Final Report

Group 21

Alex Furmston, Bailey Walters, Hunor Liszka, Leon Chipchase, Mario San-Bento Furtado, Papa Onwona-Agyeman, Yix Wang

1 Introduction

Mentoring is at the heart of Deutsche Bank work culture[1]. Therefore, we have set out to build a prototype system to support a mentorship program. This system suggests mentors for a mentee, allows users to create and manage meetings, plans-of-action, and many other features outlined in the rest of this document. Because this project is aimed at building a prototype application, the scope is limited and does not include deployment (e.g.load balancing) or related aspects handled by other departments (building a privacy policy). However, we believe we have over-delivered to produce a website with more features than initially required, each of which is fully integrated with the backend, not making use of any mock data as would be typical with a simple prototype.

For information on specific requirements, please refer to the Requirements Analysis document.
For more information on the design of the system, please refer to the Design and Planning document.

2 Glossary

- **API -** Application Programming Interface
- **Backend -** The server-side component of the system. This includes the storing and processing of data, and communication with the frontend of the system.
- **CI/CD -** Continuous Integration/Continuous Deployment
- **DOM -** Document Object Model.
- **Frontend -** the client-side component of the system. This includes the user interface and enables the user to interact with the system.
- **MoSCoW -** Must Should Could Would, a method of classifying requirements for a system with different priorities.
- **React JS -** A free and open source frontend JavaScript library.
- **MUI -** a React component library.
- **MVCS** (Model View Controller Services) - a software design pattern commonly used to implement user user interfaces, data, and controlling logic.[5]
- **ViewModel -** A model of data which is presented to the frontend, and contains all data necessary to represent the object to the user.
- **VPS -** Virtual Private Server
- **NF/F.C/D X -** NF = Non-functional, F = functional requirement, C = customer facing, D = developer facing, X = requirement ID e.g. *NF.C5* = non-functional customer-facing requirement 5

3 System Overview

Onboarding

To use our site, a user needs to sign up as a mentee and/or a mentor. To do this, they must enter their details into the signup page. All communication is encrypted over HTTPS and their password is hashed and salted before being inserted into the authentication database. Furthermore, the language will default to the system/browser language so long as it is supported (currently English, German and Hungarian). Next, they are redirected to the Login page. Upon logging in, they will be presented with an onboarding process. Here, they enter: their name; their business sector; whether they are a mentor and/or a mentee; and the topics they need help unblocking / are willing to teach. This is all presented in an aesthetically-pleasing and intuitive way to ease the process. At the end of the onboarding process, they are asked what qualities they would like in a mentor; The matching system then suggests a list of mentors, ordered by suitability. The onboarding process is supported on all platforms, including mobile (as shown below by Figure. 1).

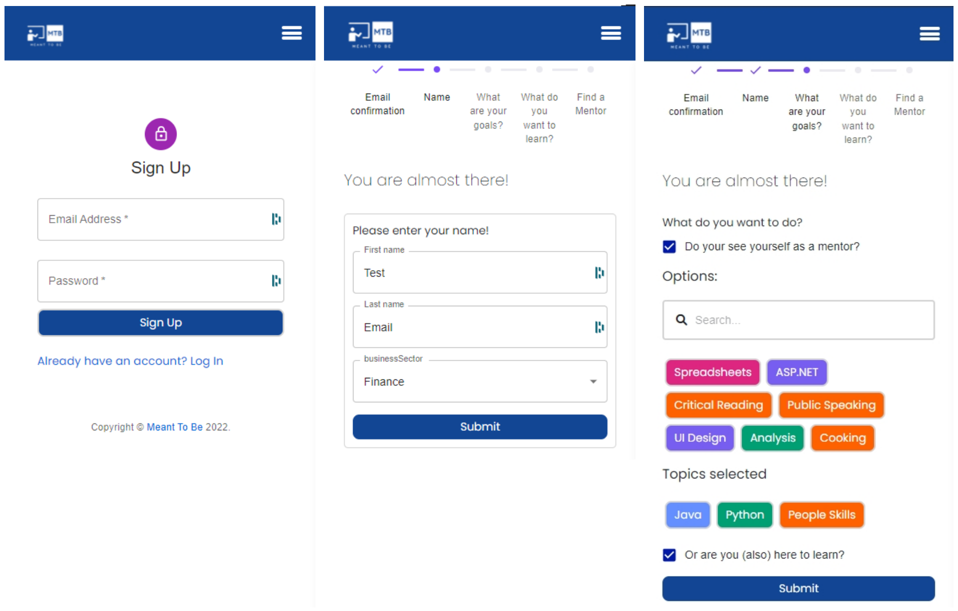
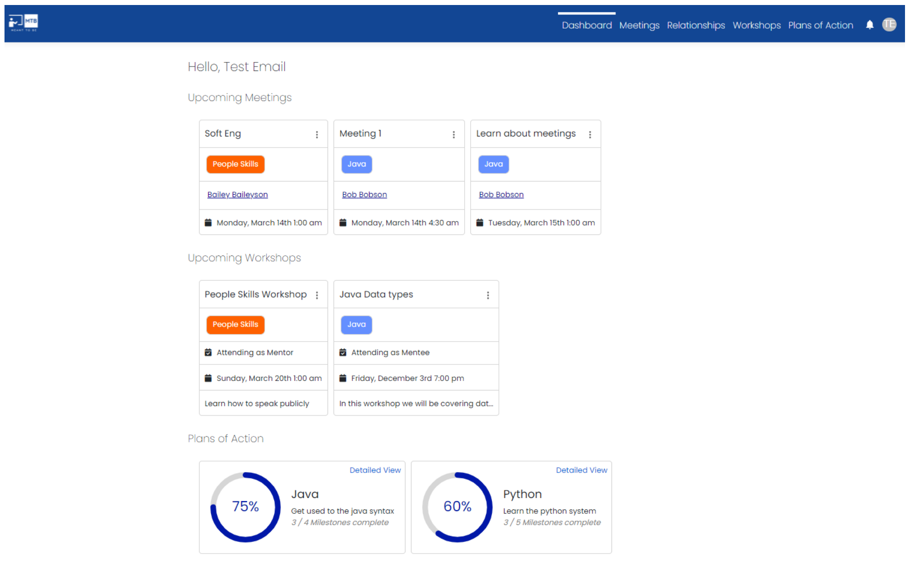


Figure 1: The onboarding process on an iPhone 12 Pro

Dashboard

The dashboard is the main page - it is what the user sees immediately after they log in. Therefore, it was important to include key information here, such as: a user's upcoming meetings, upcoming workshops, and their active plans of action (as shown in figure 2).



Hello, Test Email

Upcoming Meetings

Soft Eng	Meeting 1	Learn about meetings
People Skills	Java	Java
Bailey Baileyson	Bob Bobson	Bob Bobson
Monday, March 14th 1:00 am	Monday, March 14th 4:30 am	Tuesday, March 15th 1:00 am

Upcoming Workshops

People Skills Workshop	Java Data types
People Skills	Java
Attending as Mentor	Attending as Mentee
Sunday, March 20th 1:00 am	Friday, December 3rd 7:00 pm
Learn how to speak publicly	In this workshop we will be covering dat...

Plans of Action

75% Java
Get used to the java syntax
3 / 4 Milestones complete

60% Python
Learn the python system
3 / 5 Milestones complete

Figure 2: The "Dashboard" page displayed on a 16:9 monitor.

Meetings

Here, a user can view their upcoming and previous meetings. They may cancel a meeting, click “View Meeting” to see (and edit) more details (as shown in figure 3 about a given meeting or create a new meeting. When a mentee creates a meeting with a mentor, it sends the mentor a notification that they have been invited to a meeting, which they can accept or choose an alternative date for.

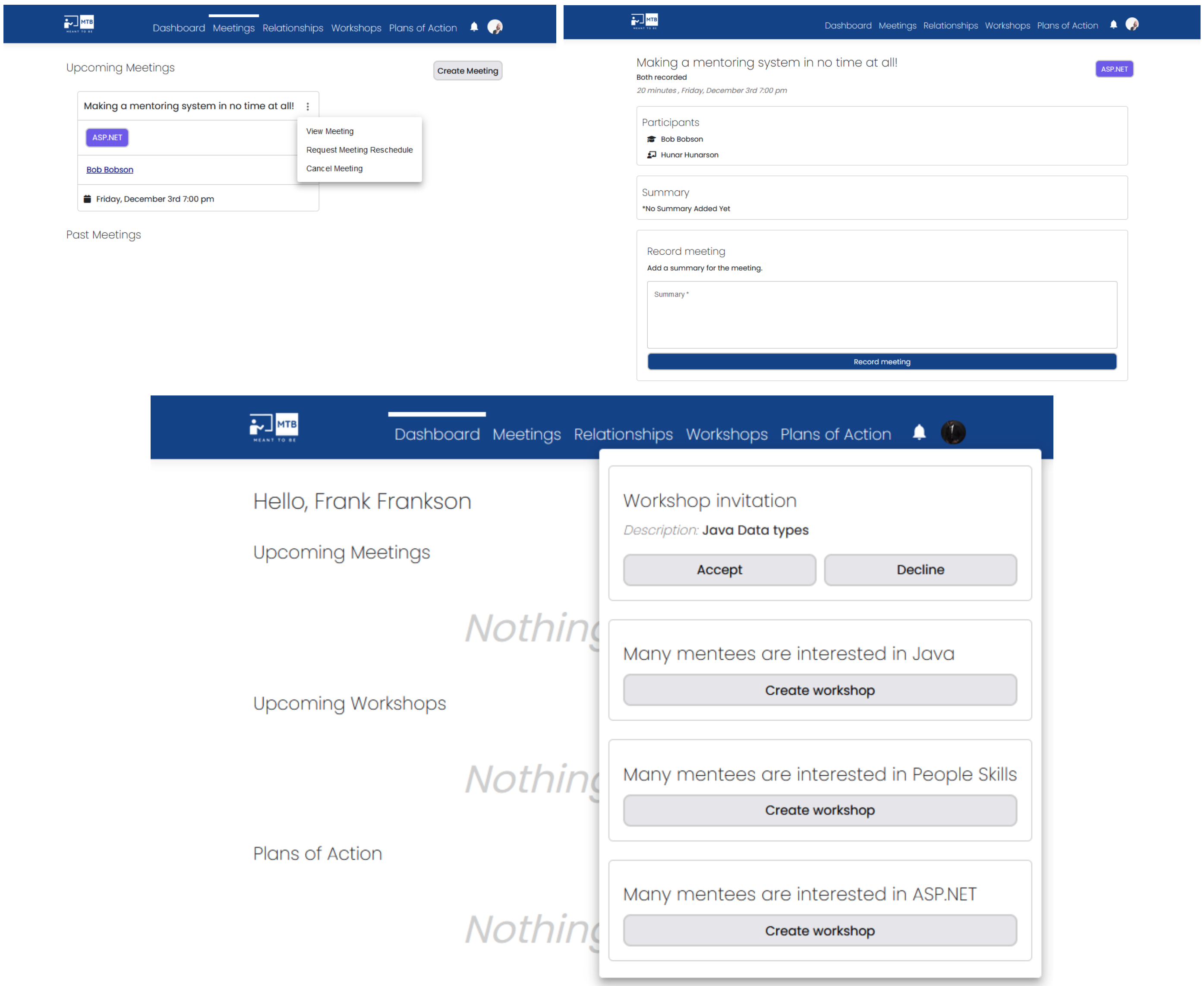
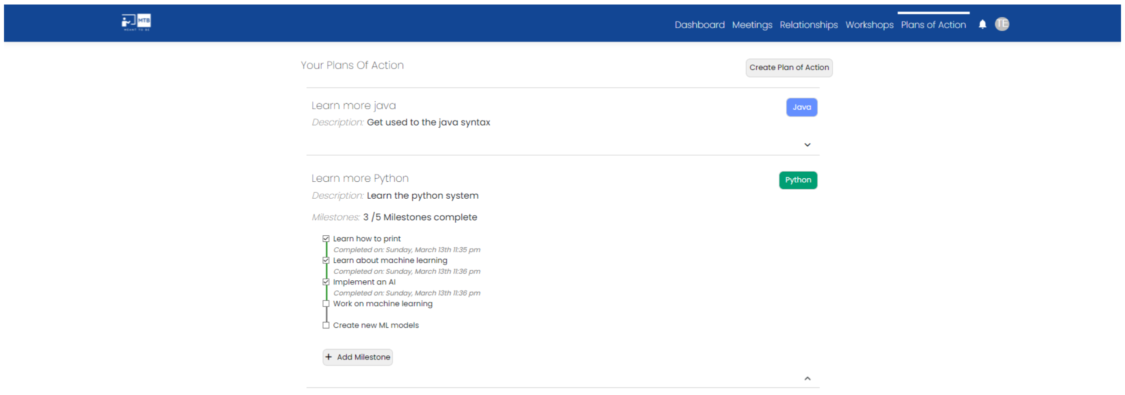


Figure 3: The meetings page, the summary page of that meeting, the notifications panel.

Plans of Action

Plans of action are an important feature for helping users to ensure their mentoring experience is relevant to the goals they want to achieve. On this page, users can create and view plans of action and the milestones that make up a plan of action; Milestones can be marked and unmarked and their date of completion is tracked and displayed along with overall progress in completing the plan of action.



Your Plans Of Action

Learn more java
Description: Get used to the java syntax

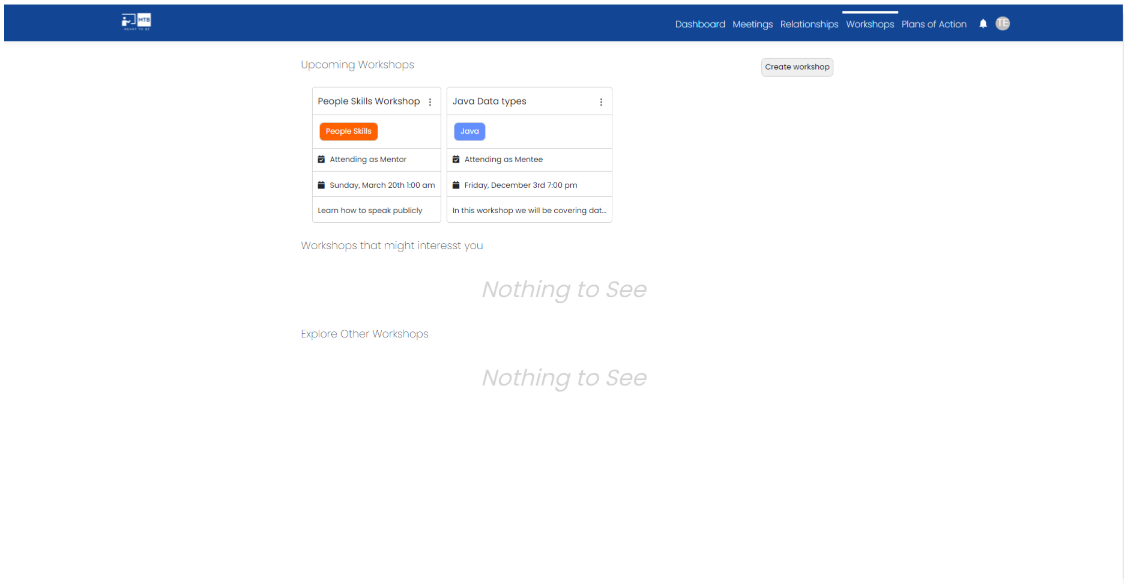
Learn more Python
Description: Learn the python system
Milestones: 3 /5 Milestones complete

- [x] Learn how to print
  Completed on: Sunday, March 13th 11:35 pm
- [x] Learn about machine learning
  Completed on: Sunday, March 13th 11:36 pm
- [x] Implement an AI
  Completed on: Sunday, March 13th 11:36 pm
- [ ] Work on machine learning
- [ ] Create new ML models

Figure 4: The "Plans Of Action" Page displayed on a monitor.

Workshops

Workshops are a great way of delivering teaching to multiple mentees at once, especially when a topic is in high demand. Therefore, when many mentees want to learn one topic, our system automatically sends a suggestion to a mentor (who is able to mentor that topic) that they host a workshop on that topic; clicking on the notification to create such a workshop will automatically fill in all suggested attendees.



Upcoming Workshops

People Skills Workshop	Java Data types
People Skills	Java
Attending as Mentor	Attending as Mentee
Sunday, March 20th 1:00 am	Friday, December 3rd 7:00 pm
Learn how to speak publicly	In this workshop we will be covering dat...

Workshops that might interesst you

Nothing to See

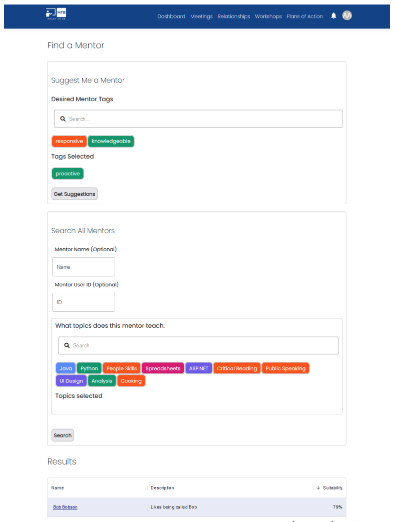
Explore Other Workshops

Nothing to See

Figure 5: The "Workshops" Page displayed on a monitor.

Suggestions Page and Relationships

The relationships page is where a user can manage their relationships with their mentors / mentees. A user may be both a mentor and mentee, so these lists are separated. As time goes on, a user may accumulate many mentors/mentees so the status (whether a relationship is active or over) is displayed alongside the mentor/mentee. On this page a user may click a button to acquire new mentors suggestions which will redirect them to the suggestions page, where they can enter the characteristics of their ideal mentor and have the system automatically find a list of suitable matches ordered by a suitability rating determined by the AI. Alternatively, they can enter the name, ID, or the topics taught by a mentor to do an exact search for certain profiles. Once they have found a mentor they deem suitable, they can request for that mentor to mentor them via their profile.



Name	Description	Suitability
Bob Bobson	Likes being called Bob	79%

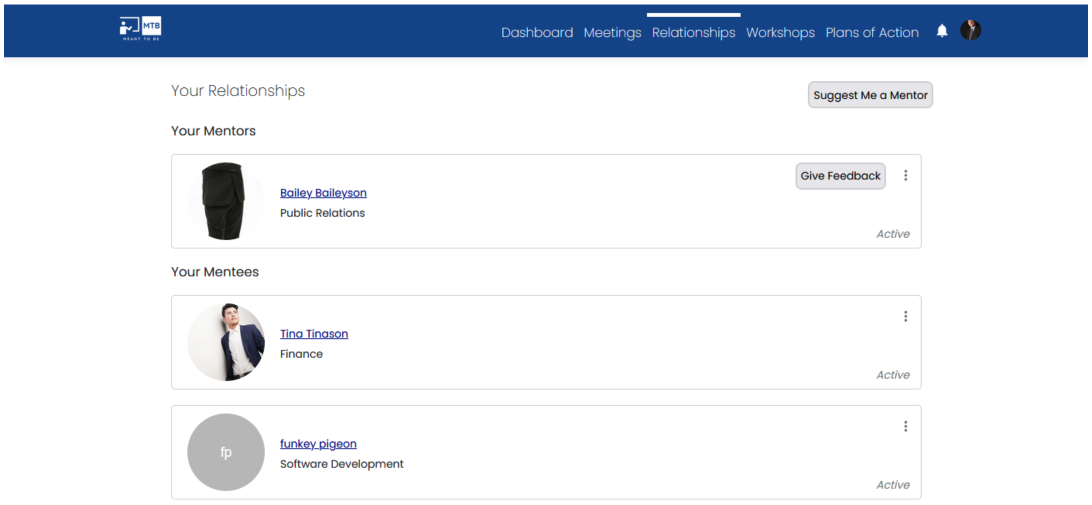
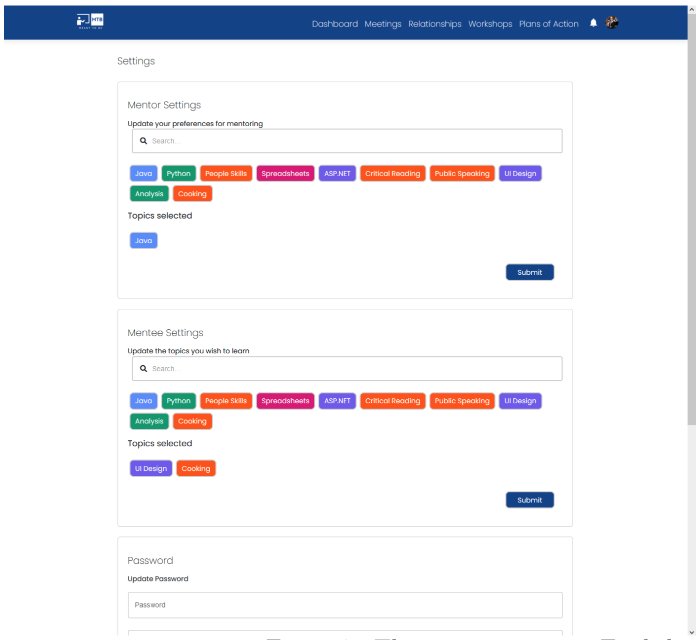


Figure 6: (left) Alex asks for a proactive mentor and is suggested Bob. (right) Leon's Relationships page

Settings

It is important for users to be able to edit information which they have entered earlier. Therefore, on the settings page they may edit the topics that they mentor, the topics they are mentored in, their password, and their language. The first two are implemented on the frontend but not hooked up to the backend. The only languages available at the moment are English, German, and Hungarian. However, the frontend infrastructure exists so that any new languages could be added very easily. This feature was viewed as important due to the international nature of a large company such as Deutsche Bank.



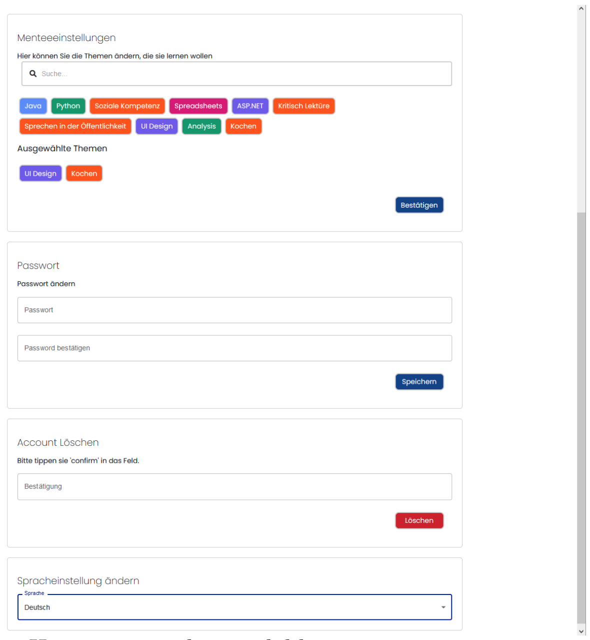


Figure 7: The settings page in English and German. Hungarian is also available.

4 Modifications

Overall, the changes made to the system requirements were limited. We aimed to ensure all essential requirements were fulfilled in order to create the best possible prototype in the given time frame. However, despite this, due to a combination of inaccurate timeline estimation and unforeseen delays due to continued problems with unfamiliar tooling, some features needed to be reworked. Other requirements were slightly altered in order to change or improve the features due to new knowledge, learning more about the system in development and user feedback. The overall product still provides a well rounded application which fulfils all the essential requirements requested by the customer with changes only being made to non-essential requirements.

4.1 Missed and Changed Requirements

The following subsections outline the missed or changed system requirements as specified in the Requirements Analysis document. Each change or omission will have a description justifying the change, or the reason for the omission.

Key: *M = modified requirement, D = dropped requirement.*

4.1.1 Missed and Changed Non-Functional Requirements

C7 M Ensuring full GDPR compliance would have taken a very long time, and would be best done in conjunction with a legal team to ensure all laws and regulations are followed correctly. Therefore, we cannot guarantee that the site is fully compliant but it follows GDPR as much as is practical for a prototype. Many measures were taken to ensure the maximum security of user data: using a secure site connection; separating user authentication details from the main system; and encrypting user data in-case of a database breach.

C9 D There will not be an email notification system as it is not an important feature for a prototype, especially when considering that there is an on-site notification system. Therefore, there is no need for *C9*.

C16 M Testing response times can be very time consuming, particularly for the prototype, therefore the main goal was to test the functionality of the system using unit tests and endpoint tests. As the system would be scaled, this would be important to ensure the quality of life for the users using the system.

4.1.2 Missed and Changed Functional Requirements

C8 D Mentors will not be able to mark milestones as complete for mentees they are in a relationship with. This requirement was not implemented because we deemed it more fitting to the rules of mentoring for plans of action to be managed by the mentee alone.

C10 M Instead of a mentee being able to suggest multiple time slots, a mentee can suggest one time slot. The mentor can then choose to accept the meeting or suggest an alternative time. Therefore, the site still allows a mentee to book a meeting in a time slot, but in a slightly different way.

C11 D *C11* was not considered essential to ensure the full functionality of the system, as mentors and mentees are able to write the location (room) in the meeting invite description. However, for quality of life and ease of use this is a feature which should be added in the future.

C12 M Although not a requirement, the design doc specified that *C12* would be implemented with a single-matching system (for recommending multiple mentors to a single mentee, in order of suitability) as well as a multi-matching system (for matching multiple mentors with multiple mentees at once). The latter system was dropped as it was not considered important for the proper functioning of a prototype or even a final product. Furthermore, it would provide only minor benefits to users. The single-matching system works as intended.

C15 D C15 was not judged to be important to the functioning of the site as mentors and mentees can specify their location in their profile description and make their own decisions based on that. Therefore, *C15* was dropped in favour of more important requirements.

C18 D This functionality was removed since the system is still a prototype, and other features which still needed to be implemented were prioritised over this.

4.2 Non-Requirement Changes

While the database was almost identical to what it had been planned to be, there were some minor alterations in the relationships - many to many, many to one, or one to one, and various tables required additional fields in order to function as specified. The new database diagram is displayed in figure 11.

The UI has undergone extensive changes and additions from the initial wire-frames. While colourblindness support for Protanopia, Deuteranopia and Tritanopia has been maintained[8], the colour-scheme has been expanded for a more vibrant and expressive visual language, aimed to better differentiate repetitive information such as business sector by a brief glance, aided alongside by greater use of iconography. Unfortunately, the aforementioned benefits will not affect users with Monochromacy colourblindness, although their ability to use the application is unaffected.
Another significant area of change is the inclusion of several additional pages to better fulfil the requirements, such as the search page.

The design doc specified that *F.C12* would be implemented with a single-matching system (for recommending multiple mentors to a single mentee, in order of suitability) as well as a multi-matching system (for matching multiple mentors with multiple mentees at once). The latter system was dropped as it was not considered important for the proper functioning of a prototype or even a final product. Furthermore, it would take a lot of time to implement would provide only minor benefits to users. The single-matching system works as intended.

5 Discussion of Development

5.1 Development Tools Used

5.1.1 Source Control - Github

Git was used for version control and Github was used for remote hosting and CI/CD (continuous integration / continuous deployment). This is a system that most of our team was familiar with and is rich with features that were useful during development. The repository consisted of a "dev" or development branch. This was linked to a VPS (Virtual Private Server), and each time the dev branch was updated a series of Github Actions were run (such as linting). If the Github Actions passed and the build succeeded, the updates would be pushed to the VPS. Ideally, each time dev was updated all unit tests would have been run before making any changes live to ensure any changes made would still pass all unit tests to ensure the "dev" branch was always fully functional so team members could rebase their branches without fear of introducing unrelated bugs. However, towards the end of the development these actions could no longer be run as we reached the running-time limit for Github Actions imposed on private repositories.

Each time a new feature was implemented, a new Git branch would be created for isolation between independent features and merged once the feature was fully implemented; Isolating changes in different branches rather than directly committing to the "dev" branch mitigated the potential issue of code erasure significantly, preventing any large setbacks during development.

5.1.2 Development Environment - Visual Studio

Most of the team used was Visual Studio 2022 as their development environment, this was chosen since it had the widest range of features for C# development due to official support from Microsoft who develop both the IDE and language. The Github integration and build tools also helped significantly streamline our development process.

5.1.3 Language - C#

Our chosen programming language for the backend of the system was C#. Initially this seemed the most appropriate choice for the team since it is an Object Oriented Programming language - a paradigm that the team was very familiar with, allowing us to achieve *NF.C12*. Furthermore, C# is slightly less verbose than Java. In hindsight, Python may have been q more appropriate language for such a short-term project due to the speed at which Python code can be written. However, it is very easy to write insecure, error-prone code in Python due to its dynamic typing which does not catch type errors at compile time, which is why we decided not to use it. Overall, C# was a decent choice and would make further developments after the prototype stage easier and more sustainable.

5.1.4 Framework - ASP.NET

ASP.NET is the industry standard for web development in C#. It was a good choice for our group as some of us already had experience with ASP.NET and most of us had experience with C#. Furthermore, it allows for good scalability, which would be very important if this prototype were to be developed into a full product. Unfortunately, ASP.NET is more complex that some web frameworks, such as Flask or Django. The ease-of-setup of these frameworks would have been beneficial when considering the tight timeline of this project. However, once everyone had gotten more familiar with ASP.NET, development was very efficient and we benefited greatly from the highly-structured nature of ASP.NET development. Therefore, this can be considered a good choice of framework overall.

5.1.5 Language - TypeScript

Our chosen language for the frontend was TypeScript. This is due to the static type checking capabilities offered by it over standard JavaScript which results in a large number of runtime bugs being caught at compile time instead, providing higher productivity during development and greater security in the application. We believe that this was a good decision as backend modifications of the REST API schema were easily adapted on the frontend by only modifying the TypeScript interface definitions which would instantly flag up all files that still used the old schema. Unfortunately, a pitfall not covered by the static type checking is when external IO is involved such as in network communication for API calls, this is because IO happens at runtime and all type information is stripped after compilation, resulting in the type checker being unable to correctly verify conformance. To mitigate this, we used a layer of adapter functions that handle network communication so any type faults are at least easily isolated to the adapter layer.

5.1.6 Framework - React

We have chosen to use React as our frontend framework as it is the most popular of the industry standard frameworks. We feel this was a good choice due to React being a fairly light and simple framework which allowed for the frontend team who were mostly experienced only in backend development to learn as the project went on. Furthermore, React is a purely declarative web framework based around composition, where state and logic is isolated from parent components, allowing for greater fault isolation and the unique advantage of being able to insert pre-made components (such as those of the MUI library we utilised) for significantly increased productivity during development.

5.1.7 Containerisation - Docker

Docker[6] is the industry standard for containerisation. We used it to ensure our site worked identically on every team member's computer. Furthermore, our use of Docker allowed us to use a Github Action to automatically push our changes from the "dev" branch to a remote server without the configuration needing to be changed. The remote site *was* hosted on Digital Ocean, a cloud hosting provider, at `https://www.cs261group21.com`.

When setting up Docker, we ran into a lot of configuration problems. For example, we encountered end of memory errors, and different configurations were needed for Linux and Windows. In spite of this, we persisted and the time spent wrangling with Docker greatly decreased as the project progressed and the productivity benefits of a consistent environment outweighed the initial cost of learning and setting up Docker.

For this project, we used 4 containers: backend, frontend, main database, and authentication database. The databases are persisted on the host machine whilst dependencies aren't due to the OS-specific binaries required by certain dependencies (such as *esbuild*). As multiple team members worked on Windows, it made more sense to separate it from the the Linux-based docker containers. We managed docker on windows with docker-compose. Whilst this could be done with Kubernetes, that was deemed unnecessary due to the limited scope of this project.

5.2 System Components

5.2.1 Authentication

The tokens themselves are stored as Cookies in the user's browser. They are only sent on HTTPS requests and the SameSite attribute is set to strict to defend against CSRF[12] attacks. If an authenticated user is required for a given action, the request is passed to the AuthenticationMiddleware which checks the validity of the token. As these tokens are stored in the database, if a user's account is compromised, all of their login tokens can be revoked to ensure no malicious actor would be able to access the system. This might be necessary, as tokens are designed to either last a single day, or when the user requests it, 31 days; expiration of tokens is handled by the backend for greater security, as the client-side expiration date of cookies may be tampered by the end user. To maximise flexibility, the system was designed to be able to revert to a Java Web Token-based authentication. This could happen when the system needs more scaling and the security feature of access revocation is not necessary.

5.2.2 Matching System

The matching system consists of three main functions: GetScore, SearchMentors, and SuggestMentors. As with the rest of the backend, the matching system was written in C# with the core logic being handled by the Mentor Suggestion Service.

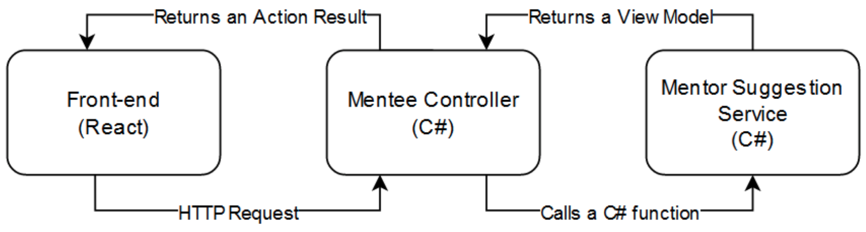


Figure 8: Matching System

GetScore calculates the compatibility of a mentee-mentor pairing based on the following formula: $score(mentor, mentee) =$

$$(1 \times \frac{topics\_in\_common}{topics\_mentee\_wants\_to\_learn} + 0.3 \times \frac{average\_rating\_across\_common\_topics}{5}$$

$$+0.5 \times \frac{tags\_in\_common}{tags\_the\_mentee\_wants} + 0.1 \times \frac{overall\_mentor\_rating}{5}) \times 100 \div 1.9$$

Note: $\times 100 \div 1.9$ is there to make the overall score a percentage.

The weights of these factors were carefully selected based on experimentation with the test data to ensure what we viewed as the best matches would be scored appropriately. If a mentee and mentor are in the same business sector or have no topics in common then the score would be 0, the mentor would not be recommended and the rules of mentoring[7] would be upheld. Therefore, functional requirement C13 is fulfilled.

SearchMentors searches through the database to find mentors matching the search query. Users are able to search by:

ID Find the mentor whose MentorID exactly matches the provided ID.

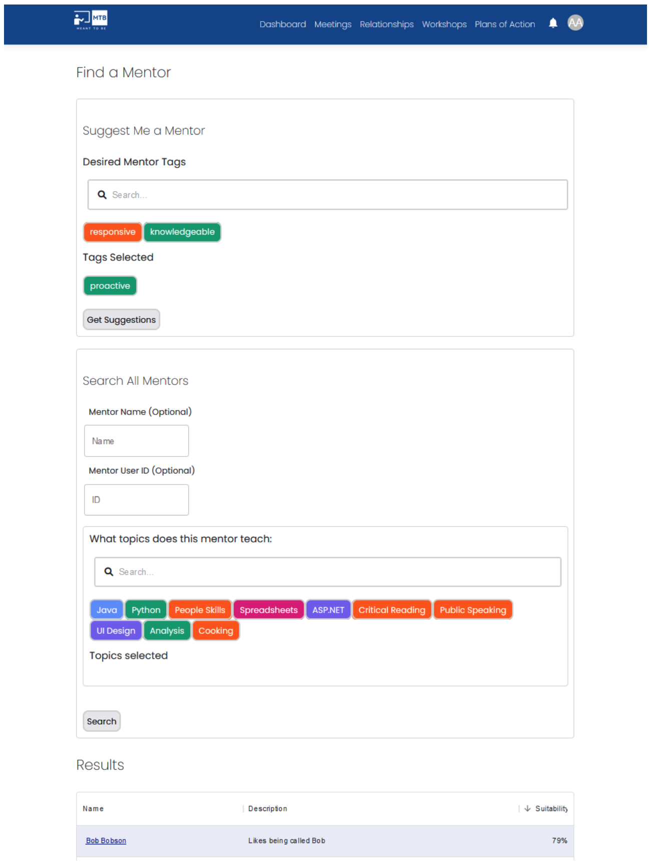
Name Search for mentors who contain the given string as a substring of their first and/or last name.

Topic Search for any mentors willing to teach 1 or more of the listed topics.

By default, searches (not suggestions) involving topics will be sorted by the number of topics the mentor and search query have in common (alternatively by first/last name, business sector, and rating (mentor's global average rating)) and will fetch results based on matching IDs, partially matching names and/or mentoring capability in the selected topics.

SuggestMentors returns valid mentor suggestions for a given mentee (valid means that a mentor and mentee are from a different business sector and the mentor is willing to mentor at least 1 topic that the mentee wants to learn). It gets suggestions by initially filtering out all invalid mentors for the given mentee, then selecting all mentors in decreasing order of suitability (as determined by GetScore). Although not currently used, the method has the capacity to only search up to a set number of mentors to improve the efficiency of the suggestions in a large database. Note: SuggestMentors will work even before many ratings are added into the database as it is able to use factors (e.g. topics in common) that don't rely on reviews from past mentees.

In summary, SuggestMentors clearly fulfils C12; C13 is fulfilled as SuggestMentors won't suggest an invalid mentor and the backend won't let mentees create a relationship with an invalid mentor; and C14 is fulfilled as GetScore provides a score for mentor-mentee pairings based on the relevant parameters. C15 was not completed. However, it was rated as a 'could' under our implementation of the MoSCoW system, so it was considered more important to work on other features. Furthermore, mentees can manually choose their mentor based on location.



Name	Description	↓ Suitability
Bob Bobson	Likes being called Bob	79%

Figure 9: User "Alex" requests that the site recommend a "proactive" mentor who would be a good match for them. This request is fulfilled using the SuggestMentors function. "Bob" is recommended, with a suitability score of 79%.

5.2.3 Frontend/UI

The frontend was developed as a Single-page application[11] in Typescript using the React library. The frontend only retrieves necessary data from the backend via the REST API, allowing for optimal performance due to minimised network activity.
Because this is a single-page application, routing is handled client-side by the React Router library[13]. This gives further performance benefits and allows the website to feel dynamic and responsive, as page reloads are eliminated and only the altered parts of the DOM are re-rendered on navigation.

The frontend is decomposed into individual React components encapsulating only their own necessary logic and state. They are organised into a hierarchical tree with each component managing its own life-cycle and cascading changes down onto its child components. Separating these features into different components allows for vastly increased re-usability and fault isolation, allowing for increased productivity and faster debugging.

The necessary models used by the REST API are defined as TypeScript interfaces and interaction with backend endpoints is handled by a layer of functions that correspond to each endpoint, to allow for easy dependency inversion by swapping out the layer for one that returns fully correct mock data instead of using the backend, this allows for easily testing the frontend in isolation to any potential backend faults.

The UI makes use of MIT-licensed React component library, MUI[9] for increased productivity within the limited time constraints. Furthermore, use of these widely-used components helps users to feel more familiar with our site.

A somewhat systematic approach for frontend development was initially chosen. However, despite these plans and frequent communication between frontend developers, development fell behind the schedule outlined in the Gantt chart. This can be attributed to a lack of prior experience with the various frontend technologies used, which led to learning and development occurring simultaneously, slowing down development time substantially. This issue was solved by restructuring roles within the team, with backend developers frontend areas.

5.2.4 Testing

As part of our Scrum approach, we intended to implement test driven development. We used xUnit[3] to perform unit tests on our C# code, Postman[4] to perform endpoint tests, and manual integration testing. The goal was to write our unit tests at the start of each sprint cycle, then check that any features implemented in the sprint cycle were correct. This would have given us an indication of the progress we had made in each sprint before starting new tasks. However due to delays in implementation and our lack of familiarity with xUnit and Postman, this was not possible. Instead, it was often more practical to write tests at the end of a Scrum cycle. However, each section of the project was still thoroughly tested. We split our testing into 3 broad categories:

- Unit Tests (xUnit)
  Unit tests were used to test the services, which were used by the controllers. These tests ensured that data passed to services by other parts of the website would be correctly manipulated and the correct output returned. These tests included edge-cases (such as trying to match users with themselves in the matching system) to guarantee that our services were as reliable as possible. The organisation of the backend into services and controllers means that these were the most important tests for the backend (the controllers are mostly wrapper functions of corresponding service functions), as such these are the most extensive tests.
- Endpoint Tests (Postman)
  Endpoint tests were used to determine whether the API endpoints were working correctly, and that the controllers would receive the data as intended. This minimised the effort to integrate the frontend and backend as we knew our endpoints were working correctly.
- Integration Tests (Manual)
  Integration testing was performed manually as the time saved from automatic testing would not have been worth the time taken to develop automatic tests. Nonetheless, integrating testing made sure that the frontend and backend were properly integrated and the user would have full access to the data they required which was stored in the database.

Unit and endpoint tests were continuously run throughout the development of new features, checking that any new development would work with the system as a whole before being merged into the "dev" branch. Integration tests were performed periodically (upon any merge into "dev").

5.2.5 Backend/API

For our backend development, we used the MVCS (Model View Controller Service) paradigm as a basis for our development. This splits the backend into:

- Model:
  A structure which holds the data and logic for a particular object.
- View:
  The visual representation of the model which is displayed to the user.
- Controller:
  The endpoints of the API which handles any requests made by the client.
- Service:
  An additional layer of decoupling in the system which acts as an intermediary between the database and the controllers themselves.

Each controller uses a set of services, each of which would insert, update, delete or retrieve data from the database which would then be returned to the user.

For the connection to the database, set of contexts were used to allow the backend and the database to communicate. In order for the frontend to be able to communicate with the backend, a set of controllers were created, each containing a set of API endpoints which processed all requests made to the system. The controllers implemented were:

1. AppFeedbackController
   The AppFeedbackController handles the storing of feedback submitted by users. This uses the AppFeedbackService which converted an AppFeedback view model to an instance of an app feedback object to store in the database. This controller was created to fulfil the requirement that the users of the system are able to provide feedback about the system. This means the backend was implemented to achieve the functional requirement C3, as users' feedback would be able to be processed and stored.
2. AuthenticationController
   The AuthenticationController handles login and session-related activities. It generates tokens on login, validates them when passed in from requests. It is also the controller, through which the user's ID is fetched, which is required for requests by other services.
3. MenteeControler
   The MenteeController handles all data relevant to a user who is registered as a mentee. It uses the services PlanOfActionService, ScheduleService, MentorSuggestionService, and RelationshipService. These services provide all relevant functionality to the mentee. This allows the mentee to update any relevant information to their active relationships, their schedule, their plans of action and to search for and choose a new mentor.
4. MentorController
   The MentorController handles all data relevant to a user who is registered as a mentor. It uses the services PlanOfActionService, ScheduleService and RelationshipService. This allows the mentor to update any plans of action their mentees are currently undergoing, manage their schedule - their upcoming meetings and workshops, and manage any relationships with mentees, provide feedback or manage a plan of action.
5. MiscController

The MiscController returns the full set of business sectors, topics and mentor tags and uses the MiscService to achieve this. The data from here will be used to display to a new user the business sectors which currently exist, the topics they would like to teach or learn, and the different tags that a mentor could choose to define their teaching style.

6. OnboardingController
   The OnboardingController is used to handle user registration. This gets the personal details for a user after they have registered for an account. This includes their full name, business sector, topics they want to learn/teach and if they want to be a mentor, a mentee, or both.
7. PersonalController
   The PersonalController handles all data relevant to specific users. It uses the PersonalInfoService and the ScheduleService, which allows users to view the specific details about themselves, such as their mentoring profile and their schedule. This also allows the user to delete their account and any data which has been stored associating to that account to maximise how closely GDPR regulations have been followed.
8. RelationshipController
   The RelationshipController allows users to manage their relationships. It uses the RelationshipService to achieve this, meaning the user can view information about a specific relationship, all their current relationships, and also end a relationship - when they no longer wish to mentor/be mentored by this particular individual. The information about past relationships is maintained in order to keep a record of the past experiences and feedback the user has had.
9. UpdateProfileController
   The UpdateProfileController allows the users to update their personal information. This uses the UpdateProfileService, and allows the user to change the business sector - in-case they have moved to a new position, change whether they would like to become a mentor or a mentee, or manage the topics which they are teaching or are learning.

Each of the controllers allow the user to effectively receive and update information relevant to them, whilst compartmentalising different components of the system so that each request only requires a small subsection of the system. This allows for improved maintainability, more readable, decoupled code, and lets different members of the backend team work in different areas of the project in parallel speeding up the development of the system. Furthermore, the design follows OOP design principles[14] (*NF* C12). Figure 10 illustrates how the MVCS model is implemented in the system, as well as how the services provide an extra layer of compartmentalisation between the system as a whole, acting as an intermediary to the database contexts. “Mentorship Services” refers to all services but the Authentication and AppFeedback Service, to maintain clarity in the diagram.

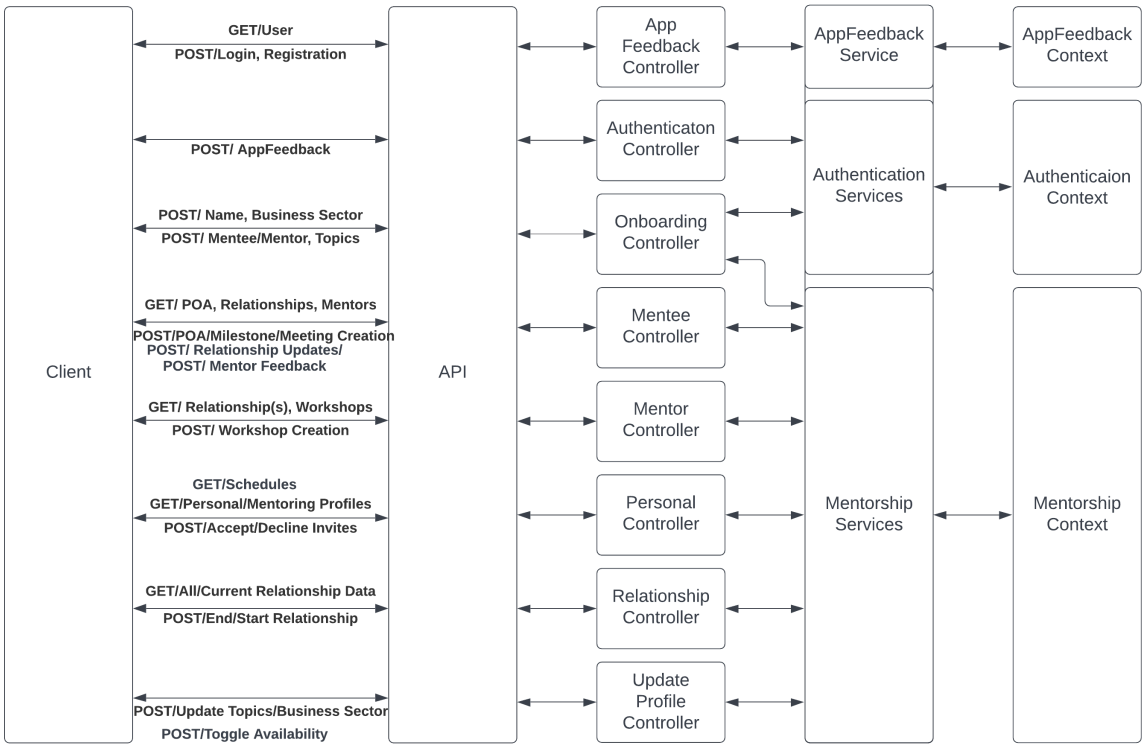


Figure 10: API Diagram

The API created is documented using Swagger, an Interface Description Language used to describe APIs. Each endpoint has a summary of its functionality, its parameters and the type that it returns. This was done to ensure the future readability and maintainability of the system by providing ease of access to documentation, helping to achieve the requirement: *NF*.C11. Moreover, this also allows the frontend team to look at the generated Swagger Doc (the generated document created from the API documentation), to understand what exactly is needed for the API calls. This makes frontend development of these calls far more efficient and less prone to issues with integration.

5.3 Database

As shown in figure 11, the database was split into three contexts. This is far less than initially intended, as we had planned to break up the contexts further to ensure that if some data was needed from the backend, only the tables directly related to this access should be accessible. However, many inter-table relations would have spanned affected multiple contexts. Therefore, configuring these contexts would have been difficult, if not impossible.

- The Authentication Context
  The user table is stored in the authentication database, which is the link to the rest of the system. This is then separated into a separate context; therefore, whilst handling authentication of the user, only the needed database tables are able to be accessed. This improves both the security and maintainability of the system, as the size of the system increases due to keeping distinct data in separate containers.
- Mentorship Context
  The Mentorship Context handles all data in green in figure 11. This is the main system context, and is used by all controllers which handle user, relationship, and workshop data. The reason for the singular context covering all these tables was due to the interconnected relationships of all the tables. Therefore, almost anytime data from a single table was needed, another entity which was related to that table would be required. Defining these relationships in a single database context was the most appropriate choice.
- App Feedback Context
  The app feedback table is kept separate from the rest of the system because there are no entity relationships linked to it. This meant that it could be stored in a separate context - only the required table was needed, reducing the load on the system. This also ensures no other data can be accessed maliciously.

The separation of these contexts helps us to achieve the requirement of maximising GDPR compliance, as it prioritises the safety of the data on the system at all times. This also ensures that only the data required for the user at any stage is available to them.

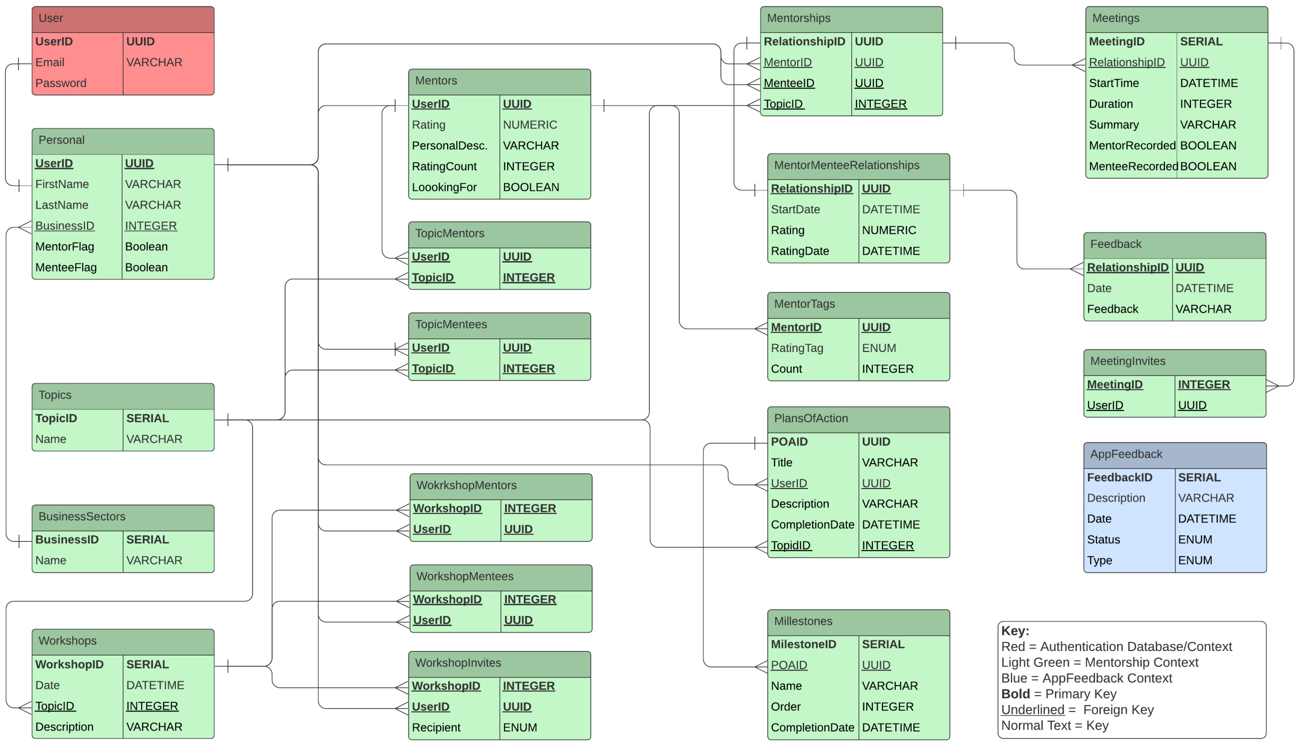
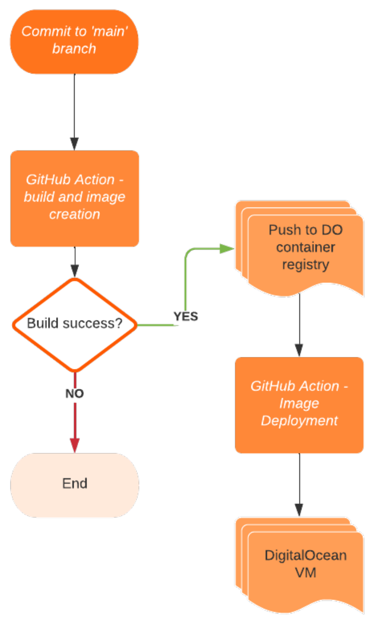


Figure 11: Database Entity Relationship Diagram

6 Deployment, Maintenance and Scaling

To simulate deployment and ease testing of software, we set up a Github Action to deploy to a remote server. This ensured that our code would both build successfully, and could be run on a generic host. Another Action was created for checking the formatting on the backend, ensuring conformant linting across the repository. This decreases the time for a developer to get used to sections of code, which they themselves have not written, as the styling is consistent file-to-file. The Model-View-Controller-Service pattern implemented in the program builds upon the MVC paradigm by adding an extra service layer between the model and controllers. This results in a more decoupled system, leading to lower maintenance costs in this medium-sized, growing project. For the scale of the prototype, we had decided the usage of Docker would fit the project. Nevertheless, in the case that a single machine may not keep up with user demands, there are options to handle the stress. The easier, more basic solution: Docker Swarm. This would be a plug-and-play upgrade on the current Docker Compose setup, as it builds upon it to allow multi-host containerisation. If further scaling is necessary, more fine-grained scaling can be achieved by Kubernetes, in conjunction with breaking up the main database into separate microservices.


Commit to 'main' branch
GitHub Action - build and image creation
Build success?
YES
NO
End
Push to DO container registry
GitHub Action - Image Deployment
DigitalOcean VM
Figure 12: Deployment/Build pipeline used during development

7 Evaluation - Product

7.1 Testing and Validation

7.1.1 Unit Testing

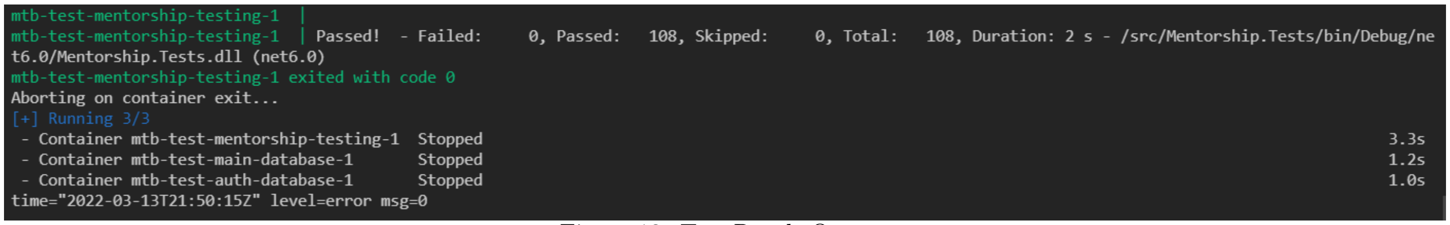
Unit tests were done using xUnit[3], an open source testing tool for the .NET framework. The primary goal of our unit tests was to ensure that all the services worked as intended to fulfil the requirements set for the project. We wrote test-cases containing valid data and invalid data in order to determine whether the service would process the data correctly. We specifically tried to write tests to cover all edge-cases.

To connect to the existing database, fixtures were used with a mock logger to mimic the state the system would be running in during its use. All tests are done using data provided by a dump file. The tests were split across the different services as follows:

Unit Test Summary

Service	Total Tests	Tests Passed	Test Success
AppFeedback	1	1	Yes
MentorSuggestion	10	10	Yes
Misc	2	2	Yes
PersonalInfo	6	6	Yes
PlanOfAction	17	17	Yes
RelationshipService	18	18	Yes
Schedule	39	39	Yes
UpdateProfile	15	15	Yes
Total	**108**	**108**	**Yes**

The total test output is displayed in figure 13.



```
mtb-test-mentorship-testing-1  |
mtb-test-mentorship-testing-1  | Passed!  - Failed:     0, Passed:   108, Skipped:     0, Total:   108, Duration: 2 s - /src/Mentorship.Tests/bin/Debug/ne
t6.0/Mentorship.Tests.dll (net6.0)
mtb-test-mentorship-testing-1 exited with code 0
Aborting on container exit...
[+] Running 3/3
 - Container mtb-test-mentorship-testing-1  Stopped                                                                                          3.3s
 - Container mtb-test-main-database-1       Stopped                                                                                          1.2s
 - Container mtb-test-auth-database-1       Stopped                                                                                          1.0s
time="2022-03-13T21:50:15Z" level=error msg=0
```

Figure 13: Test Result Output

7.1.2 Endpoint Testing

For the testing of the controllers and API endpoints, Postman[4] was used to create the appropriate requests for the creation, deletion and updating of data in the system databases. These tests allowed us to determine that the frontend would be able to communicate with the backend once it had been fully implemented. This was important as the front and backend were developed independently. Therefore, these tests ensured that there were minimal problems integrating both parts of the system. As it turned out, these tests did their job and there were no major issues with integrating the front and backend, satisfying *NF.C15*.

For all tests where a user must be logged in, the requests made also contained an authorisation token. This token authenticates that the requests made are valid and carried out by a user who had the appropriate permission to update the associated data. Furthermore, wherever a UserID is required for the request, this is taken from the authorisation middleware rather than as a part of the request body or query.

User endpoints

The user endpoint tests test whether the user registrations and onboarding processes work as specified. This includes

- Registering as a new user
- Giving their personal details - name and business sector
- Selecting whether they would like to be a mentor, mentee or both
- Choosing the topics that they would like to learn or teach

The following tests illustrate they key components of onboarding that a new user would need to do.

Test Case	Test Description	Test Input	Expected Result	Actual Result	Test Success
Create a new user.	Testing whether registering a new user is successful.	Email: test@test.com, Password: asdASD123	HTTP response code 200 and message: "Registration successful"	HTTP response code 200 and message: "Registration successful"	Yes. *F.C0*
Give the personal details.	Testing whether after entering personal details, a new Personal instance is created for that user.	FirstName: Leon, Surname: Chipchase, BSID: 1	A new Personal instance will be created with the details specified.	A new Personal instance was created for the corresponding user.	Yes. *F.C0*
Setting a new users' mentor and mentee tags, and selecting their topics.	Testing whether a new user can successfully register as a mentor, mentee or both, and specifying the topics which they would like to teach or learn about.	MentorFlag: true, MenteeFlag: false, TopicIDs: 2,4,6	The users mentor flag is set to true, and the mentee flag is set to false. A new Mentor instance will be created for the user, giving them a mentoring profile, and a set of MentorTopic instances will be created for the TopicIDs specified.	All information was stored correctly, required instances of MentorTopics and Mentor were created, and the mentor and mentee flags were correctly udpated.	Yes. *F.C1*

Mentee Endpoints

The Mentee endpoint tests verify that a user who has registered as a mentee is able to do everything they should be able to do. This includes:

- Managing their relationships with their mentors - updating and creating plans of action, setting up meetings and rating their mentor.
- Searching for new mentors, and finding suggested mentors based on their criteria.
- Accepting or rejecting any meetings or workshops.

These tests illustrate how a mentee would communicate with the system and what data they need to provide in order to be able to carry out these requests.

Test Case	Test Description	Test Input	Expected Result	Actual Result	Test Success
Creating a new plan of action.	Testing whether a mentee can create a new plan of action.	Description: Develop Java, TopicID: 3	A new plan of action is created for that Mentor-Mentee Relationship with the chosen topic.	The new plan of action was created for that relationship.	Yes *F.C9*
Searching for a new mentor.	Testing whether the correct mentors come up based on the search query	SearchQuery: "Java"	The list of mentors are queried based on whether they can teach Java and returned to the user.	The correct list of mentors were returned to the mentee.	Yes.

Find a new mentor based on the suggested matching of the system.	Testing whether the user finds a new mentor based on their preferences and compatibility.	desirebleTags: “nice”, “proactive”	The set of tags are sent to the backend, and mentors are found based on the query. They are then checked to ensure they are in different business sectors.	The correct set of mentors are returned, with no invalid mentors being displayed.	Yes. *F.C12*
Creating new milestone.	Testing whether a user can create a new milestone for their plan of action.	Name: Learn Python, OrderValue: 1	A new milestone is created for the corresponding plan of action, then the plans of action are ordered based on the OrderValue given.	The milestone was successfully created, and the milestone order was updated accordingly.	Yes. *F.C8*
Creating a new meeting.	Testing whether a user can create a new meeting with their mentor.	StartTime: 18/03/2022 12:20, Duration: 20	A new meeting is created at the specified date and an invite is sent to the mentor in the relationship.	The meeting was created at the specified date and the mentor was notified.	Yes. *F.C10*
Providing mentor feedback.	Testing whether a user can provide feedback on their mentor.	MentorID: (mentor guid), NewRating: 4.0	The rating for that mentor by the mentee will be updated to 4.0.	The mentor rating was updated successfully	Yes. *F.C14*
Accepting a workshop invite.	Testing whether a mentee can successfully accept an invite to a workshop.	AcceptInvite: true	The invite is accepted, and the mentee is added to the Workshop Mentees table.	The mentee was successfully added to the table of attending mentees.	Yes. *F.C17*

Mentor Endpoints

The mentor endpoints verify that a mentor is able to use the full functionality that is required for them:

- Creating a new workshop
- Getting a suggestion for a workshop that is in demand by other mentees.
- Accepting a mentee meeting

The following tests outline the key functionality for the mentor endpoints. All endpoint tests which are very similar to those previously tested in the section above have been left out.

Test Case	Test Description	Test Input	Expected Result	Actual Result	Test Success
Create a new workshop.	Testing whether a mentor can create a new workshop.	Date: 18/03/2022, TopicID: 1, Title: Python, Description: Learn Python	A new workshop is created, and the list of invitees is generated.	A new workshop is created and a set of invites is created for the invitees.	Yes. *F.C17*
Get a workshop suggestion.	Test whether the system correctly suggests a workshop for the mentor.	None	The system searches through mentees who want to learn a topic and returns the workshop suggestions.	The correct list of workshops were returned.	Yes. *F.C16*
Accept a mentee meeting.	Accept a request for a meeting from a mentee.	AcceptInvite: true	The invite is accepted and a new meeting is created.	The new meeting was created.	Yes. *F.C7*

App Feedback Endpoints

This endpoints reflects how a user would submit feedback about the system.

Test Case	Test Description	Test Input	Expected Result	Actual Result	Test Success
Register a new instance of feedback.	Testing whether the user is able to register a new instance of feedback.	Description: Perfect, Type: other	A new feedback instance will be inserted.	Addition of the feedback was successful.	Yes. *F.C3*

7.1.3 Frontend Blackbox Testing

To gather an understanding of what users unfamiliar with our site would think of it, we hosted the site on https://cs261group21.com and sent the link to non-computer-scientist friends and family. We gave each of them a list of tasks to do, differing by account type:

Both mentees and mentors:

- Create an account (we let them choose whether to be a mentor or mentee) and complete the onboarding process.
- Send site feedback.
- Join open workshops.

Mentors:

- Create a workshop.
- Receive meeting invite and suggest an alternative time instead.
- Write meeting summary and record as finished.

Mentees:

- Create a meeting.
- Create a plan of action and add goals to it.
- Record meeting as finished.
- Rate mentor.

The overall feedback we received was that the site looked 'very professional' but 'slightly boring'. Users liked the plan of action page and found it very easy to use. However, a common point of criticism was the disorganised presentation of the upcoming schedule as a list of upcoming events; a calendar or some sort of better visual presentation of their schedule would have been useful. A point of confusion was when forms such as creating meetings are filled correctly but violate invariants such as exceeding the duration limit on the backend. In such cases the backend error is not returned to the frontend and users are left confused as to why their form was not accepted. However this was relatively uncommon and common form errors, such as leaving out required fields, are clearly shown to the user. Lastly, users often tried to click various information cards and names (such as meeting summaries and mentor names) to attempt to get further details which was not supported by our initial design.

In response to the feedback collected, we made significant visual styling additions to the UI resulting in the modifications detailed above in Non-Requirement changes which aimed to address the "boring" presentation of the application at the cost of weakening full Monochromacy colourblindness support. Furthermore, we implemented several additional informational popups and new pages that are linked to the existing components that users can now click on for a detailed view (names linked to profile summary popups, meeting cards linked to detailed view via a menu, etc...). Unfortunately, due to feedback being taken relatively late in the development cycle, time constraints prevented us from implementing certain features. We did not have time to entirely redesign the UI for viewing the schedule. Nor did we have the time to make intrusive backend and frontend changes to propagate invariant violations to the end user. However, these changes should be relatively simple to implement given more time and should not require extensive reworks due to the decoupled architecture and form invariants were instead also implemented on the frontend, to reflect the requirements of the backend.

7.1.4 Testing Summary

Overall, the testing on the system was as rigorous as possible for the time available. The unit tests verified the services acted properly on correct data, erroneous data and edge cases. The Postman API tests ensured the frontend was properly able to communicate with the backend of the system. And the frontend testing verified that the end users were able to use the system as intended. However all tests specified were carried out by members of the team who had an intimate knowledge of the system. This has the following issues:

- The team know how the system works, therefore may unintentionally avoid sections which may break the system.
- The team know how to navigate the UI, therefore testing of the effectiveness of the frontend is difficult.
- Purely using unit tests and Postman tests will inevitably miss cases which have potential to cause an error.

A fix for this would have been to have a wide range of users testing the system. Then monitoring how the system responds, and identify where or how the user is able to break a component. However this is a very time-consuming task, and is not feasible for a prototype at this stage of its development.

7.2 Matching System and Relationship System

The rules of mentoring state:

1. "A mentee should always drive the relationship, setting the agenda and approaching the mentor when help is require". This is maintained as mentees are the only people who can start a relationship and they are the only ones who can create a meeting.
2. "A mentor will always give up time to the mentee". This is maintained as mentors may only accept or suggest an alternative date in response to a meeting / mentorship request.
3. "The mentoring arrangement should be based on themes and topic areas that they require help unblocking to achieve success." GetSuggestions clearly fulfils this as its GetScore function is based on the number of topics a mentor teaches that a mentee wants to learn.
4. "A mentor should always be from a different business area to the mentee." This is enforced in the relationship service (where mentor-mentee matching are actually created) and the GetSuggestions function in the suggestion service (where matchings are suggested).

Out of requirements *F.C6* - *F.C15* (all the matching system and relationship system requirements), all those rated 'Must' on the MoSCoW system are fulfilled. Only 2 requirements, *F.C15* and *F.C11* ('Could's) have been dropped. Therefore, overall the Matching System and Relationship System can be considered to have been very successful.

7.3 UI

The user interface was developed mainly through the use of the MUI [9] React component library. MUI is a robust library of high quality components usable under the permissive MIT license, from the navigation bar to popup modals. It allowed us to efficiently implement our website design and develop the frontend faster than would have otherwise been possible.

The MUI framework is heavily customisable, with each component having a variety of attributes to specify both styling and functional properties. This allowed us to adapt the website design to match our accessible colour scheme and unique matching system. For example, you can see below that the chips to select topics are differentiated by colour, using a hashing algorithm that picks from our preset array of colourblind-friendly colours. This ensures that any additional topics added will not require any backend nor frontend code changes and will be automatically accounted for.

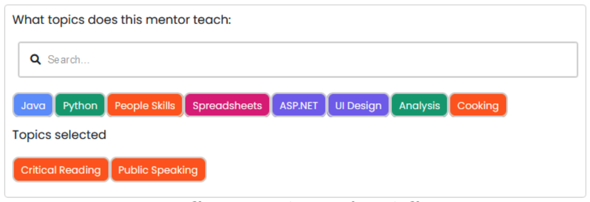
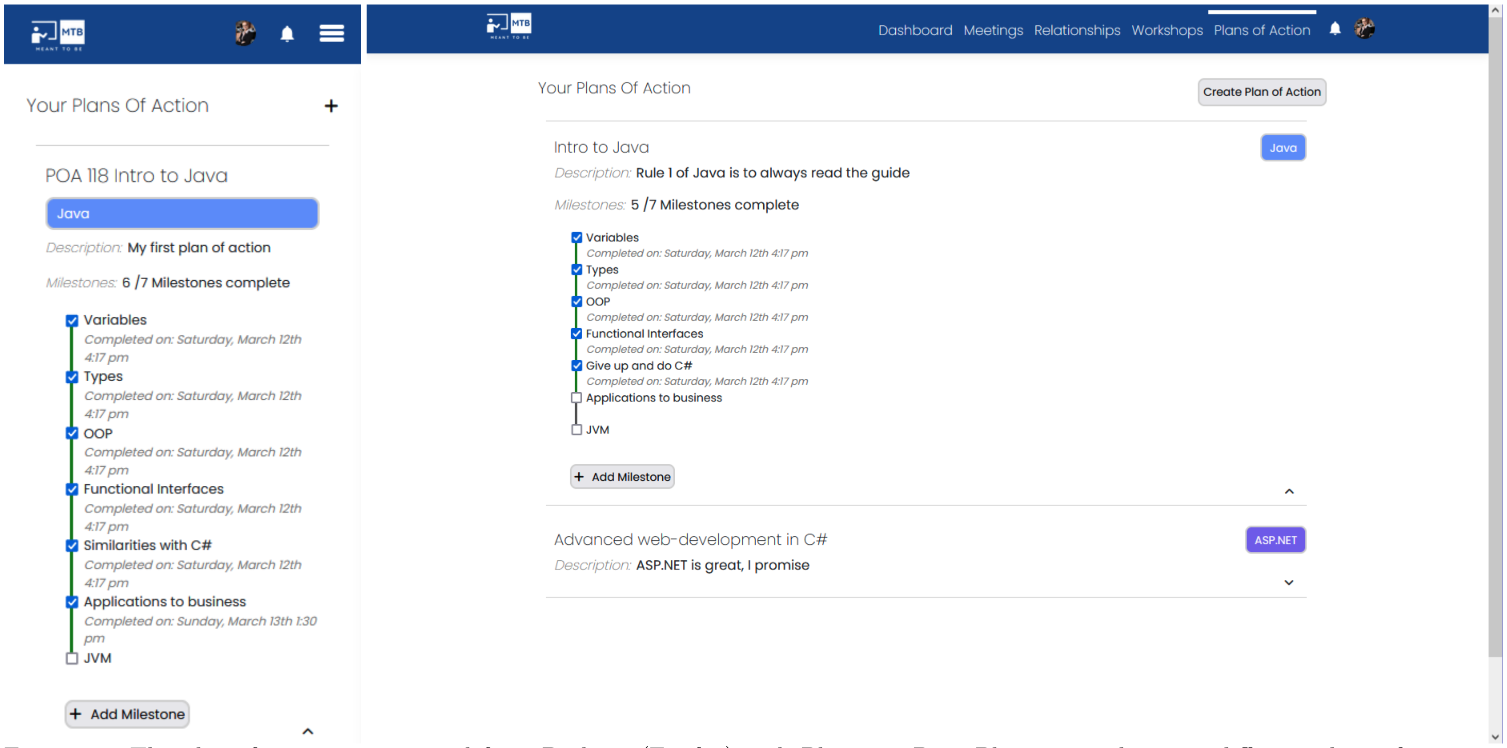


Figure 14: Different colours for different topics.

We used CSS media queries to meet our responsiveness requirement, selectively rendering components to allow for our website to be adapted for use on desktop computers, tablets and mobile phones. For example, on smaller screen sizes, the navigation bar is collapsed into a hamburger menu (as shown by Figure 15).

We have tested the web application on the latest supported versions of all major browser engines (including Gecko (Firefox), Blink (Chrome, MS Edge, Opera) and Webkit (Safari)) on all major platforms (Windows 10+, MacOS, Linux), ensuring full functionality and a consistent experience (requirement *NF.C19*). (See below)



Your Plans Of Action

POA 118 Intro to Java

Java

Description: My first plan of action

Milestones: 6 /7 Milestones complete

- [x] Variables
  Completed on: Saturday, March 12th 4:17 pm
- [x] Types
  Completed on: Saturday, March 12th 4:17 pm
- [x] OOP
  Completed on: Saturday, March 12th 4:17 pm
- [x] Functional Interfaces
  Completed on: Saturday, March 12th 4:17 pm
- [x] Similarities with C#
  Completed on: Saturday, March 12th 4:17 pm
- [x] Applications to business
  Completed on: Sunday, March 13th 1:30 pm
- [ ] JVM

Your Plans Of Action

Intro to Java

Java

Description: Rule 1 of Java is to always read the guide

Milestones: 5 /7 Milestones complete

- [x] Variables
  Completed on: Saturday, March 12th 4:17 pm
- [x] Types
  Completed on: Saturday, March 12th 4:17 pm
- [x] OOP
  Completed on: Saturday, March 12th 4:17 pm
- [x] Functional Interfaces
  Completed on: Saturday, March 12th 4:17 pm
- [x] Give up and do C#
  Completed on: Saturday, March 12th 4:17 pm
- [ ] Applications to business
- [ ] JVM

Advanced web-development in C#

ASP.NET

Description: ASP.NET is great, I promise

Figure 15: The plan-of-action page viewed from Desktop (Firefox) and iPhone 11 Pro. Please note these are different plans of action, despite the similar names.

In terms of meeting our usability requirements *NF.C18*, we ensured that our website would be clear, concise and easy to use by implementing a wide variety of intuitive controls for easy interaction; Primary action buttons (Create Meeting, Join Workshop etc.) were given more visual weight to draw the user's attention. Iconography was also used throughout the site to enhance various components and improve the user experience.

7.4 Security and Data Protection

As mentioned in the Requirements Modifications, requirement *NF.C7* was modified. Instead of the system being fully GDPR compliant, we decided to make the site as close to being GDPR compliant as reasonably practical for a prototype built in 10 weeks without the assistance of a legal team. This is an acceptable compromise when considering that the system is a prototype and other requirements must take precedence.

Nonetheless, the site has plenty of security and data protection features. Passwords are pre-hashed with SHA384, then using the BCrypt[2] scheme and a salt generated by BCrypt.Net.BCrypt.GenerateSalt(), fulfilling requirement *NF.C5.D1*. The database is encrypted with ASP.NET encryption, satisfying *NF.C5.D0*. Personal user data is anonymised in the database, accomplishing *NF.C7.D0*. We achieved *NF.C8* as the website uses HTTPS. Finally, authentication data is stored in a separate database to everything else, accomplishing *NF.C6* and further increasing the security of the website.

Functionality for deleting user data was implemented for both the main database and the authentication database. In the authentication database, all data relating to the account is deleted permanently, including login information and authorisation tokens, but in the main database, all personal information is anonymised. This ensures that whilst a users' account no longer exists, records that past meetings occurred will still exist.

To summarise, although the website may not be fully GDPR compliant, every other data protection and security requirement was met. Clearly, very strong fundamentals for data protection and security have been built into the core of this prototype. This is very important and enables further security measures to be added in a later stage of development.

7.5 Summary of Product Evaluation

Overall, we believe that the prototype that we have produced is a success. In fact, we have gone beyond what would be required for a prototype and have linked almost every frontend feature up to the backend, resulting in a near fully functional product. This is an example of the robustness of our backend.

An indicator of success is that all 108 of our backend tests passed successfully. This further highlights the robustness of both our test-driven development process and the implementation of the backend in the product.
Despite not meeting all of the requirements set out in the requirements document, we have ensured that almost all of the *Must* requirements and many of the *Could* and *Should* requirements were successfully implemented.

Initial feedback showed a generally positive response to the website, both functionally and aesthetically, another indication of the success of our prototype.

8 Evaluation - Process

8.1 Development Methodology

We adopted a Scrum-based[10] methodology with influences from plan-based methodologies such as waterfall. Our design document specified that we would allow for flexible team roles, use a Github project board for organisation, Github Actions for continuous testing, and follow 1-week sprint cycles. Due to the concrete deadlines imposed on the requirements analysis, design document, video demonstration, and final report, our methodology involved some practices typical of plan-based methodologies: we gathered to plan the architecture of the project and tightly coupled parts such as the database schema before starting development.

The flexibility provided by allowing team-members to change roles partway through the project was invaluable. Near the end of development, our project manager and scrum master switched away from their roles working on the backend to work on frontend integration and styling respectively. This helped our frontend, which was behind schedule, to catch up with the rest of the project. Furthermore, our data scientist and other backend developer took the lead on writing the report and our database engineer started working on the video demonstration towards the end of the project. These transitions in role were partially based on how complete certain parts of the project were and could not have been accounted for with a purely plan-based methodology. Hence, Agile working was beneficial to our project.

When our Github project board was used, it was useful for keeping track of the progress of the project. However, it was underutilised - not every task was added to the board and tasks which were done were not always marked as such. This was probably a contributing factor in the delays and subsequent crunch-time on the frontend. If we were to do another project, the project board would be better utilised as team members would be more familiar with it and we would know to make an effort to remind each other to use it.

Github Actions were very useful for continuous testing, ensuring we would immediately know if there was a fault in the code on Github. We also immediately knew if a pull request caused our ”dev” branch to break. This made fixing errors more efficient.

Our scrum-cycle was modified to use 2-week sprint cycles rather than 1-week. The reason for this is that we found 1-week to be too regular of an interval to carry out sprint cycle planning, review, and retrospective. Therefore, we decided that it would be more efficient to extend this cycle to 2 weeks. This sort of change is to be expected when most of the team do not have prior experience developing websites in a group. Extending our sprints to 2 weeks gave us the flexibility within cycles that was needed to maintain our focus on developing a site that effectively fulfilled the requirements laid out for it.

Our design document and our plans for the system architecture and databases were very useful in keeping the different components of our project (frontend, backend, matching system, etc.) compatible and consistent. Therefore, the plan-based aspects of our methodology were effective.

In conclusion, our development methodology was a sensible choice. Overall, it was effective in allowing us to deliver a website that met the requirements laid out in the project specification we received and the requirements analysis. While some parts could have been executed better (such as better use of the Github project board), the overall execution of the methodology was good.

8.2 Effectiveness of the Team

Overall, the team meshed together very well. The team was primarily split into backend and frontend development teams, with the project manager moving between roles to provide aid where it was needed. The frontend and backend teams worked on distinct but codependent sections of the system. Therefore, effective communication between teams was vital to the success of the project. An example of this is creating the endpoints for the API: the backend team worked on controllers and services to handle any requests made, and the frontend team needed to create the pages and create the requests for the API. Along with these endpoints, POSTMAN tests were built to show the format that the data needed to be in to be sent to the backend. Both teams referred to the system diagrams created in the design document to ensure both sections were compatible.

Due to the scope of the project, there were technologies which were unfamiliar to at least one of the team members in each group. For this reason, pair programming was used - typically in group meetings. In these meetings, 2 team members would go through documentation and implement features, ensuring that multiple team members were able to work on those components independently in the future. A key example of this was where we had begun to implement unit tests for the backend services. As this was a topic which was relatively new to the backend team, pair programming helped reduce the learning curve as we were able to work through the problems in tandem.

We chose our project manager as they had the most well-rounded knowledge of the technologies and practices required for a project of this scale to be effective. Consequently, their assistance could be of use across the entirety of the project, which helped development run as smoothly as possible. Despite this, there were learning curves for each member which meant sometimes there were delays and more attention was needed to be able to learn what was required.

8.3 Team Communication

Overall, the team communicated effectively and often met in person to discuss development and issues. However, sometimes not all team members were available to attend a meeting. Therefore, our main source of online communication was Discord, an online service allowing for private servers with multiple text, voice, and video channels. It was the best choice for our team for the following reasons:

- The whole team was already familiar with it and used it regularly.
- It allows for various channels to organise our messages.
- It allows for integration with a Github webhook, enabling the whole team to easily monitor changes to our repository.
- It allows for the pinning of relevant messages so that important information (such as meeting notes) is always accessible.

The server was split into the following channels:

#general	Used for general communication. Details about the project as a whole, organising group meeting and any other miscellaneous information relevant for the entire team.
#databases	Used for discussing development of the database system. Initial planning, changes, progress, and the insertion of test data were all discussed in this channel.
#backend	Used for discussing our backend configuration. Discussion of the authentication service used and defining which end points were needed for the system to function effectively both occurred here. Any links to relevant documentation were posted here, because many parts of the system were new to various members of the team, and this allowed us to communicate any useful information between us.
#frontend	Used for any frontend discussion, covering almost the same points as in backend, however with information relevant to the frontend team. Furthermore, any discussions on styling and specifics on the pages were also mentioned here.
#matching-system	Used by our data scientist to communicate their progress and developments to the rest of the team, as well as documenting the workings of the matching system to provide relevant team members with the information required to integrate their code with the matching system.
#report-writing	Used for the discussion on the documents which accompanied the project and the 'Dragons Den' video.
#github-webhook	This channel was used to track any Github updates, allowing the team to see if a change has been made to a branch that they may be currently working on.

Using these channels allowed clear communication between different members of the team, allowing us to effectively communicate where relevant, and quickly find any information relevant to them. Along with the pinning of messages, this meant that key information, links, and messages were always easy to access speeding up the process development.

Using the #general channel, we would aim to organise two to three meetings per week, one for the start of the sprint, and the others to discuss any other ongoing issues. Despite this, it was often difficult to find a time where all group members were able to be present due to varying timetables and commitments. To overcome this, a room would be booked where a voice call could be set up to allow non-present group members to communicate remotely.

8.4 Summary

In conclusion, the team communicated and worked well as a whole. Having two separate teams working in parallel meant that the development process was far more efficient, and each team member could specialise in the areas they were developing. This resulted in a team who had an intimate knowledge of the system, which vastly increased the pace of development and the fixing of errors. The use of pair programming allowed the teams to develop more technical parts of the system in tandem, which reduced the challenge of developing a complex part of the project alone. Despite this, there were times when both teams came across issues where the project manager's assistance was required as he had the best overall knowledge of the system.

Our choice of methodology was sensible, allowing us to have flexibility in our development. Even though the added stress of sprint planning and sprint retrospectives meant we had to adjust our Scrum cycle from one week to two weeks, having a regular cycle to follow helped with planning and following the goals set out for the project.

References

[1] https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs261/project/, 09/03/2022
[2] Niels Provos, David Mazier A Future Adaptable Password Scheme
https://www.openbsd.org/papers/bcrypt-paper.pdf, 12/03/2022
[3] .NET Foundation About xUnit.net
https://xunit.net/, 06/03/2022
[4] A Future Adaptable Password Scheme
https://www.postman.com/, 13/03/2022
[5] MDN Contributors MVC
https://developer.mozilla.org/en-US/docs/Glossary/MVC, 09/03/2022
[6] Use containers to Build, Share and Run your applications
https://www.docker.com/resources/what-container, 11/03/2022
[7] James Archbold Group Software Development Project
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs261/project/, 10/03/2022
[8] David Nichols Coloring for Colorblindness
https://davidmathlogic.com/colorblind/#%230018A8-%23FFFFFF-%23C6C8D6, 10/03/2022
[9] MUI https://mui.com/, 11/03/2022
[10] SCRUM https://www.scrum.org/resources/what-is-scrum, 11/03/2022
[11] SPA (Single-page application)
https://developer.mozilla.org/en-US/docs/Glossary/SPA, 10/03/2022
[12] https://owasp.org/www-community/attacks/csrf 08/03/2022
[13] https://reactrouter.com/ 11/03/2022
[14] Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides, John "Design Patterns: Elements of Reusable Object-Oriented Software" 21/10/1994 , Accessed: 23/01/2022