SEM5640 Group Project

Group Report

|  |  |
| --- | --- |
| Author: | Dominic Parr & Morgan Jones |
| Config Ref: | SEM5640.2019.tp |
| Date: | 9th January 2020 |
| Version: | 0.1 |
| Status: | Draft |

<https://learn-eu-central-1-prod-fleet01-xythos.s3-eu-central-1.amazonaws.com/5d5bfc0d30f9a/1109439?response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27SEM5640_Project_Assignment_Sheet.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200110T191537Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200110%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=7e60f8a5a399a268f65f9ac453c37a01c554a7db916c93e43ae0175e47717a24>

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2015

CONTENTS

CONTENTS 2

1. Overview 3

2. Requirements 3

3. Development Methodology 3

4. design 3

5. Implementation 3

6. testing 3

7. Status 3

8. critical Evaluation 3

REFERENCES 3

DOCUMENT HISTORY 4

# Overview

In this project, our task was to design the Siarad system specified Appendix A. 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 Technical Report [2], 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 Technical report [2], and UML diagrams are included within the report as appendices and are referenced and evaluated at the relevant sections.

# Requirements

**IF WORD COUNT BECOMES AN ISSUE, MOVE REQUIREMENTS TO APPENDICES AND REFERENCE.**

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.

**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.

This functionality has yet to be implemented.

**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 has yet to be implemented.

**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 has yet to be implemented.

**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 yet to be implemented.

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

We assume that another user can be a staff member.

This functionality has yet to be 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.

This functionality has yet to be implemented.

**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 has yet to be 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 functionality has yet to be 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 yet to be implemented.

**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 yet to be implemented.

**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 yet to be implemented.

**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 as expected, the notifications are sent to the user via email, mentions and replies are also notified to the user on the front-end server.

**2.5. 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.

Although the LDAP was configured for the application, using the LDAP 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 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.

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 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.

# Development Methodology

During this project, selecting a development methodology to follow was a challenge as there were only two people working on the project.

We pursued a Scrum methodology whilst adopting small components from Extreme Programming, the main component we adopted was pair programming, this was used primarily during integration steps of the front-end system.

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 microservices as sprints, each sprint would consist of usually 1 week. 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.

# design

# Implementation

In this section, each sub-section is broken down in to two parts (unless state otherwise), these parts are the backend configuration, and the front-end 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 frontend interface design as well as the functionality behind the Identity database, the webseries guide presented by “Kudvenkat”[5] of Pragimtech[6] 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.

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.

The frontend 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 frontend 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 to handle this for us, but due to technical difficulties we couldn’t get LDAP 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 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**

Not yet implemented, talk about both the backend and frontend implementation, I don’t know enough about it to write the section.

**5.4. SolR**

Not yet implemented, talk about the backend implementation -> If frontend is implemented, mention that, else omit it and state that the front end is not implemented.

**5.5. Notifications**

Talk about the backend implementation. (This should be much longer than the frontend)

On the frontend 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).

# 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 <INSERT>.

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 frontend server, if we erroneous operations here, it is safe to say the frontend 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. In the test plan we talk about Moq tests for this section, but MSTests were employed.

Integration tests were primarily applied through Apache JMeter [4] using JMX tests, these tests allowed us to test the HTTP requests that are exposed from the database, this is because the frontend 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 frontend would be down to the implementation of the frontend service, and not the database. This allows us to work on the frontend service disjointly from the database by giving us confidence in the backend.

For functional testing we exploratively 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 frontend 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 exploratively 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.

# Status

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. This is considered erroneous user activity and we would not expect it to happen in general as it would require a user to know exact URLs.

# critical Evaluation

REFERENCES

|  |
| --- |
| [1] Blackboard, Inc (2019). Blackboard website (Online). https://uk.blackboard.com. (Accessed 10th January 2020)  [2] Solr, Apache. Apache website (Online). https://lucene.apache.org/solr/. (Accessed 10th January 2020)  [3] Gruber, J. (2019) Markdown (Online) https://daringfireball.net/projects/markdown/ (Accessed 16th January 2020)  [4] JMeter, Apache. Apache website (Online). https://jmeter.apache.org/. (Accessed 16th January 2020)  [5] YouTube, Kudvenkat Asp.net core tutorial (Online). https://www.youtube.com/channel/UCCTVrRB5KpIiK6V2GGVsR1Q. (Accessed 16th January 2020)  [6] Pragimtech (Online). https://www.pragimtech.com/. (Accessed 16th 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 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |