the notifications are sent to the relevant users.

Authentication and Authorisation
The system will use authentication and authorisation.

- **AA-FR1 – Authentication**
  The system should be configured to use an LDAP server within the Computer Science department to authenticate users. In addition to LDAP, the system should have an alternative authentication method for development and test purposes.
- **AA-FR2 – Authorisation**
  Users may be identified in the LDAP server as staff. This information can be used to distinguish between staff and student user categories.

External Interface Requirements
This section lists general interface requirements for the systems.

- **EIR-1 Appearance**
  The system should be developed as a set of microservices. There should be a suitable User Interface that allows access to the different facilities; there might be one user interface that interacts with the

different services. User access to the service is through a web interface that can be accessed through modern web browsers.

- **EIR-2 Internationalised interface**
  The system should be internationalised so that the user interface can be available in different languages. English and Welsh should be supported in this version of the system.
  For delivery, only English language localisation need be provided.

Performance Requirements
For this prototype, there are no specified performance or reliability requirements.

Design Constraints
The following design constraints must be met.

- **DC-1 Use of Java EE**
  At least one of the microservices must be written in Java EE.
- **DC-2 Use of .NET**
  At least one of the microservices must be written in ASP.NET Core.
- **DC-3 Use of web services**
  The system will use RESTful web services for communication between different microservices.
- **DC-4 Reuse of 3rd party software**
  Use of existing 3rd party libraries for parts of the solutions is allowed. Please note that there is possibility of licensing the system under a commercial licence or a licence such as the Apache licence. Your group would need to check the licence terms for the 3rd party software and discuss with your manager before committing to the use of any 3rd party software.
- **DC5 Use of Docker**
  The services are to be deployed to the Docker facilities for testing and demonstration. The group will be provided with access to Docker, hosted on a University machine.