L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Computer Information Systems

NEMESYS
A Near Miss Exposure and Reporting System

Stefano Schembri (29396M), Liam Curmi de Gray (345495M)
B.Sc. (Hons) Software Development

Study-unit: **Web Applications Architecture and Systems Development**
Code: **CIS2055**
Lecturer: **Dr Chris Porter**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I / We*, the undersigned, declare that the assignment submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Stefano	Schembri S
Student Name	Signature
Liam Curmi de Gray	[illegible]
Student Name	Signature
Student Name	Signature
Student Name	Signature

CIS2055	Nemesys - Assignment Stefano and Liam 2021
Course Code	Title of work submitted

18/06/2021

Date

Table of Contents

Introduction

The task assigned for this coursework was that of creating a web app using ASP .NET Core MVC, SQL Server and Entity Framework from scratch to understand how these technologies work and what benefits they provide. The general problem case given was that of a near-miss reporting system which allows for users to create reports to highlight hazards on the UoM campus and have these reports investigated and taken care of.

This project is a product of a two-person team who is responsible for the design, implementation, and testing of said project from the ground up. In this documentation we shall be discussing the process of developing this web app, the architecture of the app and the best practices used for the development of this web app, which include code related best practices as well as broader project best practices. We will also be elaborating on the database used for this project and how certain parts of the database structure had to change as the project developed, as well as external libraries used, user management flows with respect to the web app UI design, use of roles and the challenges which were faced when developing this system.

After this, we discuss the testing strategies deployed in which we explain how the system was tested for robustness and functionality and finally a section is dedicated for future possible improvements on the system.

Description of Architecture

As mentioned above, this project was developed using the ASP .NET Core MVC architecture. This type of architecture splits the project into three main parts: Models, Views and Controllers.

The way we went about creating the core structure of the app is by first designing and implementing a first draft for a database. We then used the ASP .NET MVC scaffolding tool to automatically generate the classes for the models, views, and controllers. The scaffolding feature creates the basic backbone structure for all the main entities in the project. This resulted in several classes being automatically generated which, at the end, were not used.

All the core models were used in order to generate the controllers and views for the specific model. Each model had its own controller and respective pages, namely index, create, details, edit and delete pages.

The models were always used to create an instance of the object and save it in the database; however, these models were accompanied by additional classes called ViewModels. These ViewModels were used to create an instance of a particular model for a specific page or pages. For example, the model of a report might have the normal report attributes however a ViewModel for the details page of the reports also needs an instance of the Investigation model in order to display the investigation details if there is an investigation open.

The concept of ViewModels was used rigorously throughout the development of this project and multiple different ViewModels were created to cater for different Reports and Investigations pages and the Hall-of-Fame page. Some examples of ViewModels created for this project include:

- CreateInvestigationViewModel/ CreateReportViewModel
- InvestigationViewModel/ ReportViewModel/ HallofFameViewModel
    - These three ViewModels were then added to a list inside three other view models in order to display all of the instances
- StatusCategoryViewModel

These ViewModels still create an object of the base model however they allowed us to work with different attributes for different scenarios.

In terms of controllers for this project, there were three main ones which provided us all the functionality needed. The three controllers are the Reports, Investigations and Hall-of-fame controllers. The reports and investigations controller each have their own index, details and delete methods as well as the create and edit methods, each having a GET and a POST method. All these methods were used to deliver the functionality needed for this project.

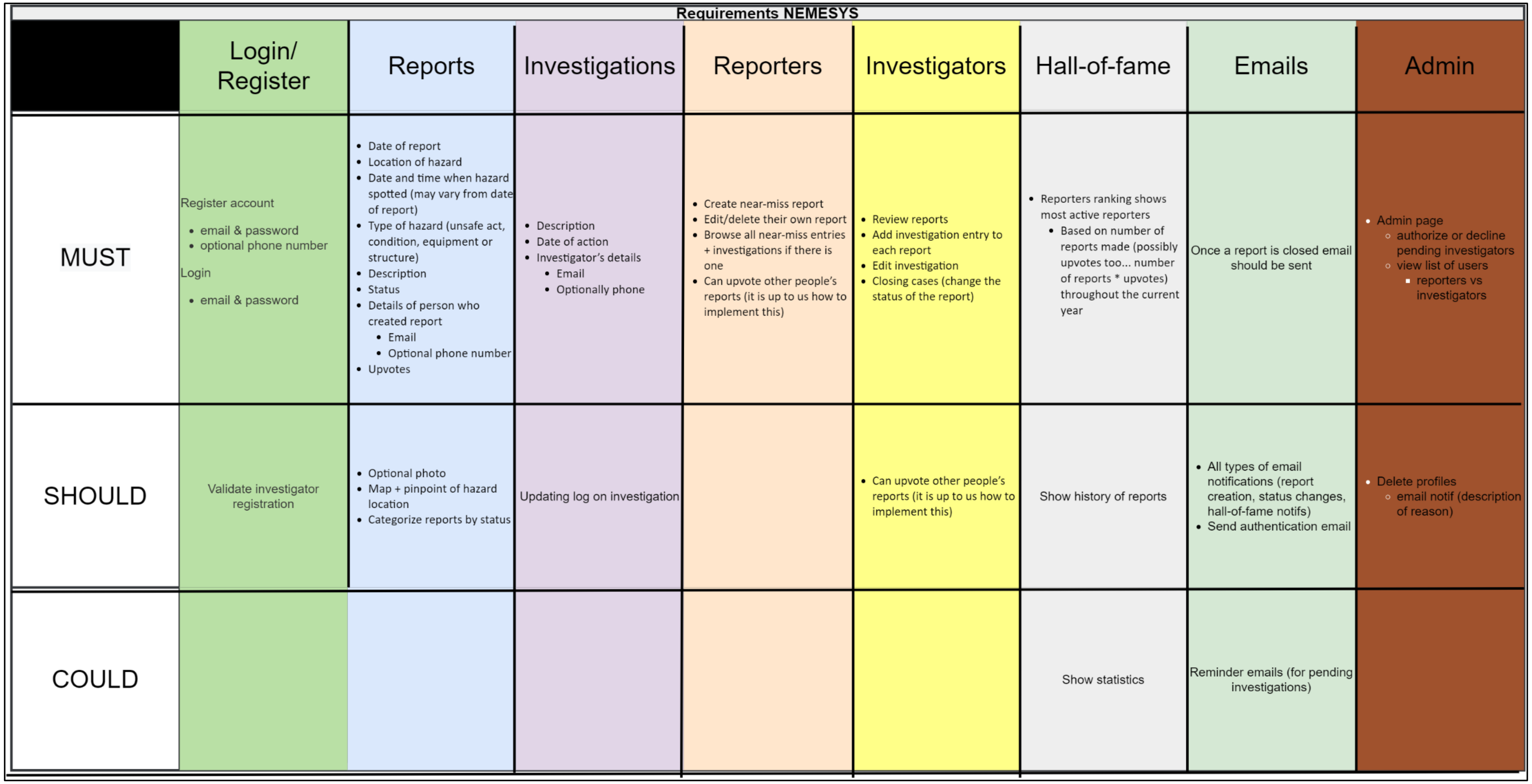
The Hall-of-fame controller was modified to only make use of the index method as this page is only tasked with displaying the top reporters and redirecting to different pages.

In terms of views, the scaffolded Razer views were kept and for the Reports and Investigations part, all the pages were used. Only the delete page, however, was not modified and the other four, namely the index, create, edit, and detail pages were all changed to make use of the respective ViewModels. The reports and hall-of-fame index pages were also modified to make use of the "card" layout as demonstrated during this units' lectures.

Design

The first step that was taken by the team was to hold a meeting and discuss the problem thoroughly while taking down as many details as possible. We then set up another meeting before which we researched the problem area to have a better understanding of the problem at hand and how to go about developing a solution.

After the problem was discussed extensively, we developed a MoSCoW table to have a clear understanding of what requirements we should tackle, and their importance in the context of this assignment.



Requirements NEMESYS

	Login/ Register	Reports	Investigations	Reporters	Investigators	Hall-of-fame	Emails	Admin
MUST	Register account • email & password • optional phone number Login • email & password	• Date of report • Location of hazard • Date and time when hazard spotted (may vary from date of report) • Type of hazard (unsafe act, condition, equipment or structure) • Description • Status • Details of person who created report • Email • Optional phone number • Upvotes	• Description • Date of action • Investigator's details • Email • Optionally phone	• Create near-miss report • Edit/delete their own report • Browse all near-miss entries + investigations if there is one • Can upvote other people's reports (it is up to us how to implement this)	• Review reports • Add investigation entry to each report • Edit investigation • Closing cases (change the status of the report)	• Reporters ranking shows most active reporters • Based on number of reports made (possibly upvotes too... number of reports * upvotes) throughout the current year	Once a report is closed email should be sent	• Admin page ◦ authorize or decline pending investigators ◦ view list of users ▪ reporters vs investigators
SHOULD	Validate investigator registration	• Optional photo • Map + pinpoint of hazard location • Categorize reports by status	Updating log on investigation		• Can upvote other people's reports (it is up to us how to implement this)	Show history of reports	• All types of email notifications (report creation, status changes, hall-of-fame notifs) • Send authentication email	• Delete profiles ◦ email notif (description of reason)
COULD						Show statistics	Reminder emails (for pending investigations)	

Figure 1

Use case diagrams were also drawn up to help us brainstorm different scenarios that can occur while using this kind of system. This allowed us to have a better understanding of how the basic structure of the system would be in terms of functionality and how different entities may interact with each other.

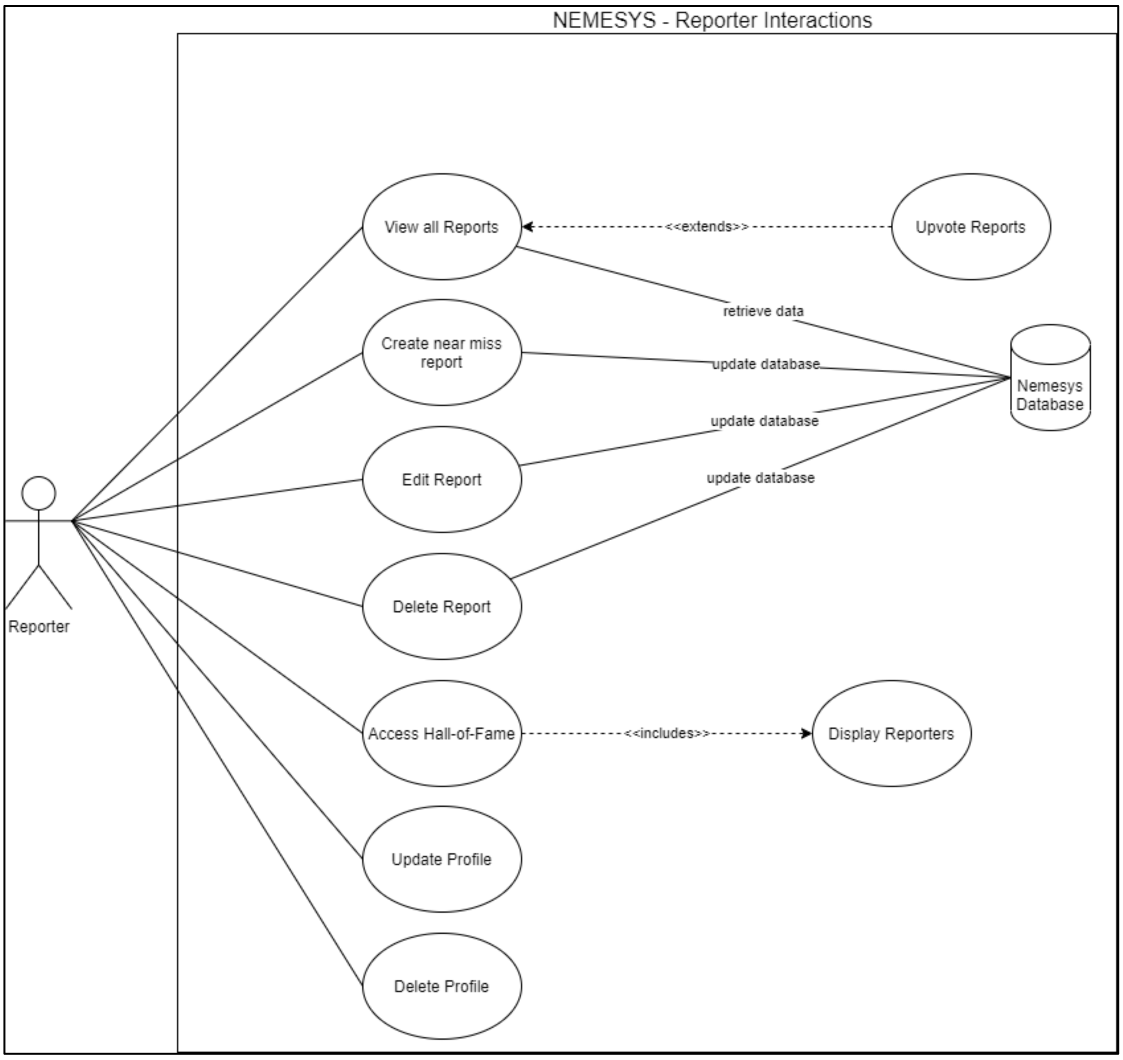


Figure 2

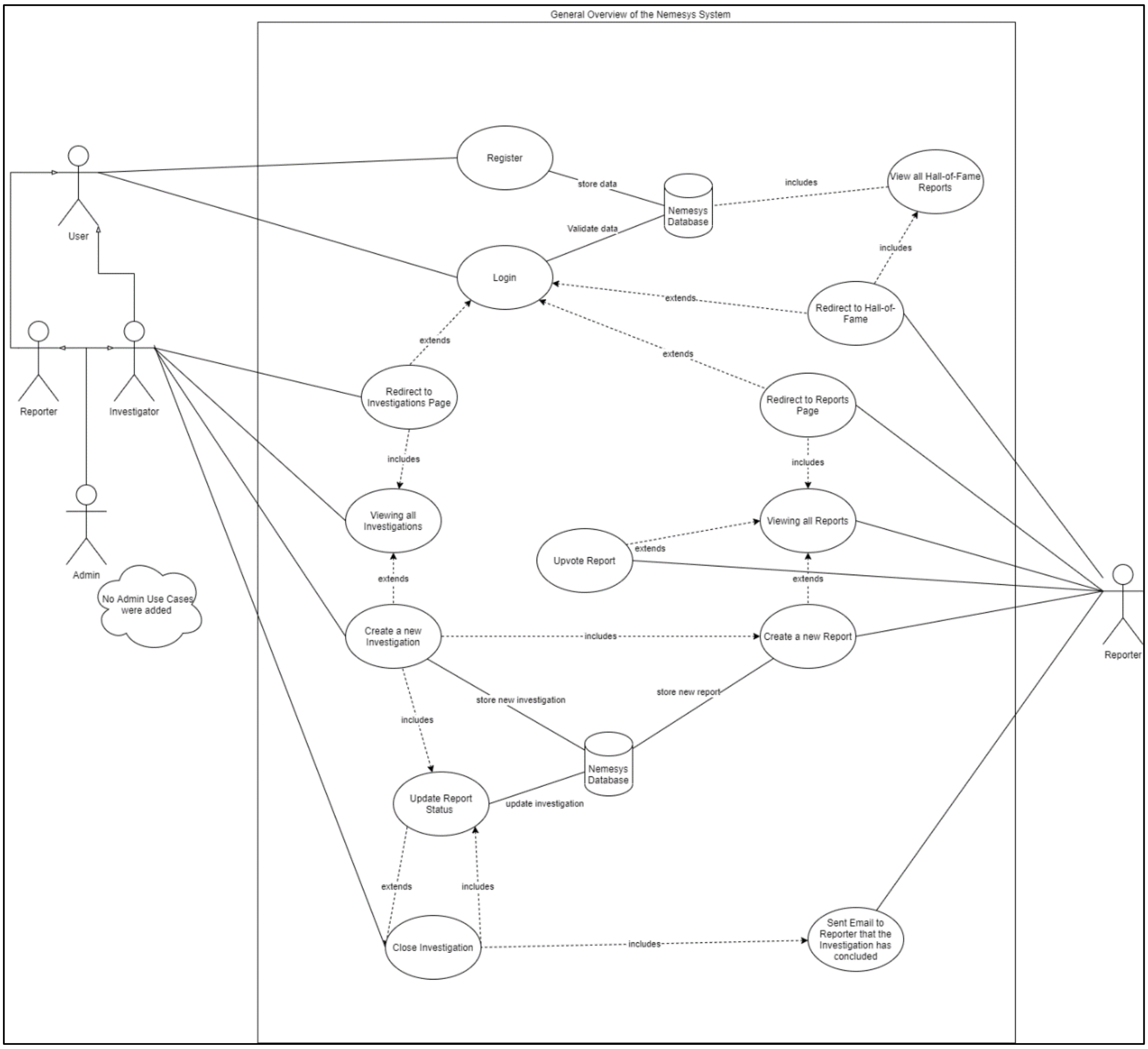


Figure 3

These diagrams were only used as a guide and some structure and functionality ended up being modified in some way or another.

The use of prototypes also helped in understanding what data to expect from the different users and model out project accordingly. Below you can find prototype pages for the login/registration, creating a report and hall-of-fame. It is important to note that these are early prototypes and that the final design of the pages ended up being quite different, however they helped a great deal in understanding what data to include in these pages.

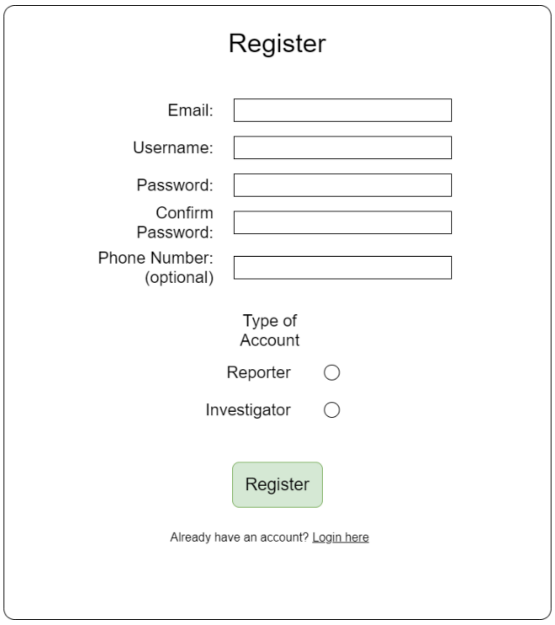


Figure 5

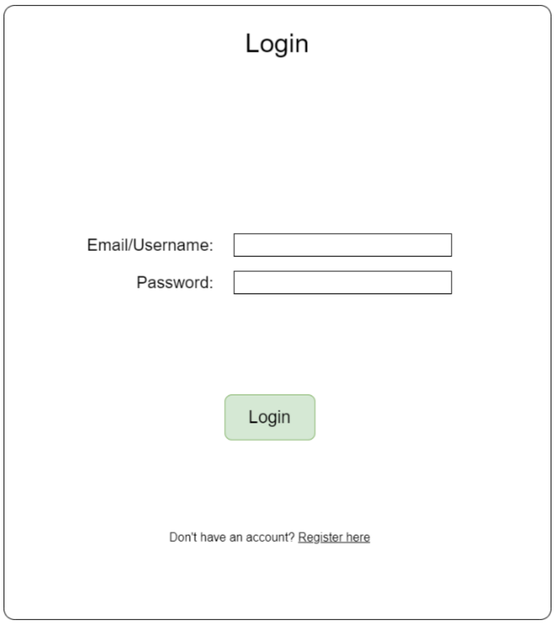


Figure 4

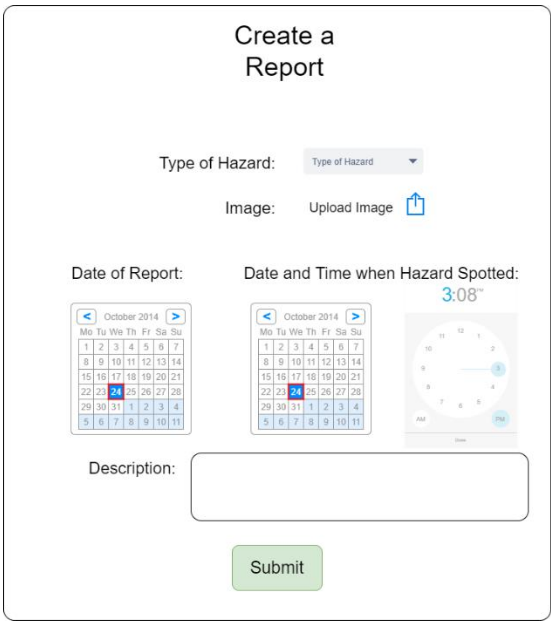


Figure 7

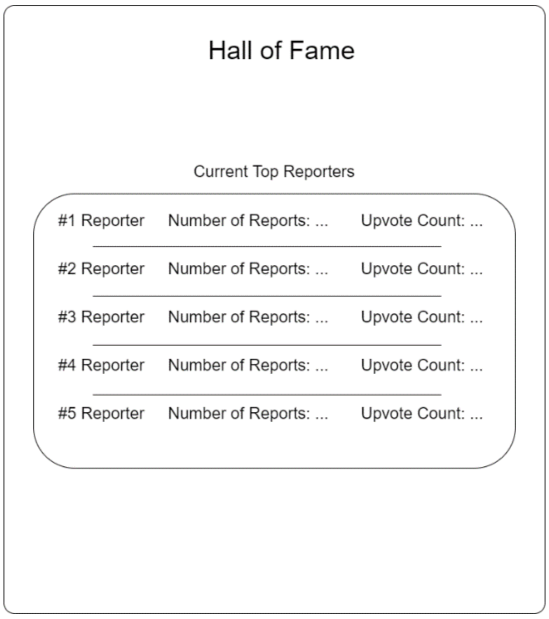

Hall of Fame
Current Top Reporters
#1 Reporter Number of Reports: ... Upvote Count: ...
#2 Reporter Number of Reports: ... Upvote Count: ...
#3 Reporter Number of Reports: ... Upvote Count: ...
#4 Reporter Number of Reports: ... Upvote Count: ...
#5 Reporter Number of Reports: ... Upvote Count: ...

Figure 6

The next step was to create a database diagram. This is an important step in any project as the design of the database heavily affects the structure of the deliverable. Database structure also affects functionality and determines how entities interact with each other and how the data flows within the system. Again, the below is an entity-relationship diagram of a schema which was developed early in the project lifecycle and not the end database schema. However, the core structure was maintained, and it still served as a valuable reference when developing the database, especially from a code-first approach.

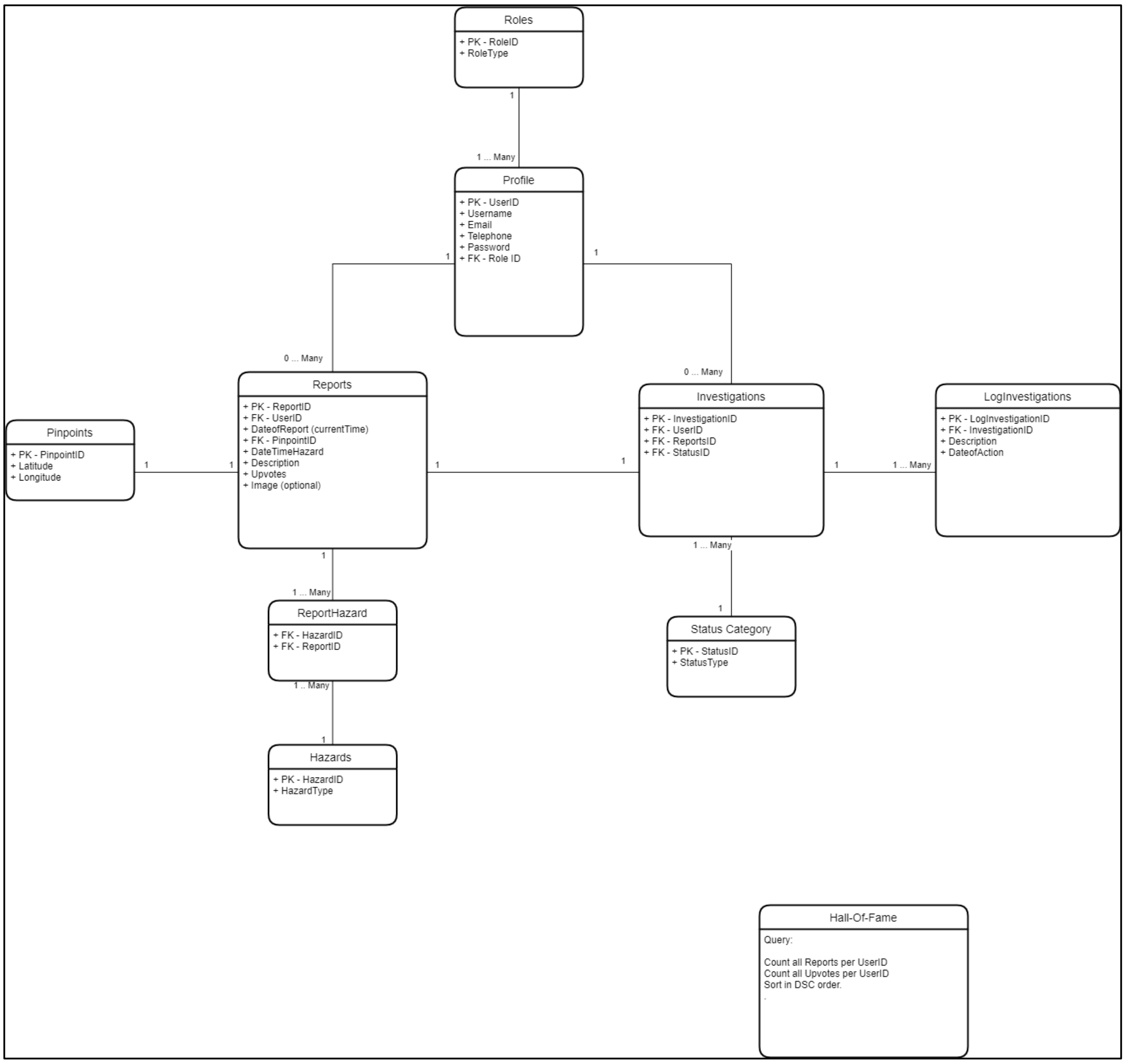


Figure 8

Database Schema

This section will comprise of the DB initialiser, first thoughts on how the database schema should look like, what became the final database schema and lastly the differences between the initial ER Diagram (Figure 8) as mentioned in the previous section compared to the Final ER Diagram.

DB Initialiser

With the addition of a Code – First approach and ASP.NET Core Identity, the DB Initializer class was created. The purpose of this class is to populate the database with some initial data upon running the solution for the first time without having a database such as:

- Role:
  - Role Name: [“Admin”, “Reporter”, “Investigator”]

- User:
  - Username: : [“admin@gmail.com” , “reporter@gmail.com”, etc...]
  - Email: [“admin@gmail.com” , “reporter@gmail.com”, etc…]
  - Password: [“Admin123!”, “Reporter123!”, “Investigator123!”]
  - Author: [“Admin”, “Reporter”, “Investigator”]
  - Phone number (an optional field) : [21222222 , 21333333 , 21444444]

- Status Categories:
  - StatusType: [“Open”, “Being Investigated”, “Closed”, etc…]
- Hazards:
  - HazardType:[“Structure”, “Fire”, “Electrical”, etc…]

Final ER Diagram

Figure 9 as shown below, shows the final ER diagram that became part of the solution:

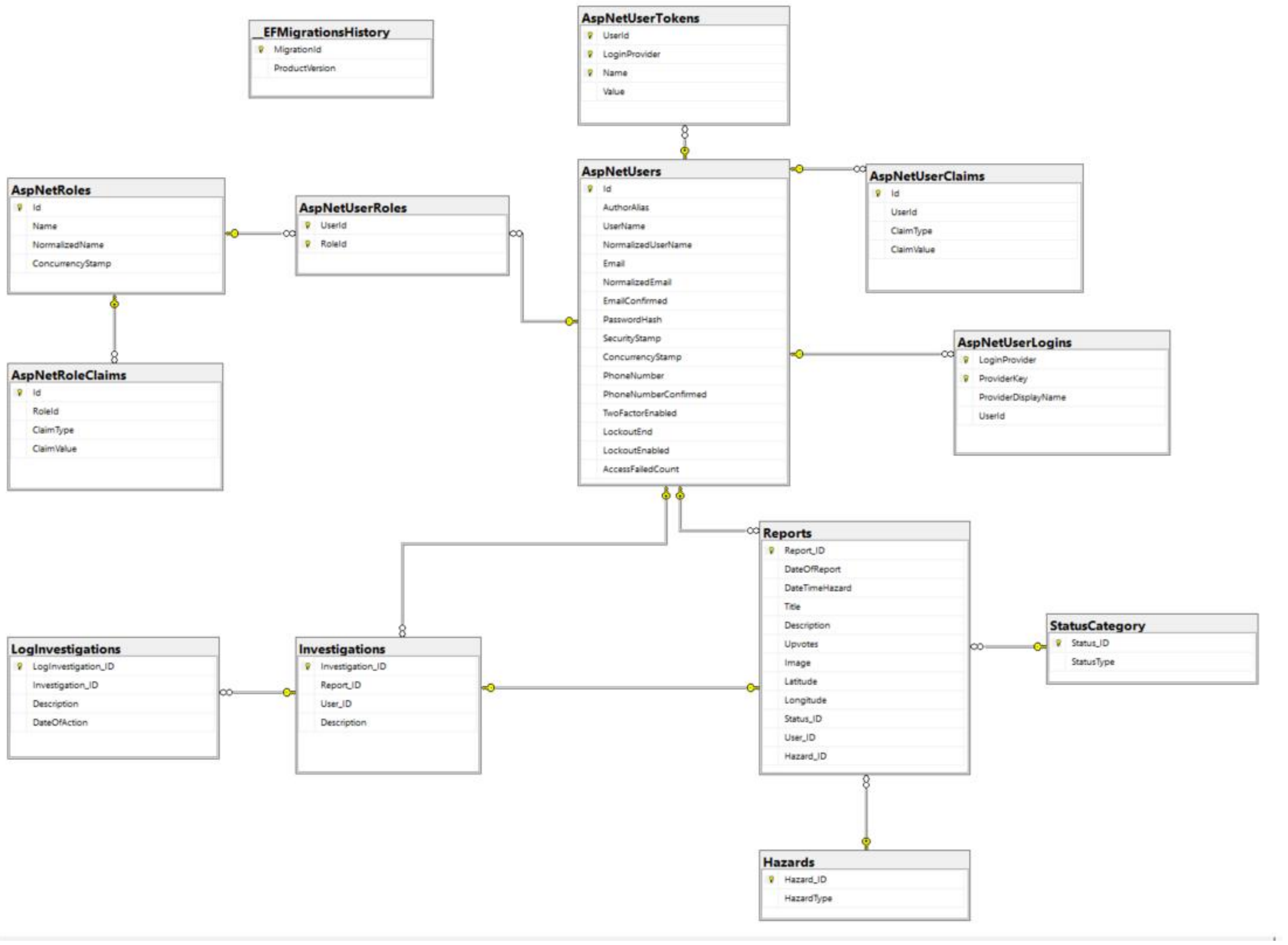


Figure 9

Differences between initial ER and final ER diagram

User & Roles

Initially as mentioned previously in this documentation, the idea of using ASP.NET Core Identity was not considered, however by the end of the development it became the primary tables to store Users and Roles. Between Figure 8 & 9 in terms of fields we received additional Identity field including an Author field which served as the nickname that is displayed on the web application.

Reports

From Figure 9, Reports table had the additional fields of Title which served as the heading of a report. The Pinpoints table was dropped, and the Latitude and Longitude fields were added directly on Reports as throughout development, this made more sense. StatusID foreign key from Investigation was moved to report, to introduce when a report is initially created it would have the default “Open” status for investigations to be commenced on.

Investigations

Investigation table included and additional field Description, which marked as the heading for an investigation.

Log Investigations

One of the only tables that did not change from initial design. The purpose for a Log Investigation is to add new logs to ongoing investigations.

Hall of Fame

As noted from Figure 8, there was a slight debate whether a Hall-Of-Fame table was required. By the end of development, it was noted this was not required as querying the Reports table proved sufficient enough to display on the web app.

Report Hazard

From Figure 8, the purpose for this table was to include a many-to-many relationship between Reports & Hazards table. This was thought to allow multiple hazard types can be marked on the Report. However, due to time constraints this table was dropped and instead was made a one-to-many relation between Reports & Hazards.

Hazards & Status Categories

These tables in the final schema have remained the same from the initial design, the only changes made were the foreign key constraints to different tables as mentioned in the previous sub-sections.

Best Practices

It was clear from the star that this project relies heavily on the concept of best practice when it comes to developing solutions. This was kept in mind throughout the development lifecycle of this project and is reflected in the work put in for the design stage, in the best practice techniques used in the code itself, and in the tools and techniques used for managing the project development.

Trello

Since this project was not a small task and was somewhat labour-intensive, we wanted to be sure that the handling and tracking of tasks was done properly. To handle task division for a big project, one needs the necessary tools for the job.

An excellent tool for this is Trello. Trello is a web app which helps users to create boards for their project and create and keep track of tasks for the project by adding cards to the board. The cards can be assigned to different members and are organised in different sections and can be moved from one section to the other. A due date can also be added to each card. This format helped us to visualise what tasks we needed to do and how much work was done or still needed to be done.

Trello proved to be a very important part of this project and helped keep things running smoothly even while doing development sprints.

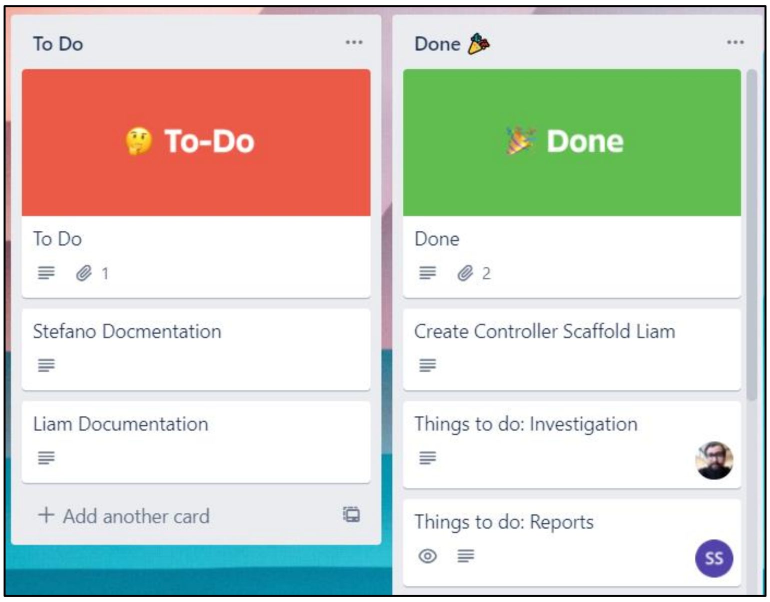


Figure 10

GitHub

GitHub is another tool used for this project which falls under best practices. It is a tool all developers use and was a requirement for this project.

GitHub allows for a repository to be created which holds the project solution including all code and relevant files. It works by allowing each contributor to have a copy of the repository on their local machine and make changes to it.

These changes stay on the local machine until the developer chooses to push their changes to the repository. Branches can be created to safely modify code and have a fall-back option if the modified code does not work as expected.

Once a branch is opened and changes are pushed onto it, this branch can then be merged with the main branch by creating a pull request. Pull requests are requests made by a developer when they wish to merge their new branch with the main branch of the repository. A repository may be set up to have security checks, meaning pull requests cannot be merged onto the main branch without approval of other developers of the same repository. Once the pull request is approved, the new branch is merged with the main branch which updates the project repository, meaning the main code and files are updated to that version.

GitHub was an essential tool in developing this project as it provided us with a safe and organised way to collaborate with each other and merge our individual work together. The GitHub repository can be found here: https://github.com/LiamCurmideGray/cis2055-nemesys

Entity Framework Core

Entity framework Core is a light-weight open source from the Entity Framework data access technology. It allows us to connect with our respective SQL Server and Database and able to retrieve, populate, update, and delete to our database connected based on the models created on our solution. Utilizing this proper framework allows to create migrations to adjust and modify the schema of our database

ASP.NET Core Identity

ASP.NET Core Identity is a full-featured membership system that allows for creating and maintaining users. With the Identity it enabled us to register , login and modify profiles which were automatically built into our solution with very little coding development required. In conjunction with the entity framework core, it additionally allowed us to connect to the models created on our solution to the identity classes via the User Manager which handles all user related data such as creating (which automatically hashes the password assigned), updating user, also enables us to assign roles to users. Identity also manages the Sign-In Manager which is responsible for login and logout from of the user.

The inclusion of Identity allowed for easier validation when interacting with the controllers created to assign Authorizations tags to allow only specific user assigned roles to interact with specific controllers, such as: Only Users that have a Role type “Reporter” can create reports. Including the additional validation amongst same role types, such as: Only the author of the Report can edit or delete their own report, other reporters are only able to view the details of the report.

Scaffolding

One of the many benefits of Entity Framework is the ability to scaffold files based on the models created and the context file established. This allows us to generate the minimum code required to have a basic functioning web app. Entity Framework scaffold generates the controllers' and respective views of the models. In addition, with ASP.NET Core Identity it also allows us to generate the Identity files which are the user files to incorporate identity in the web app, such as the creation of the register, login, logout, a partial view that gets attached to the layout, etc…

Repositories Pattern

Throughout the initial development we had not considered to incorporate a repository pattern. However, as development went on the controllers set up were handling all the logical operations to the entity framework core. This became very cluttered and unclean code as the operations between each controller method almost copied from each other. Incorporating a Repository Pattern allows the data being requested to be mediated to and from in the Domain and Data Access Layers. Which allows repository classes to be hide their logic required to store or retrieve data. It also allows for a cleaner separation of concerns and used to build cleaner solutions as show in Figure 11.

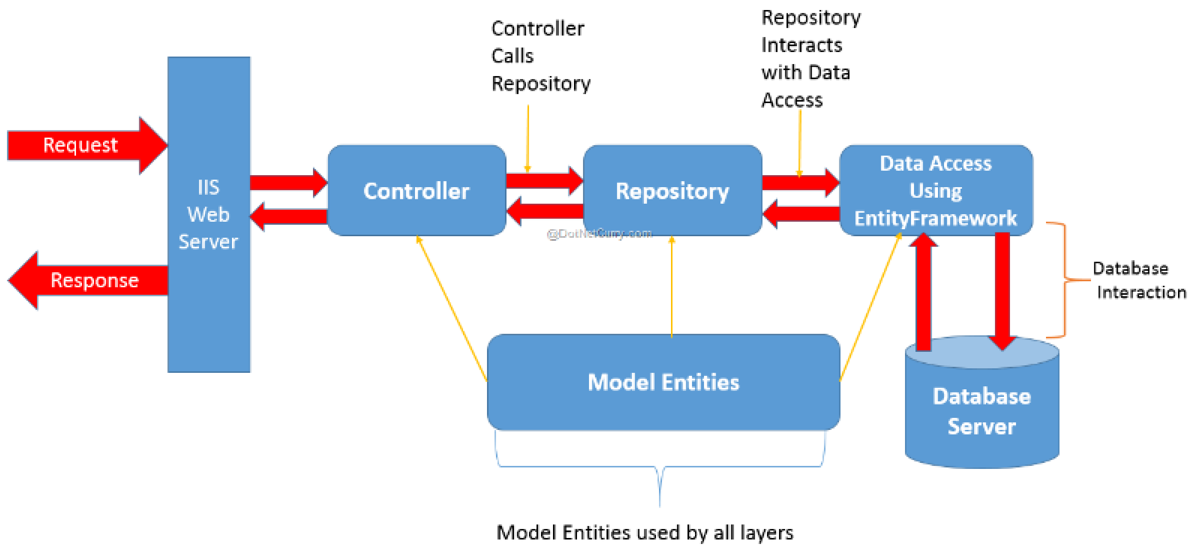
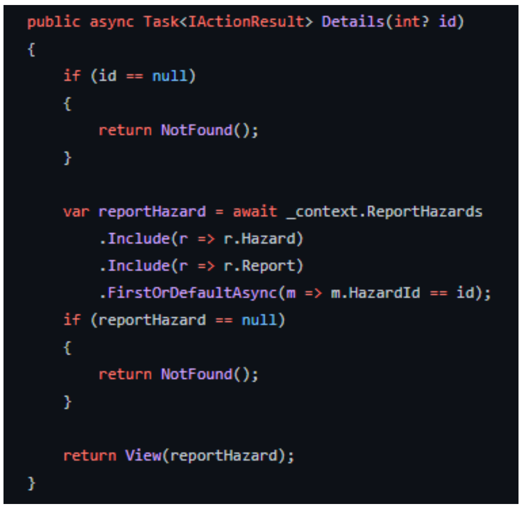

Controller Calls Repository
Repository Interacts with Data Access
Request
Response
IIS Web Server
Controller
Repository
Data Access Using EntityFramework
Database Interaction
Model Entities
Database Server
Model Entities used by all layers
@DotNetCurry.com

Figure 11

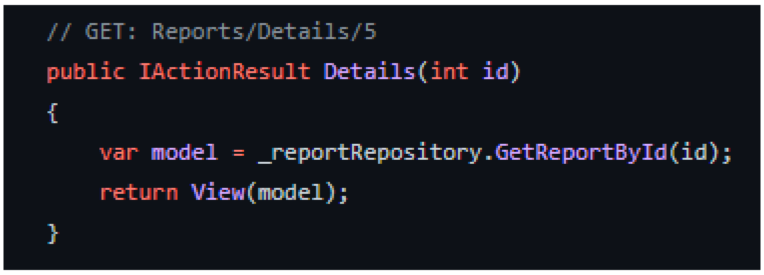
An example to show the differences in the code solution can be found in Figure 13, the figure shows how the Reports Details method after created through scaffolding with from the model. After refactoring and incorporating the report repository, this is how the new Report Details method look like as shown in Figure 12.



```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var reportHazard = await _context.ReportHazards
        .Include(r => r.Hazard)
        .Include(r => r.Report)
        .FirstOrDefaultAsync(m => m.HazardId == id);
    if (reportHazard == null)
    {
        return NotFound();
    }

    return View(reportHazard);
}
```



```
// GET: Reports/Details/5
public IActionResult Details(int id)
{
    var model = _reportRepository.GetReportById(id);
    return View(model);
}
```

Figure 12

Figure 13

As noted by the differences between Figure 12 & 13, Figure 13 functions as the same method as Figure 12 with a much cleaner and less code. This is due to creating the repository and focusing the logic and data interaction with the database on the repository instead on the controller.

Code-first approach

Incorporating a Code – First approach , just like the repositories were incorporated to the solution midway through the development of the solution. This was due that in the beginning the approach of Data - First approach felt more appropriate to tackle this assignment due to needed to first knowingly know how the database architecture was required to manipulate the data as seemed fit. The Code – First approach was adopted after the inclusion of the ASP.NET Identity was required to be implemented. This shift was required to integrate the models of the Identity which the models created with our database as this was not possible to do so from a Data – First approach.

Shifting to Code-First had its initial challenges to be adopted, however we soon realised that due to building from Entity Framework approach as well including incorporating the Repositories as a useful Domain Driven Design we came to realise most of the groundwork was already established. The only missing piece from the development was the inclusion of Migrations to update the database. However, the old Database Schema from the Data – First approach had to be deleted to correctly allow for the database to be updated fully from migrations, however it did not deviate too far from the initial schema that we intended on.

The additions of Migrations allowed us easier to update the database whenever we required to modify some of the existing models, such as accidentally forgetting to add a Title field for Reports', but which was quickly patched.

External Libraries

A project of this size will often make use of external libraries. Apart from the many libraries included with the ASP .NET MVC package, it was also decided that we should implement the Google Maps API service.

The Google Maps API service allows for developers to implement Google Maps into their web apps by creating an API key and using this key to include the Google Maps Javascript source file.

A project was set up on the Google Cloud Developer Console and an API key was generated. This key was then used to establish a connection with Google and request the Maps service.

Google Maps was used in three different pages in the web app; the create report page, where the Google Maps box displays a map and can also be clicked to place a marker highlighting where the hazard was spotted, the edit report page with has the same functionality as the create page and the report details page where a map is shown with the marker placed representing where the hazard was spotted which cannot be changed or removed.

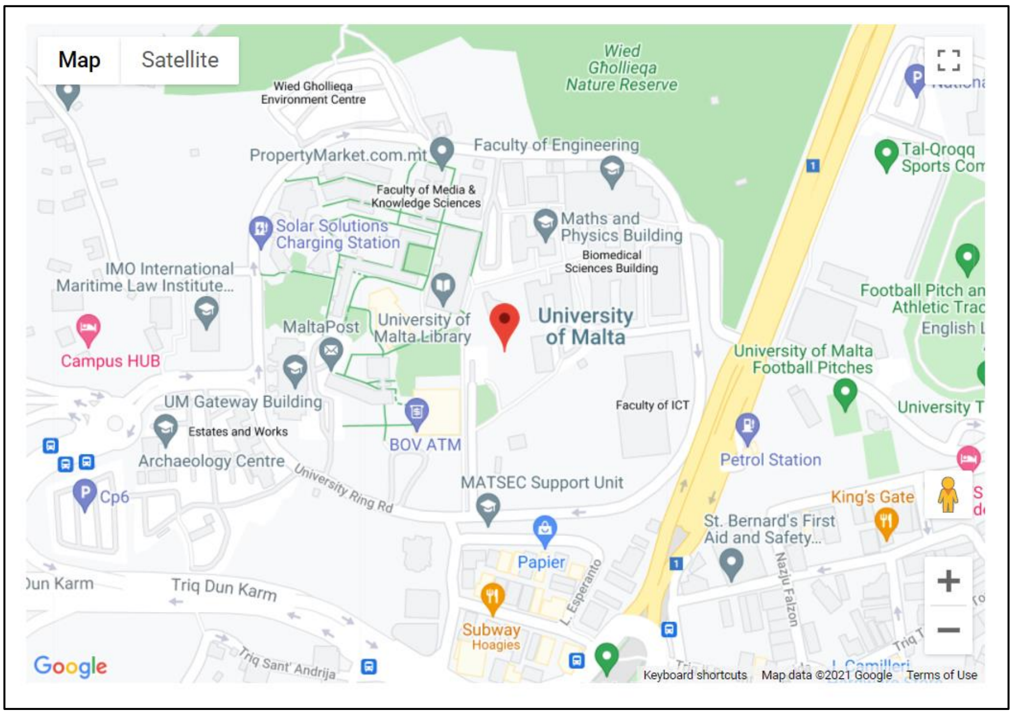


Figure 14

User Management Flows

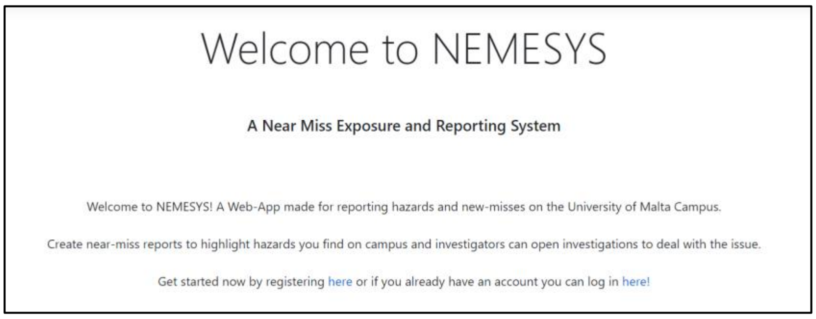
This section here will preview as a walkthrough throughout the web application as the perspective of a user. We will be highlighting the different views made from a non-authenticated & is-authenticated person with the additional different views and interactions of a Reporter or Investigator accordingly.

Non-Authenticated:

The following sub-sections highlight when a user is not logged in. What features and pages they can interact with.

1. Homepage

When initially running the application for the first time, the user is presented to the Homepage of the web application. Here the user has only a few limited options, such as displaying the contents of the Homepage and interacting with the navigation bar.



Welcome to NEMESYS

A Near Miss Exposure and Reporting System

Welcome to NEMESYS! A Web-App made for reporting hazards and new-misses on the University of Malta Campus.

Create near-miss reports to highlight hazards you find on campus and investigators can open investigations to deal with the issue.

Get started now by registering here or if you already have an account you can log in here!

Figure 15

2. Navigation Bar

From the navigation bar all references made according to Figure 15. The options here are limited, firstly on the top right there is the **Nemesys Home** button which would redirect to the home page. The on the left-hand side there are the **Register** and **Login** buttons which would redirect the user to the register and login pages accordingly.

Nemesys Home
Register
Login

Figure 16

3. Register

On the Register Page, the user can insert data in the respective fields to register themselves. This user data is part of an object of **NemesysUser** which inherits from **IdentityUser.** This data stores a new record of type user and the user can select which Role they wish to register as. Only an admin with access to the database directly can assign a different Role to a user.

4. Login

On the Login page, a user who had successfully registered themselves on the database are able to login to the website after filling in the respective forms. Validation is added within the forms if a login is incorrect, or fields were not filled. Once logged in the user will be redirected back to the Homepage.

To try out the login yourself, there are 4 users' options that can be used to test on. There is a user for Role type **Reporter & Investigator** as noted by Table 1:

	Reporter	**Investigator**
Email	reporter@gmail.com	investigator@gmail.com
Password	Reporter123!	Investigator123!
Email	reporter2@gmail.com	investigator2@gmail.com
Password	Reporter123!	Investigator123!

Table 1

Is-Authenticated

The following sub-sections will highlight the pages and interactions accessible once a user has logged in regardless of what Role type, they are.

1. Navigation Bar

Once logged in the navigation bar gets updated with the new following redirects. **Investigations**, **Reports**, **Hall of Fame buttons** would redirect you to the Investigations Index page, the Reports Index page, Hall of Fame page respectively. Lastly on the left hand side of the Navigation bar, the Register button has been switched with Hello user.author, here the user would be redirected to their profile page. The login button has been switched with the log out button which once pressed would log them out and redirect the user back to **Non-Authenticated Homepage.** As shown in Figure 16:

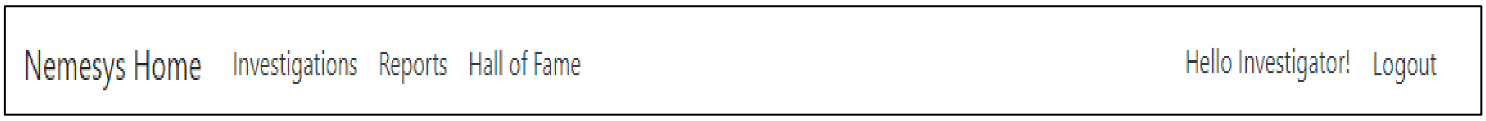


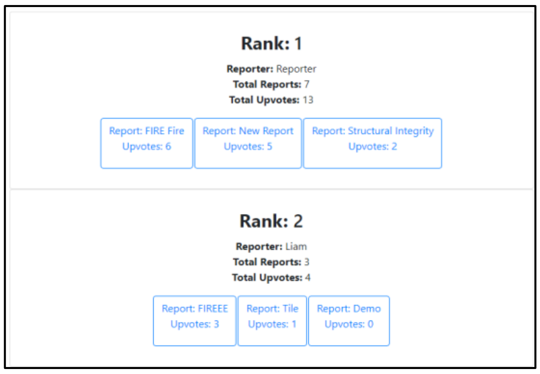
Figure 17

2. User Profile page

Once in the Users' Profile page, there is a range of options here. Most notably the first page retrieved would be the ability to change the authors name and phone number should they wish. On the side is a side bar would redirects the user the following: **Profile** is the view just mentioned, **Email** allows the user to change their email address**, Password** allows the user to change their password.

3. Hall of Fame

Hall of Fame page shows the ranking of the top 3 reporters based on the number of reports they have submitted. Each slot shows the rank of the Reporter, their author's name, total reports they have contributed and the total combined upvotes they have ever received. In addition, highlight the top 3 most upvoted reports per reporter. As shown in Figure 17:



Rank: 1

Reporter: Reporter
Total Reports: 7
Total Upvotes: 13

Report: FIRE Fire
Upvotes: 6

Report: New Report
Upvotes: 5

Report: Structural Integrity
Upvotes: 2

Rank: 2

Reporter: Liam
Total Reports: 3
Total Upvotes: 4

Report: FIREEE
Upvotes: 3

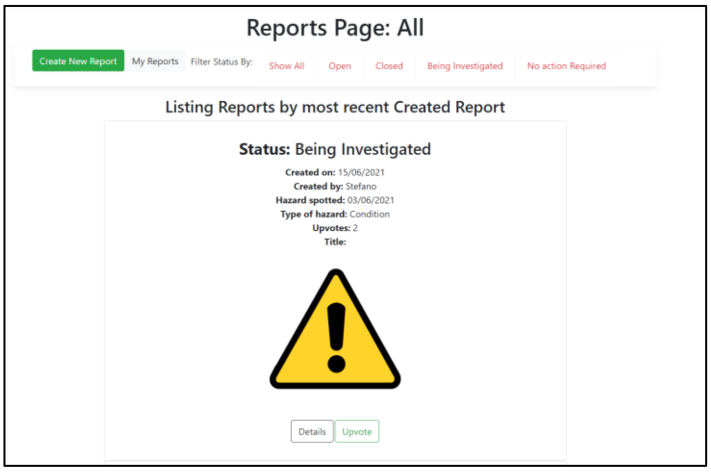
Report: Tile
Upvotes: 1

Report: Demo
Upvotes: 0

Figure 18

4. *Reports Index*

Any user logged in can view the reports page. The main purpose here is to give a brief overview of what each report consists of, having their own small respective details. The user here can filter the reports according to what status they currently are and is able to filter the reports accordingly to what type of status they are. All reports here shown are ordered by the most recent report created. More features are expanded on this page according to what Role they are.



Listing Reports by most recent Created Report

Status: Being Investigated

Created on: 15/06/2021
Created by: Stefano
Hazard spotted: 03/06/2021
Type of hazard: Condition
Upvotes: 2
Title:

Figure 19

5. *Report Details*

Reports Details page highlights all the details related to said report including the description, image, the coordinates marked on the map and if an active investigation is ongoing or not by the display of the Investigation Title. At the very bottom left of the page there is a redirection button to take the user back to the reports index page.

6. *Investigations Index*

The investigations index page highlights a list view of all active investigations currently ongoing. The user also has an option to be redirected to the details of the investigation.

7. *Investigation Details*

The investigation details highlight a list of the ongoing investigation of the report including a list of all logs added on to the report of the same investigation. At the bottom left of the page, the user can redirect themselves to the report details of the ongoing investigation and is also able to be redirected back to the investigation index page.

Reporter

The following sub-sections here will highlight the user that is logged in of Role type **Reporter.** All mentions from the previous section **Is-Authenticated** are inherited here.

1. *Reports Index*

The additional features for a Reporter to be on here is the inclusion of extra redirect on the sub-navigation bar next to the change **Filter Status by.** The additions here are: **Create New Report button** which would redirect the reported to create a new report page and **My Reports** which would return the same index page however only showcasing the reports made by the current reporter logged in.

Within each report card, if report was created by them, the web app will also view 2 additional buttons next to the **Details button**, on it's left would be and **Edit button** to edit the respective report and to the right a **Delete button** in order to delete reports.

Reporters are able to view other reports made by other reporters however they are not able to edit or delete these reports. Instead, they can upvote the report next to the **Details Button** which would increment the upvotes counter of the report selected.

2. Report Details
If a reporter views a report details created by themselves, the additional feature here is that at the bottom left of the page. They will also find an **Edit button** that would redirect them to the edit page.

3. Create Report
A reporter has the capability of creating a new report, in this view they are able to fill out the required form details including the additional options of uploading an image (only jpegs or pngs allowed) and marking on a Google map where the hazard was located. Upon submission the report would be viewable on the report index page.

4. Edit Report
A reporter is able to edit the details of a report already creating, they view here would look almost the same as the Create Report page except all the fields are filled already including the marker on the map. A user should they wish upload a new image, the old image gets deleted from the solution.

5. Delete Report
A reporter is able to delete their own report. They are first redirected to the delete page and is shown the current contents of the report. Then the reporter can click the **Delete button** again to fully delete the report and redirect themselves back to the Reports Index page.

Investigator
The following sub-sections here will highlight the user that is logged in of Role type **Investigator.** All mentions from the previous section **Is-Authenticated** are inherited here.

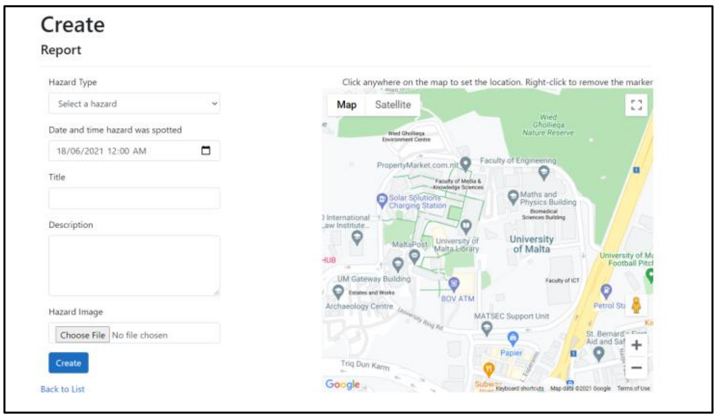
1. Report Details
Whilst being able to view the Reports detail, if a report has no active investigation on going then at bottom left of the screen, The investigator is able to create a new Investigation and be redirected to the Create Investigation page.

2. Investigation Index
Investigators who have ongoing investigations under their name will notice new features accessible to them. On each side of the **Details button** they will find an **Add Log button** which would redirect the investigator to the **Edit Investigation** page. Thy are also able to redirect themselves to the Delete page once if they click on the **Delete button.**

3. Investigation Details
Should an investigator redirect themselves to the details of the investigation page the created. The additional feature here would be the **Add Logs button** that would redirect them to the **Edit Investigation Page**

4. *Create New Investigation*

After an Investigator decides to create and investigation from the report details, they are able to add a new Investigation Heading and a new Log Description which highlights the current action made by the investigator. Once submitted, the investigator would be redirected to the Investigation Index page.



Create

Report

Hazard Type

Select a hazard

Date and time hazard was spotted

18/06/2021 12:00 AM

Title

Description

Hazard Image

Choose File No file chosen

Create

Back to List

Click anywhere on the map to set the location. Right-click to remove the marker

Figure 20

5. *Add Log*

An investigation can edit the status of the investigation and edit the Investigation Header. The investigator also shows a list of all the logs made by the investigator and can add a new entry in the log investigation description.

6. *Delete Investigation*

An Investigator can delete their own investigations, they are first redirected to the Delete Investigation page where then they're able to confirm the deletion of the investigation. From the database side, all logs under the investigation are deleted as well.

Use of Roles

Two main roles were used to determine different functionality of the system. The roles used were those of Reporter and Investigator.

As one might expect, the reporter role is able to create a report and able to edit or delete their own reports. Reporters can also view a list of their own reports or a list of all reports currently on the system. An upvote feature was added to allow for reporters to upvote other reports, but not their own. This upvote feature is only restricted in terms of reporters not being able to upvote their own reports and there is no limit to how many times a reporter can upvote another report.

Investigators can open investigations on any report which does not already have an investigation open. They can also edit or delete their own investigations and have the added functionality of adding investigation logs on their investigations. All investigators can view a list of current investigations as well as the full list of current reports.

Every user registered and logged into the system can view the hall-of-fame page.

All the above functionality requires a user to be logged in as a reporter or an investigator, meaning that if any person who is logged in to the system will not be able to access any of the pages mentioned above.

Challenges Faced

Implementing Identity

One of the biggest challenges faced as we were developing the solution was once, we became introduced to the concept of ASP.NET Core Identity throughout the course studies. As before we reached this topic, we had just finished development creating from the ground up our own version of User and Roles including the controller methods for account and register, login and logout page respectively. With the addition of creating a separate class to handle session variables once the user is logged in or out.

At the time we reached a low point to our morale as we unintendedly attempted to reinvent the wheel for User manager and held it off for a while as we knew that we would have to refactor almost everything to accommodate the new introduction of Identity. Once the time arrived with a burning passion, we refactored the solution and adopted Identity through persistence and understandability. This became a huge turning point throughout the development as we also transitioned into a Code – First Approach including adopting proper Repository Patterns to the solution too.

As much of a drawback it seemed at the time to accommodate Identity, once we overcame it, we learnt more how to structure our code and increased production than ever before to achieve our final solution.

Time Management

As with every assignment time management is important to finish a solution of this magnitude especially throughout this year where the workload to keep up with all the units of the year went from manageable to almost nonstop juggling different assignments. This assignment also suffered consistent updates due to prioritizing other assignments in terms of their workflow and deadline, as noted by Figure 7 taken from the GitHub Repository.

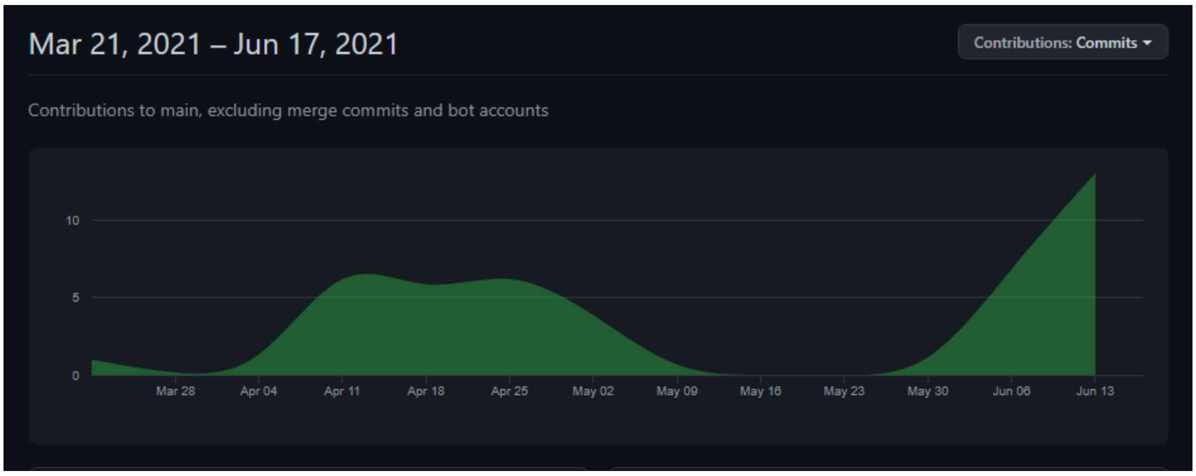


Figure 21

As shown, whilst from end of March there seems to not be any commits, there were meetings involved to plan how to tackle this assignment as development initially started beginning of April and kept at a steady pace till the beginning of May. Unfortunately, here is where the consistency ended as the demand for other assignment was increased and the focus was moved onto them. Production rapidly increased again end of May until we successfully completed the requirements of the assignment we were tasked.

Testing

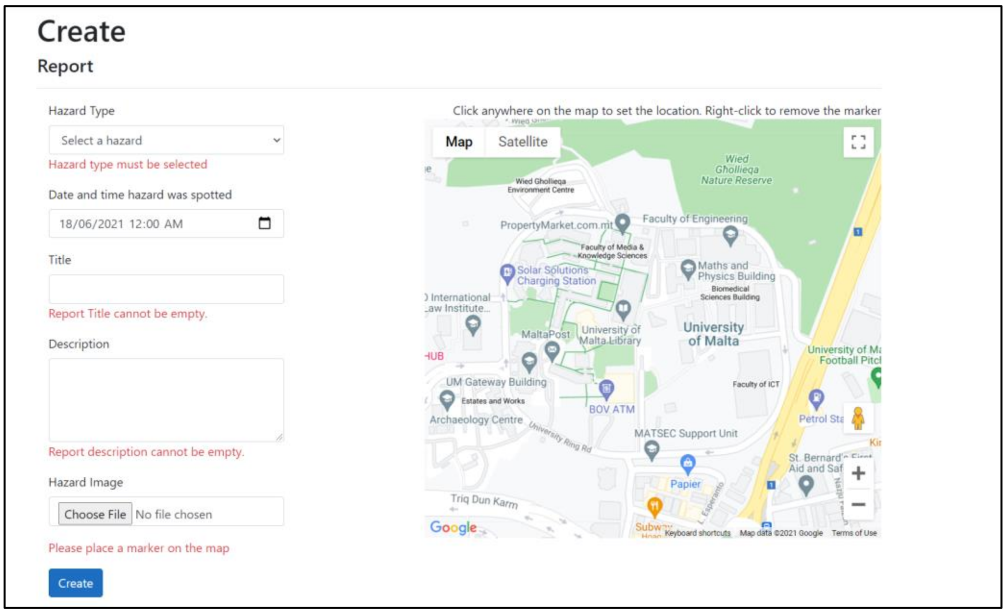
A project which focuses on best practices, validation, and logical structure as much as this needs to have proper testing done. Rigorous test of the system will show how reliable and robust it is and how well it handles certain, possibly unwanted, situations.

For testing of the system, we employed a user-based method in which we acted as users of the system and performed different actions under different scenarios.

When testing, the main things kept into consideration were the login state, meaning if the user is logged in or just visiting the site as a guest, and if logged in, the role the user is logged in. Depending on what role the current user has, tests were made to determine if the expected functionality is allowed and works well while checking that functionality not permitted to that role is not given.

The initial tests made were in terms of login authorization. By default, if a user is not logged in they will not have the available buttons for navigating to the reports, investigations or hall-of-fame pages, however the URL for each page was entered in order to make sure that the app does not let unauthorized users access these pages. If one of these URLs is submitted while not logged in, the app automatically redirects the user to the log in page.

The register page was then tested thoroughly, making sure to input different combinations of valid and invalid details. It was also kept into consideration to check that the required fields such as email, password and nickname, are actually required by the system and that it would not let a user register without filling these details. Email and Password fields were checked in order to confirm if they have the necessary validation. All this functionality and validation came with the Identity Framework.


Create

Report

Hazard Type

Select a hazard

Hazard type must be selected

Date and time hazard was spotted

18/06/2021 12:00 AM

Title

Report Title cannot be empty.

Description

Report description cannot be empty.

Hazard Image

Choose File No file chosen

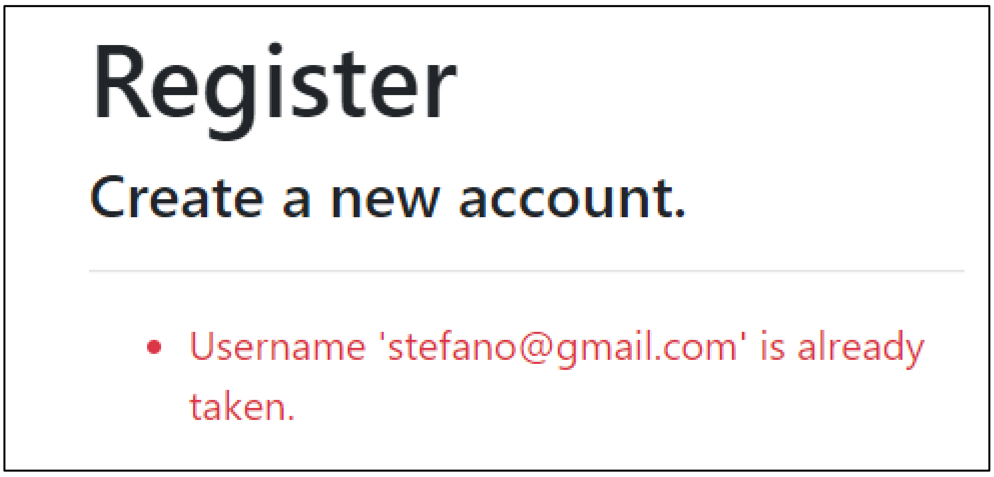
Please place a marker on the map

Create

Click anywhere on the map to set the location. Right-click to remove the marker

Figure 22

Once registered, a user is shown a page in which they can verify their account by simulating an email verification step. An issue with the current version of the app is that an actual email service is not set up and a dummy link must be clicked upon registration to confirm the account. If this link is not clicked and the user navigates to another page, the system will create the account however it will not let the user log in. This results in the user not being able to register with the same email address again as they will get a warning message stating that the email is already in use.



Register

Create a new account.

- Username 'stefano@gmail.com' is already taken.

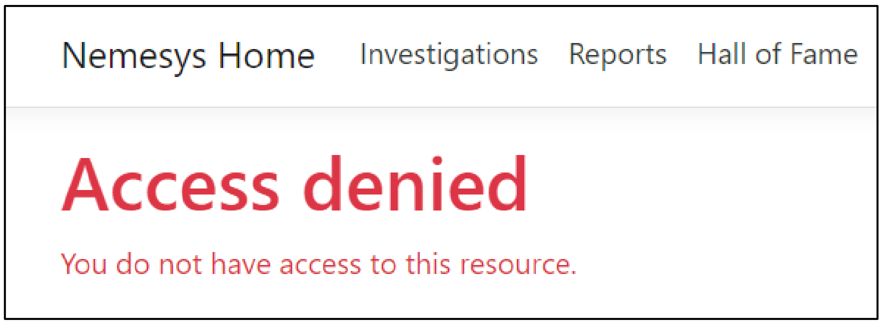
Figure 23

By default, new users are registered as reporters.

Once logged in, the navigation bar displays the appropriate buttons according to the role of the current user. A reporter can access the reports page which takes them to the index page listing all of the reports. Here, the user can click on the different filter buttons to select different views and these work as expected.

Reporters can also upvote other people's reports but not their own. This was tested and works as expected as well.

New report creation was tested by attempting to create new reports using invalid or missing data. The report was not created if this was the case. Once the form was filled in correctly, the report was created successfully and added to the database. We also attempted to access this report creation page by entering the reports/create URL while logged on as an investigator and got the following error.

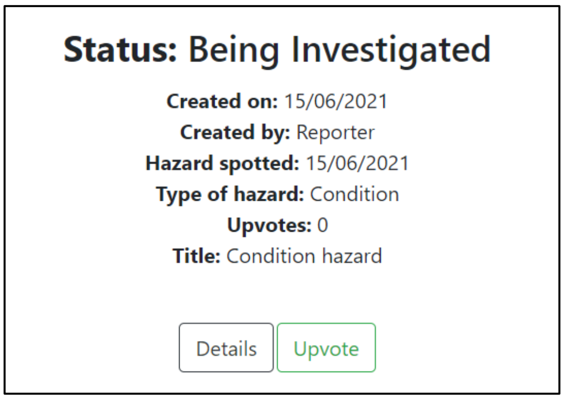


Access denied

You do not have access to this resource.

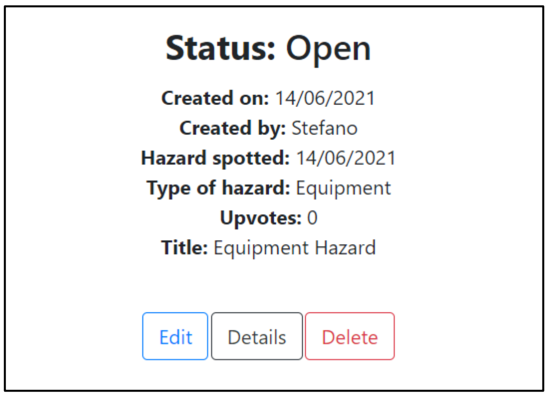
Figure 24

The reports index page also shows the appropriate buttons according to the user logged in. If a report is shown which belongs to the current user, the Edit, Details and Delete buttons appear, however if a report does not belong to the current user only the Details and Upvote buttons appear.


Status: Being Investigated

Created on: 15/06/2021
Created by: Reporter
Hazard spotted: 15/06/2021
Type of hazard: Condition
Upvotes: 0
Title: Condition hazard

Figure 25


Status: Open

Created on: 14/06/2021
Created by: Stefano
Hazard spotted: 14/06/2021
Type of hazard: Equipment
Upvotes: 0
Title: Equipment Hazard

Figure 26

The details, delete and edit pages were all tested accordingly. The edit page was checked properly in order to make sure the form validation is in order. One thing to note about the create and edit report pages is that the upload image section can accept any file type and save it without validation. Another thing is that there currently is no way for an image to be removed once it is added to a report, only changed to another image.

The cross-user validation was also tested by logging in as another reporter and trying to edit or delete a report not belonging to this account by typing in the URL. A 401 not authorized error was shown.

The equivalent for the investigator role was done by trying to create or edit investigations using invalid forms and trying to access functions which the current user does not have access to such as editing or deleting other investigators' investigations.

Finally, the hall-of-fame page was tested by accessing it, viewing the top reporters and seeing if the data displayed is correct, and clicking on the report buttons to see if the link works as expected.

Future Work

There are never enough additions that can be made to a website. Due to the time constraints the following concept were not achievable in the final product, these should be reserved for future work.

- Homepage:
    - An additional Google Map which would highlight all concurrent reports
- Overall increased UI display to give the web app a more polished final solution.
- Improving the UI on the account pages
- Reports:
    - Pagination for the index page, to not have all the reports loaded in a single page
    - Organize the index page based on upvotes
    - Remove reports loaded that has a status closed or no action required
- Investigations:
    - Refactoring the investigations controller to have a repository pattern approach
    - Addition of status category on the investigation index page
- Admin:
    - Admin view to manage the accounts registered in the database
    - Assign Roles to users and approve Investigators accounts.

Contributions

Stefano:

- Researched necessary fields such as ASP.NET, Database technologies, Entity framework.
- Contributed in the design stage (use case diagrams, prototype pages, database diagram etc.)
- Contributed in outlining the MoSCoW rules.
- Code:
    - Created the reports Models, ViewModels, Controller and all relevant functionality.
        - Create
        - Edit
        - Delete
        - Index
        - Details
        - Relevant ViewModels
    - Created the investigation Models, ViewModels, Controller and all relevant functionality apart from LogInvestigations.
        - Create
        - Edit
        - Delete
        - Index
        - Details
        - Relevant ViewModels
    - In charge of setting up and developing Google Maps API
        - All instances of Google Maps throughout the web app

Liam

- Creating the GitHub repository and adding the team collaborators
- Contributed in the design stage (use case diagrams, database diagram)
- Contributed in outlining the MoSCoW rules.
- Created the Database
- Code:
    - Created the initial Models, Controller and View with the assist of scaffolding
    - Initialized the DBContext to connect with the database
    - Added User and Roles until they had to be removed
    - Incorporated the ASP. NET Identity to the solution
    - Introduce the concept of Repository Patterns to the solution
    - Established Migrations for the solution to the database
    - Added InitializeDb class to populate the database after initial launch
    - Adjusted all Startup related methods in the solution
    - Added UpVote Functionality
    - Created the Hall of Fame page
    - Added the inclusion of Log Investigations to an existing investigation
    - Refactored Reports Controller to follow Domain Data-Driven Principles
    - Refactored Identity pages to allow more user accessibility