

2023

Student Dashboard

This report showcases the final conclusion and reports of a specialist software engineering project conducted in web the web development sphere and is a byproduct of a team's dedication and continuous efforts.

Report outline

01

The developments made to the original project outline.

The technical specifications in developing the project.

02

03

The final conclusions and portfolio of the project.



Report aims

This report aims to showcase the team building and management processes undertaken during the development of a student dashboard as part of a specialist engineering project. The goals of the assignment include highlighting modifications made to the original project outline, documenting technical specifications and the development process, and presenting a final portfolio of the website. The report provides insights into effective team collaboration, discusses key changes to the project plan, outlines technical aspects such as backend development and optimization techniques, and summarizes the completed student dashboard. It concludes with reflections on lessons learned and the project's potential for future expansion.

Table of contents

Plan development	4
Missed aspects from the original plan	4
Admin and executive accounts:	4
Deployment of the application:	4
Developments to the resources	5
Developments and changes to the tools:	5
Developments to programming tools utilized:	6
Delayed application features:	8
Estimations and risk management	9
Constraints:	9
Contingencies:	11
Team building processes undertaken	12
Management of task allocation:	12
How communication was established among the team:	12
Project management setup:	13
Communication with relevant parties:	13
Overall attitude and behavior:	14
Time allocation and timescale adjustments	14
Deviations from the outlined critical path:	15
Actual development timeline:	16
Technical Specifications	17
Development Process – Frontend UI and UX	18
Login page:	18
Home page:	19

A specialist engineering project

Student view:.....	21
Teacher view:.....	23
Development Process – Backend logic & APIs	26
Main classes and database:	26
APIs:	30
Cloud migration:	33
The graphing algorithm:	35
Major technical issues faced during development	38
Database dictionary mapping:.....	38
Potential improvements.....	39
Using docker and Kubernetes:.....	39
Better security measures:.....	40
Testing plan	41
Graphing algorithm:.....	41
Testing loading times:.....	41
Test of usability:.....	42
Note on security testing:	42
Final Portfolio	44
Conclusion	44
Expereience and technical effort:.....	44
Project's incomplection:.....	44
Website's pages.....	45
Landing and login pages:	45
Student view:.....	48
Teacher view:.....	51
Meeting minutes	53



Plan Developments



Missed aspects from the original plan

The following section will delve into the aspects that are currently unavailable in the application and explore the underlying reasons for their absence. Despite the comprehensive development process, it is important to acknowledge that no project is without its limitations and constraints. The aim of examining the missing aspects is to gain a deeper understanding of the challenges faced during the development phase and shed light on the factors that contributed to their exclusion.

Admin and executive accounts:

Excluded section:

One of the aspects that could not be fully implemented in the application is the inclusion of admin and executive accounts. Initially, there were plans to incorporate these account types to enhance the application's functionality. The admin account was intended to have privileges such as adding, removing, and editing student, teacher, or executive accounts. On the other hand, executive accounts were meant to have access to view all aspects of the database without the ability to make edits.

Reasoning:

The primary reason for excluding the admin and executive accounts was the constraint of time. As the project encompassed learning web development as a major component, the team's focus and efforts were primarily dedicated to acquiring the necessary skills and knowledge required for the project's implementation. This learning process significantly impacted the team's speed and overall production process, limiting the time available for developing complex features such as admin and executive accounts.

Given the time constraints, the team and the school supervising the project made the decision to prioritize the foundational aspects of the application, ensuring the core functionalities were implemented effectively. The emphasis was on creating a solid framework that could later be expanded upon and enhanced. Consequently, the development of admin and executive accounts was deprioritized in this iteration of the project.

While the inclusion of admin and executive accounts would have added valuable administrative capabilities and enhanced the application's functionality, the team recognized that focusing on the fundamental features and ensuring their robustness within the given timeframe was crucial. The exclusion of these account types allowed the team to allocate more time and resources to the essential components, ensuring a solid foundation for the application's future development and potential expansion.

Deployment of the application:

Excluded section:

One of the aspects that could not be fully realized in the project is the deployment phase. Deploying a web application involves the process of making the application accessible and available to users over the internet. It encompasses various tasks such as acquiring a domain name, setting up hosting infrastructure, configuring server environments, and managing data migration to the cloud, among others.

Reasoning:

The decision to exclude the deployment phase was influenced by several factors. Firstly, the school, acting as the project supervisor, considered it unnecessary at this stage of the project. The primary objective was to establish a solid foundation for the project, focusing on functionality and core features. The deployment phase, while important for the long-term success of the application, was deemed non-essential for its initial development.

Furthermore, the consideration of additional fees associated with acquiring a domain name and migrating to the cloud played a role in the exclusion. Given the project's scope and resources, the school and the team agreed that allocating these funds towards other critical aspects of the project would yield more substantial benefits.

Additionally, time constraints were a factor in the exclusion of the deployment phase. The project had a predetermined timeline, and completing essential components took precedence over the final deployment stage. By focusing on core features and functionality, the team aimed to deliver a robust foundation upon which the project could be further built and expanded in the future.

While the deployment phase was not realized in this iteration of the project, it remains an important consideration for future development and expansion, as it would provide the means to make the application accessible to a wider audience and ensure its seamless operation over the internet.

Developments to the resources

This section will discuss any developments points that did not deviate – or at least not significantly– from the initial development plan that was given in the second report and any additions that were implemented onto them. This includes the tools that were stated to be used and task allocation. After which, the features which were kept in would be discussed as well as the ones that are staying but are not possible to add due to time-related constraints.

Moreover, the section will go over any changes that were made to the technical specifications and justify why said changes improve the original plan or why it has been implemented as a risk management measure. It should also be noted that any features or tools not mention in remain unchanged relative to what was stated in the second report.

Developments and changes to the tools:

HTML and CSS:

Bootstrap: Bootstrap is CSS library that provides automatically generated CSS code which enhances the look and shape of default HTML elements such as buttons or tables. While creating a new Blazor WASM file project, bootstrap is automatically installed. As such, all that needed to be done was simply utilizing the predetermined CSS bootstrap classes on a given element for the CSS stylesheet to be applied as seen in figure 1.

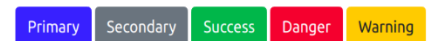


Figure 1 - Bootstrap button classes

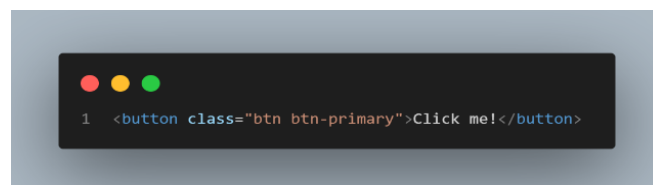


Figure 2 - bootstrap class example

An example of its usage can be seen in figure 2, which showcases a button having the “*btn btn-primary*” class being applied to it. Which changes the generic default HTML button to the one shown in the first button image shown in figure 1.

Emmet: Emmet on the other hand is a tool utilized specifically in order to enhance the development process of HTML. Emmet is a built in feature found in Visual Studio Code (which will be discussed shortly after), with its main usage being HTML auto completion. The way in which it functions is that instead of having to type entire lines of HTML code, there are shortcuts that could be written from each Emmet is able to complete the rest of the code for the developer. Although this might sound insignificant, however, it drastically enhances the overall development flow and decreases time wasted on simply typing.

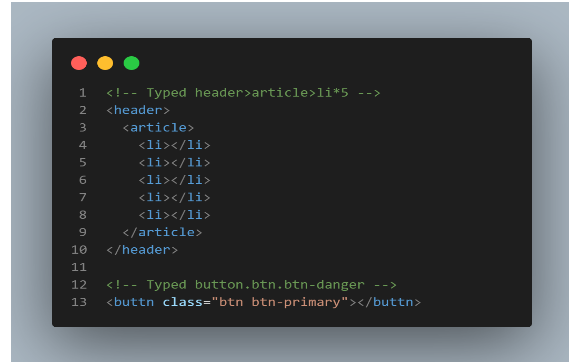


Figure 3 - Emmet example

JavaScript integration with C#:

Another tool that was added and is going to be utilized is the integration of native JavaScript functions into the application. This works due to a Dependency Injection (DI) in Blazor called *IJSRuntime*. An example of the way in which the DI can be used is with controlling the default alert message that shows up in the browser. The reason for that is due to JavaScript's native integration into web browsers allowing it to control the browser's controls and features, making this an extremely powerful tool.

The following code snippets showcase an example of the DI being used within a method. The showcased method returns a Boolean and within runs the DI in order to invoke a native JS method called "confirm" which causes a confirm box to appear on the user's browser and returns a Boolean with its value being dependent on what the user has chosen in the pop up box. From there the developers can use the returned value as needed in the application. Moreover, the *IJSRuntime* DI does not have to rely on integrated JS functions but rather it is possible to create custom functions to be used through referencing a ".js" file as a script source (`<script src="file location"/>`) in a file called `index.html` which is automatically created when a new Blazor WASM project is made.

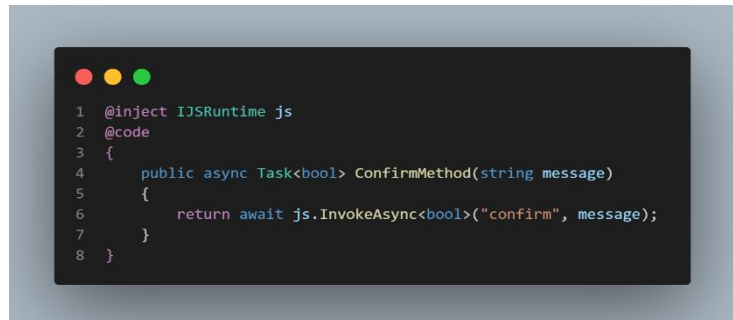


Figure 4 - Example of the IJSRuntime DI

Developments to programming tools utilized:

Switching from Visual Studio to Visual Studio Code:

A major divergence that was made from the original plan was the switch from utilizing Visual Studio (VS) as the main IDE to Visual Studio Code (VSC), which – unlike Visual Studio – is a text-editor rather than a fully fleshed out IDE. What this means is that it does not have the full range of development tools and capabilities such as an integrated debugger, compiler, NuGet package manager, etc.



Figure 5 - VSC vs VS

However, the main reason that the switch was made was due to the fact that VSC had a massive range of extension available that could be downloaded by simply navigating to the extension page. Due to VSC being open-source and extremely popular, the range of extension available on it is extremely wide and allows developers to basically turn a plain-text editor into an IDE

with the ability of further customizations. Not to mention that VSC is simply more friendly and less prone to restricting the user from accessing a given file or feature.

Some extensions that were added to the VSC are MSSQL's database viewer, a NuGet package manager and a GUI for it called NuGet Gallery. A debugger extension specific to Blazor, a C# code color highlighter. CodeSnap, allowing you to take easier and better-looking code snippets for sharing purposes. And Prettier, an easy-to-use code formatter. Some of the stated tools are unavailable in VS, not to mention the extremely easy Git integration.

Using Git and GitHub:

What is git: Git is what is known as a Version Control System or VCS, a tool that allows developers to create workspaces known as “repositories” or “repos” and save their work on their local machine through the use of the command line in a process known as committing. However, the major aspect of it is that with the use of git, developers are able to go back to previous “commits” and look at older version of the codebase. Moreover, developers can “branch” out of the main codebase, add changes to it that other developers in the same team cannot see. Meaning changes made in a different branch do not affect the main codebase and therefore different developers can work on different features without interfering with one another's work. It should also be noted that git utilized a special Command Line and shell scripting language known as Bash.



Figure 6 - git logo

Moreover, if a developer is going to work on a new feature that might potentially break the codebase, then they can simply branch out or create a new commit before working on said new feature and if it were to cause an error, then they are simply able to go back to the previous commit. This is but a few of the features that make git such a valuable tool for developers especially when working together as it allows for seamless operations even when multiple developers are working on the same codebase over different machines.

What is GitHub: GitHub on the other hand is a cloud-based web service that allows developers to upload their git repositories into the cloud. This is important for two main aspects. The first being that it is simply a processing of backing up the codebase in a location other than the local machine of the developer through a process that is seamless and easy to use as it can be done by simply calling the “push” command in git which uploads (“pushes”) the codebase into the GitHub repository where it is then stored in a secure cloud storage.



Figure 7 - GitHub logo

However, the main aspect that one might use GitHub for is not for simply backing up, but rather for collaboration. In that if two or more developers were to work together on a single codebase, said codebase needs to first be uploaded onto the cloud for them to be both be able to “pull” the codebase into their own machine, work on it, and then push it back into the cloud where both parties are able to see the newly edited codebase without interfering with one another's work.

How are they being used in the development environment: Both tools have been utilized during the development process in order to ease the process of working together especially in an environment where our working times may differ from one another. An example of how git and GitHub was used by the team during the development process is that two team members would work on two different features at the same time, the dashboard UI and the navigation bar. Naturally, either team member might have to make changes in the code that the other member might currently be working in order to get their feature to work. However, with git both developers can work and change each other's codebase without interfering with one another's work. When they are both done, both upload their work to GitHub and then the codebase can be changed accordingly during “merging” in order to get both features done. Basically, two features were being worked on at the same time, both got done together, and then both developers are able to merge their work into the main codebase on GitHub.

Delayed application features:

Light and dark mode:

Although having a dark and light mode toggle feature adds a sense of professionalism and quality to the website for the end user and is generally a loved feature, it is not a necessary and core feature that the application requires. Although it might seem like a simply feature to add, it does require quite a bit of consideration and change to the entire website in order to ensure that not only the background color is suitable, but the color of all related elements work nicely together and do not contrast one another in a way that makes the colors uncomfortable to look at. For that reason the feature will be delayed until a further time to be implemented.

English and Arabic languages:

Another feature that is not a core part of the application is a toggle between both English and Arabic languages in the application. This would need to be done as having to write all the text in the website in two languages while simultaneously working on the core features is going to be a major hinderance. As such, it is a feature that is going to be delayed until a further date. However, this does take precedence over the light and dark mode feature since – as stated in the second report – some teachers, parents, or younger students may not be proficient in English and having the website in Arabic would be the only way for them to use it effectively.

User Authentication:

Lastly, although having Two-Factor Authentication (2FA) is an extremely helpful feature when it comes to enhancing the security of the application, the development time that would be required for implementing it would be quite significant. And due to the feature not being a core aspect of the application, it would be ignored for the time being as to prioritize more important aspects of the website. However, due to its major support in increasing user's security, it would be a feature that needs to be implemented further down the development cycle of the application.

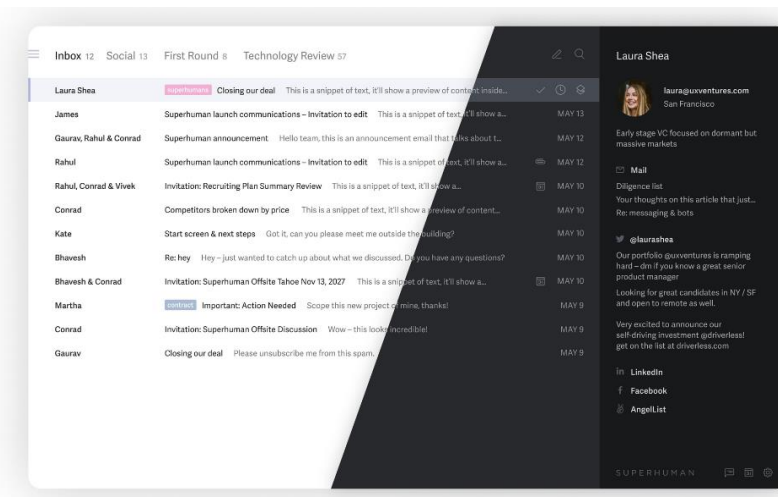


Figure 8 - Light and dark mode

Estimations and risk management

In this section of the planning report, it will be examined how the team tackled the project and addressed any potential issues that arose during or after the development process. The two main categories of issues are constraints and contingencies, each of which has specific sub-categories that fall under them.

In the previous report, several constraints and contingencies were identified, and different actions were prioritized based on how they were going to be addressed. These issues were thoroughly examined to determine the best course of action, and the team made significant progress in addressing them as they were faced during the development process of the application.

Constraints:

The constraints relate to the difficulties that might be faced due to limitations posed upon the team due to insufficient resources and are some of the immediately evident issues that the team had to tackle.

Budget:

No issues were encountered when purchasing the required courses, domain membership, and other necessary items mentioned in the previous report. This was due to the fact that the team had a good idea of all the necessary tools that might be used beforehand and as such no unexpected requirements occurred during the development which also required financial backing.

Moreover, the team was able to obtain all the required supplies with the assistance of the RGOTC teachers' team and executives. They purchased the necessary items on behalf of the team, eliminating the need for team members to spend their own money. This support from the RGOTC staff was greatly appreciated and contributed to the smooth progress of the project. For that reason, the mitigation plan – which initially involved switching to cheaper domain membership plans – was not utilized.

Expected risk	Actual outcome
Low	Low

Time constraint:

Brief: One of the most challenging and feared constraints that the team anticipated when embarking on the project was time constraint given upon the team. The team recognized that failing to complete the project on time could result in various problems, including a low score and an incomplete project. As such, the team was highly conscious of the need to manage their time effectively throughout the development process. They implemented several time-management strategies to ensure that they stayed on track with their progress.

Despite encountering some setbacks throughout the project, the team was able to meet the final deadline by implementing a mitigation plan that had been established from the beginning. By prioritizing tasks, setting realistic goals, and utilizing effective communication and project management software, the team was able to stay on track and work efficiently towards the project's completion.

The core aspects of the initial mitigation plan were still utilized, however, it was changed drastically in order to involve industry standard methodology utilized in larger software development teams as incorporating such methodologies turned out to be more beneficial to the overall productivity and performance of the team.

A specialist engineering project

Mitigation plan:

- **Agile Methodology:** Adopting an agile approach to project management, which involves breaking down the project into smaller, more manageable tasks that can be completed within shorter timeframes. Said timeframes are usually 2-to-3-week periods generally known as “sprints” in the industry. In said sprints the team members working on their task – while using version control – and by the end of the sprint the team holds a meeting discussing the progress that was made and any changes that need to be implemented. This approach allows teams to continuously review and adjust their plans based on feedback and changes in the project requirements, enabling them to remain responsive and adaptable to changing circumstances. It should still be noted that this was done alongside the smaller general daily and weekly meetings.
- **Iterative Development:** Adopting an iterative development approach, where the project is broken down into a series of smaller iterations that can be completed in shorter timeframes. This approach allows teams to test and refine their work continuously and adjust their plans as necessary to ensure that they remain on track to meet project goals.
- **Continuous Integration and Deployment:** Implementing continuous integration and deployment practices to automate the process of building, testing, and deploying software. This approach enables teams to deliver software more quickly and efficiently, reducing the time required to complete development tasks. This was done through using Git and GitHub and essentially making them an integral part of the development process. Which allowed the team to improve their productivity through continuously working with a version control system.
- **DevOps Practices:** Adopting DevOps practices, which involve integrating development and operations teams to improve collaboration and communication. This approach enables teams to identify and address potential issues early on in the development process, reducing the risk of delays or setbacks. A way in which this was implemented was having continuous updates on a person’s current progress multiple times throughout the day through direct communication or simply when adding a new commit the project’s repository.

Result: Even with using the mitigation plan, the time constraint was still a major issue and caused a fair amount of stress for the time. However, plan taken in order to mitigate the effects of the time constraint, the team was able to finish the majority of the project. Albeit slightly less than what was initially expected, however all the core and necessary features have nonetheless been implemented.

Expected risk	Actual outcome
Extreme	High

Scope:

The scope of a project refers to the specific requirements and features that define its boundaries and objectives. In the context of the project, the scope is quite broad and contains a significant number of features and requirements on both the frontend and backend. This causes a risk of the project not being completed within the allotted time. For that a number of points were outlined in the previous report in order to mitigate said risk.

The team faced some challenges with the scope during the project, but they were able to effectively address them by making changes to the features while still maintaining the functionality and the expected user experience. These changes were made through communication with the team and potential clients (i.e. students) to ensure that the updated features were within the project's scope. For that reason the mitigation plan initially highlighted was not needed as the team was able to handle the scope issues efficiently.

Expected risk	Actual outcome
Medium	Medium

Contingencies:

Loss of a team member:

Issue: During the course of the project, we encountered an unforeseen challenge when one of our team members, Ziyad Al-Saleh, fell seriously ill. This unexpected circumstance had the potential to disrupt our workflow and jeopardize our progress. However, we were able to rely on the mitigation plans that had been put in place to address such situations.

Mitigation plan:

- **Cross-training:** Ziyad took the time to explain his tasks and responsibilities to the rest of the team before his absence. As a result, we had a good understanding of his work and were able to step in and complete the remaining tasks in his absence. Additionally, since each team member had a basic knowledge of Ziyad's role, we were able to provide support and assistance to one another during this challenging period, with Ziyad himself also providing guidance remotely whenever possible.
- **Documentations:** Secondly, Ziyad had maintained thorough documentation of his work on the project, which was stored on Notion. This documentation proved to be an essential resource for the team. It provided insights into Ziyad's ideas, vision, and progress on his part of the project. Whenever issues were faced challenges or uncertainties, we referred to his documentation, which greatly helped the team stay on track and maintain the project's continuity.

Overall, the implementation of these mitigation plans played a crucial role in mitigating the impact of Ziyad's absence and ensuring that the project progressed smoothly. The experience highlighted the importance of teamwork, knowledge sharing, and comprehensive documentation, serving as a valuable lesson for future projects.

Expected risk	Actual outcome
Medium	Medium

Power outages:

As part of the team's risk management strategy, we had a contingency plan in place to address potential power outages during the development process. This plan aimed to mitigate the impact of power disruptions and ensure minimal disruption to our workflow. However, we were fortunate that we did not experience any power outages throughout the entire duration of the project. The reliable power supply allowed us to maintain uninterrupted progress and focus on our tasks without the need to activate the contingencies outlined in the plan. While we are grateful for this stroke of luck, the existence of the contingency plan provided us with a sense of preparedness and reassurance in the event that such an incident had occurred.

In anticipation of potential power outages or data loss, we implemented a robust backup strategy to safeguard the website and its development progress. One of the key measures taken was to store regular backups of the website in multiple locations, including GitHub. Mohammed Al-Hosni played a vital role in ensuring the website's safety by periodically backing up the project to GitHub. By doing so, we established a reliable off-site repository that would serve as a backup in the event of a power outage or any other unforeseen circumstances. This approach provided an additional layer of protection and gave us peace of mind, knowing that our hard work and progress were securely stored and could be easily recovered if needed.

Expected risk	Actual outcome
Low	Low

Team building processes undertaken

In this section, the progress and achievements of the development team would be explored, shedding light on the factors that contributed to their success from a mental and team development standpoint. Furthermore, the section will delve into the task management processes implemented by the team, as well as the communication channels established among team members and external stakeholders. By examining these aspects, the aim is to gain insights into the team's journey and the strategies they employed to foster effective collaboration and seamless communication.

Management of task allocation:

In this particular aspect of the section, the initial focus was on the distribution of tasks among team members and the preparation that took place to ensure a smooth workflow. However, the subsequent discussion will not revolve around these aspects, but rather on the team's performance in executing their assigned tasks.

Task allocation management is a crucial practice that involves assigning responsibilities to individuals or groups within a team. The successful completion of tasks and efficient resource utilization depend on effective task allocation management. This process encompasses several important steps. Firstly, identifying all the activities that need to be accomplished as part of a specific project or effort. This can be achieved by breaking down the main objective into smaller, more manageable tasks and determining their sequential order.

In line with these principles, the team developed their task allocation management processes as outlined in the previous report. They diligently identified and allocated tasks according to the established plan, ensuring that each task was completed in a timely and organized manner. By adhering to these management practices, the team optimized their performance and enhanced their overall productivity.

How communication was established among the team:

During the project implementation, effective communication played a pivotal role for the team. They successfully established communication channels within the team and with external stakeholders, including third parties. This ensured smooth verbal and written exchanges of information. Verbal communication involved regular team meetings, brainstorming sessions, and status updates, fostering collaborative decision-making. Written communication, such as emails, instant messaging, and project management tools, facilitated clear and concise correspondence with external parties.

Verbal communication:

The team adopted Discord as their primary online communication tool, leveraging its user-friendly interface and practical features. In addition to virtual interactions, they recognized the value of face-to-face communication and organized regular in-person meetings. These communication channels served as platforms for discussing various project-related matters, including the overall project scope, management processes, and the implementation strategy. As the project progressed, each team member shared updates on their respective tasks, fostering motivation and providing an opportunity for others to review the work completed. Furthermore, team members utilized this platform to seek assistance when encountering challenges, allowing for timely support and collaboration among team members. This communication approach facilitated effective coordination, knowledge exchange, and problem-solving throughout the project implementation phase. Especially when considering the

A specialist engineering project

fact that discord's "servers" could be easily organized by sections for each different aspect of the project, which made it much easier to communicate and work together as well as track the progress of different aspects of the project.

Written communication:

As for texting and typed communication, WhatsApp emerged as the preferred platform, especially when interacting with personnel outside the team. The team members recognized the professional nature of WhatsApp and found it to be more conducive for engaging with various third parties. This choice was driven by the ease of access to individuals such as the Head of IT Department, Supervisor, Rihal expert employee, and Executives, as it facilitated direct communication and streamlined discussions. Additionally, WhatsApp's voice messaging feature proved valuable in expediting communication, particularly when conveying detailed information or discussing complex points efficiently. The team utilized WhatsApp for essential communication tasks, including scheduling meeting times, sharing small but significant project details, and exchanging ideas for improving specific tasks.

Project management setup:

Notion proved to be an exceptional tool for our team, offering remarkable benefits throughout the development process. We embraced Notion as our primary collaborative workspace and utilized it extensively. The team was granted access to a dedicated Notion "teamspace" that served as a central hub for all project-related activities. Within this teamspace, we meticulously organized different sections to cater to each aspect of the development process, including frontend, backend, and data management. Notably, the teamspace prominently featured a calendar and a to-do list, both seamlessly integrated into Notion's platform. These integrated features empowered our team to effortlessly access and engage with essential scheduling and task management functionalities in a highly organized and collaborative manner.

Communication with relevant parties:

When it comes to communicating with third parties, such as the Head of IT Department, Supervisor, Rihal expert employee, and Executives, the team primarily engaged in meetings to facilitate effective communication. Each interaction served a specific purpose and yielded valuable outcomes, and by actively communicating with these third parties, the team leveraged their expertise, guidance, and suggestions to enhance the overall quality and success of the project.

Head of IT department:

The team recognized Ahmed Al Saleh, the Head of Information Technology and Systems at RGOTC, as an expert in web development. Therefore, they conducted meetings with him to seek advice and gather information regarding potential improvements, optimization strategies, implementation ideas, and troubleshooting errors. Ahmed Al Saleh's expertise and guidance were instrumental in shaping the project's direction.

Supervisor:

Mr. Abdulaziz Al Jabri, the project supervisor and an IT teacher with programming experience, played a crucial role in monitoring the team's progress. The meetings with the supervisor focused on regular work updates, discussions, project planning, leveraging his past experience, and receiving new ideas to enhance the project's outcomes. Mr. Abdulaziz Al Jabri's involvement ensured effective guidance and alignment with project goals.

Rihal expert employee:

The team sought the expertise of Waleed, a programming and web development expert, particularly for back-end coding matters. Planned meetings were held to address coding challenges and find solutions to errors that team members encountered. Waleed's assistance proved valuable in resolving complex technical issues and ensuring smooth progress in the project.

Executives:

Meetings with the Student Affairs department and Mr. Bader Al Nabhani, the Head of Student Affairs at RGOTC, were essential in incorporating valuable insights and feature suggestions into the project. The team engaged with them to discuss potential additions to the project and to validate the listed features. For instance, the concept of having multiple account types (students, teachers, executives, and admins) was proposed by the executives, adding depth and functionality to the project based on their input.

Overall attitude and behavior:

To foster a strong and cohesive team, it is imperative to monitor the behavior and attitude of team members and proactively address any potential issues that may arise. Failing to do so can leave the team vulnerable, making it fragile and susceptible to internal conflicts or undesirable behavior. By maintaining a vigilant approach to team dynamics, it becomes possible to identify and resolve any conflicts or challenges that could jeopardize the team's unity and effectiveness. Creating a supportive and respectful team environment, implementing effective communication strategies, and providing conflict resolution mechanisms are essential elements in building a robust and cohesive team capable of overcoming obstacles and achieving shared goals.

From the outset, the team benefited greatly from their long-standing friendship and years of shared experience, which contributed to the formation of a harmonious and cohesive unit. The team members demonstrated exemplary conduct both in written and verbal interactions, displaying friendliness, a positive attitude, and a remarkable absence of conflicts. Their exceptional teamwork was characterized by a spirit of cooperation and mutual support, as they readily assisted one another with tasks and provided motivation when needed. This culture of high moral conduct extended not only among the team members but also towards external individuals such as the Head of IT Department, the supervisor, teachers, and the members of the Student Affairs department. This positive and respectful approach facilitated effective collaboration and fostered strong relationships with key stakeholders involved in the project.

Time allocation and timescale adjustments

This section will discuss any deviations that occurred relative to the original timescale designed in the second report. It will be done through explaining what deviated from the original and justifying the reasoning behind the occurrences that lead to said deviations. This is done while showcasing the new development timeline that has naturally occurred when taking into account the aforementioned deviations.

Deviations from the outlined critical path:

Learning period:

The only part that significantly deviated from the initial timescale was the time expectancy for the learning period, where it was initially estimated to take around two months but in fact it took two months and a half for the full learning process, which was especially concerning initially.

The main reason for this is the fact that two of the three members, (Al-Hosni and Al-Farsi) felt extremely ill for a period that lasted about an entire week. Although they were nonetheless working and continuing their learning process, their progress – as would be expected – was extremely hindered and were only able to complete half of what was initially set out for that week.

Moreover, certain delays occurred due to errors that occurred during this period related to the installation and integration of MSSQL. Where the installer would run into an error in which the database service engine would fail during installation. Said issue occurred with Al-Hosni and Al-Saleh. This led to a delay of two days which were spent on researching the issue and trying out different solutions, many of which did not work.

The issue turned out to be an issue related to certain Solid-State Drives (SSDs) in which the disk sector size is given a size greater than 4095 bytes. The SSD that Al-Hosni and Al-Saleh were using had a size closer to 16k and that turned out to be the root cause of the issue. Fixing this required reconfiguring the SSD's sector size through the Powershell command terminal through a series of commands. The entire solutions can be seen in a Microsoft Documentation called "Troubleshoot errors related to system disk sector size greater than 4 KB".

Team member	Time taken to finish learning reqs.	Days above expected (49 – 62)
Al-Hosni	49 – 62	49 – 62
Al-Saleh	49 – 62	49 – 62
Al-Farsi	49 – 62	49 – 62

Frontend:

Another aspects that took longer than was initially expected was the time needed in developing and designing the frontend of the application. This was simply due to poor forecasting of the time and the challenges that would be required in frontend development, in that the team initially expected the process to be relatively easy where in fact it turned out to be the most challenging part of the project.

The reason for that is the frontend development process takes just as much time in the early stages of the wireframe drawing and choosing the most appropriate and aesthetic UI as it does in the development and implementation stage of the process, something that none of the team members expected to be the case before starting development.

Time taken to finish	Days above expected (39 – 53)
65	12

Backend:

Contrasting the two prior points, the time taken for developing the backend was in fact much shorter than what was initially expected, which was extremely important in allowing the team to finish the tasks in the appropriate manner and time considering the delays that occurred during the learning and frontend development stages. In fact,

A specialist engineering project

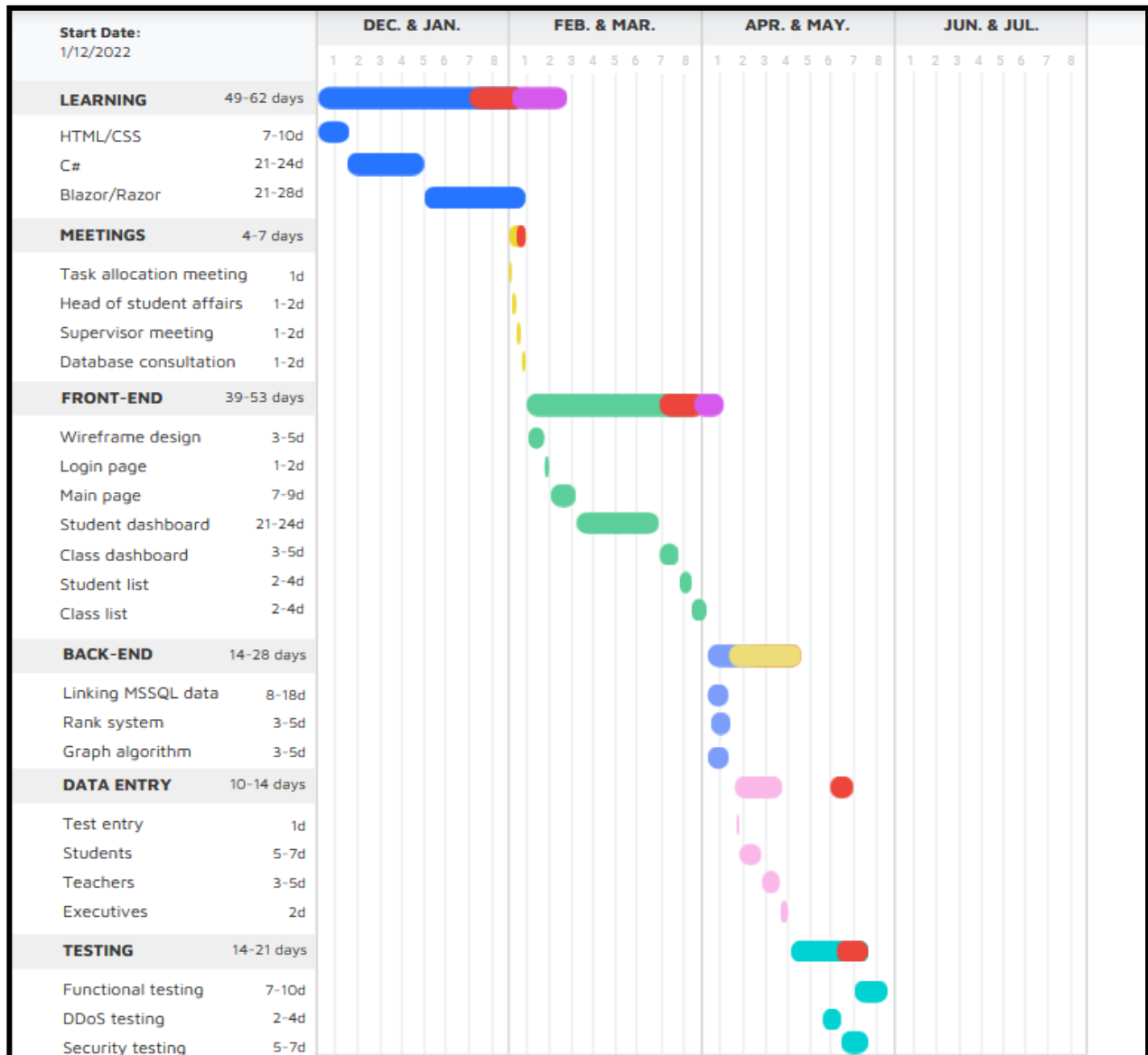
the process took us less than a week to complete in its entirety. Just for the frontend, this was caused due to a misinformed expectations of the requirements in backend developments, where in fact the process involved mostly a lot of boilerplate code that can be easily reused over multiple sections, some of which having little to no variation compared to what was initially learned during the learning process.

Time taken to finish	Days below expected (14 – 28)
5 days	9 days

Actual development timeline:

The way in which the delays were managed was that the workload was split over multiple team members in order to manage to keep up with the plan and work on multiple sections alongside one another even when time was overspent on a single section.

Purple = Unexpected delay Yellow = Unexpected decrease in work time





Technical specifications



Development Process – Frontend UI and UX

In the upcoming section, we will delve into the frontend development process of the website. This phase involved the creation and implementation of the user interface, user experience design, and interactive elements to deliver a visually appealing and user-friendly website. We will explore the technologies, frameworks, tools, and methodologies utilized by the frontend team to bring the project to life.

Note:

Due to the sheer amount of code that was written for the frontend, some aspects would be intentionally left out in order to avoid redundancy

Login page:

The frontend development process commenced by creating the login page, which serves as the initial point of entry for users visiting the website. Designing the login page proved to be straightforward since it encompasses a single page comprising form elements and div containers. The accompanying images showcase the login page's code structure, along with the containers, form, and text elements that constitute its layout. To ensure consistent styling throughout the website, a dedicated stylesheet was created, separate from the HTML file. This stylesheet was linked to the HTML file using the <link> element, enabling automatic application of the defined styles to the page.

```
1 </head>
2 <body>
3
4   <div class="limiter">
5     <div class="container-login100">
6       <div class="wrap-login100">
7         <div class="login100-pic js-tilt data-tilt">
8           
9         </div>
10
11         <form class="login100-form validate-form">
12           <span class="login100-form-title">
13             Member Login
14           </span>
15
16           <div class="wrap-input100 validate-input" data-validate = "Valid email is required: ex@abc.xyz">
17             <input class="input100" type="text" name="email" placeholder="Email">
18             <span class="focus-input100"></span>
19             <span class="symbol-input100">
20               <i class="fa fa-envelope" aria-hidden="true"></i>
21             </span>
22           </div>
23
24           <div class="wrap-input100 validate-input" data-validate = "Password is required">
25             <input class="input100" type="password" name="pass" placeholder="Password">
26             <span class="focus-input100"></span>
27             <span class="symbol-input100">
28               <i class="fa fa-lock" aria-hidden="true"></i>
29             </span>
30           </div>
31
32           <div class="container-login100-form-btn">
33             <button class="login100-form-btn">
34               Login
35             </button>
36           </div>
37
38           <div class="text-center p-t-12">
39             <span class="txt1">
40               Forgot
41             </span>
42             <a class="txt2" href="#">
43               Username / Password?
44             </a>
45           </div>
46
47           <div class="text-center p-t-16">
48             <a class="txt2" href="#">
49               Create your Account
50             <i class="fa fa-long-arrow-right m-l-5" aria-hidden="true"></i>
51             </a>
52           </div>
53         </form>
54       </div>
55     </div>
56   </div>
57
```

Figure 9 - Login page HTML code snippet

A specialist engineering project

Then, a simple script element was added to include a basic JavaScript function which allows for the image placed within the login page to tilt as the mouse hovers over it.

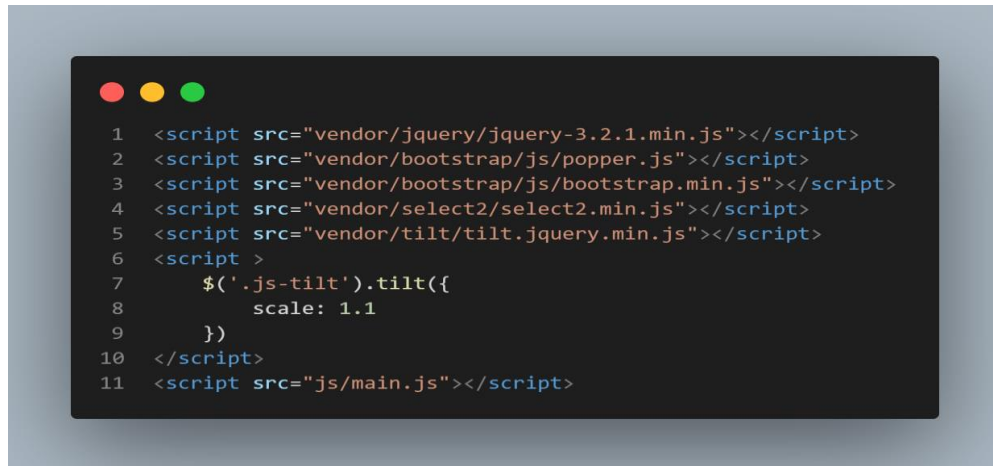


Figure 10 - Tilt script

Home page:

The provided code snippet represents the HTML structure of a web page. It begins with the declaration of the page route using the @page directive. The subsequent lines include the links to external CSS files for styling, such as normalize.css, all.min.css, and main_3.css. Additionally, the code includes links to Google Fonts for incorporating the "Cairo" font into the page.

The HTML code consists of several sections, each encapsulating different content on the web page. The "landing" section contains a header with a logo, page title, and navigation menu. It also includes a search icon for future functionality. The "service" section presents a list of offerings related to education, such as engineering skills, soft skills, business skills, and English language learning.

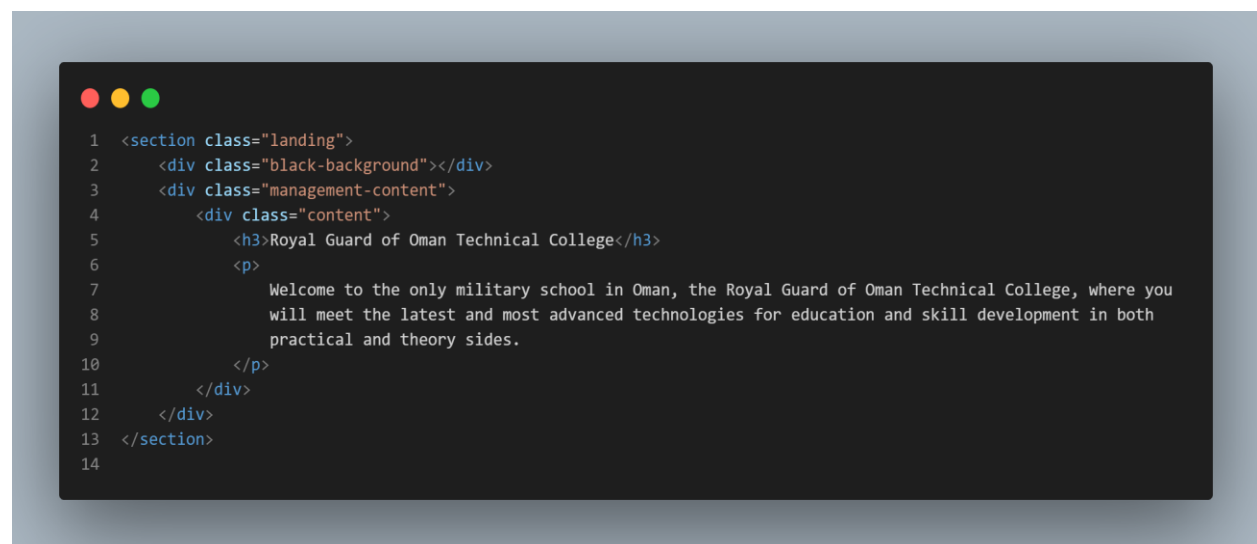


Figure 11 - Landing section

A specialist engineering project

Following that, there is a "Library" section showcasing various certificates, including Pearson BTEC, IELTS, IGCSE, and National Diploma. Each certificate is accompanied by an image, description, and additional information.

The code further includes a "Departments" section, displaying different departments like English, Arabic Studies, Mechanical Engineering, Electrical Engineering, Mathematics, Science, Information & Technology, and AutoCAD. Each department is represented by an image, title, and description.

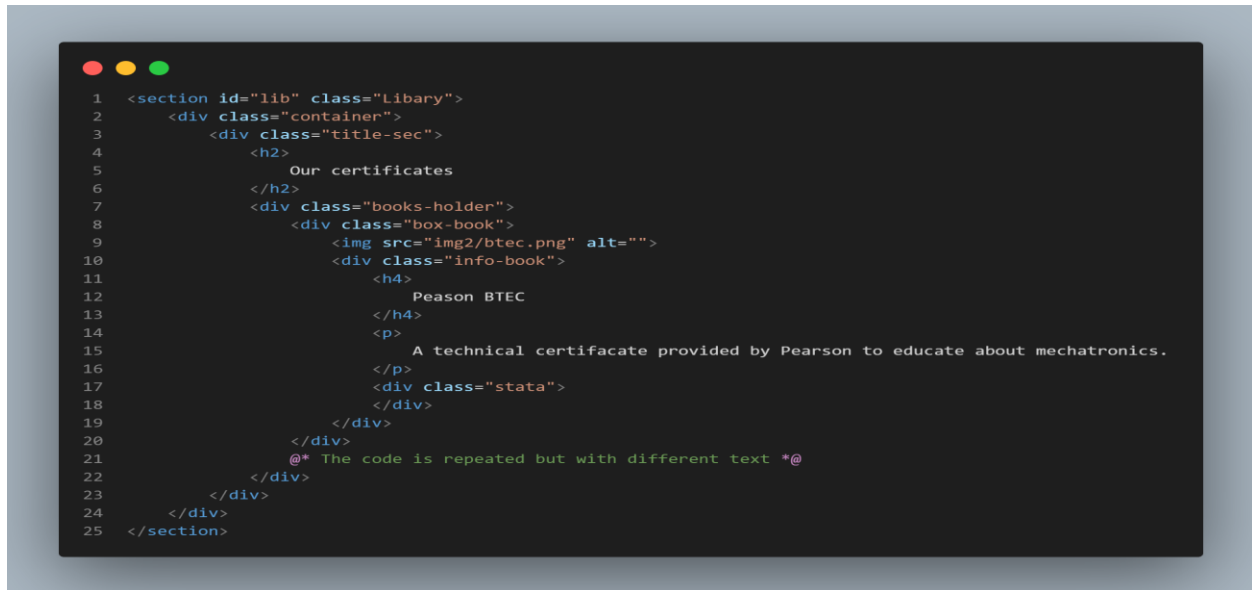


Figure 12 - Certificates showcase section

Lastly, the code concludes with a footer containing the website's logo, the RGOTC acronym, social media icons, and a copyright notice. In addition to the HTML code, there is a small code block written in C# using the `@inject` and `@code` directives. It defines two methods (`DashboardTravel()` and `TeacherTravel()`) that are invoked when clicking on specific links in the navigation menu. These methods utilize the `NavigationManager` to navigate to the corresponding pages ("`/Dashboard`" and "`/Teacher`") using the `NavigateTo()` method.

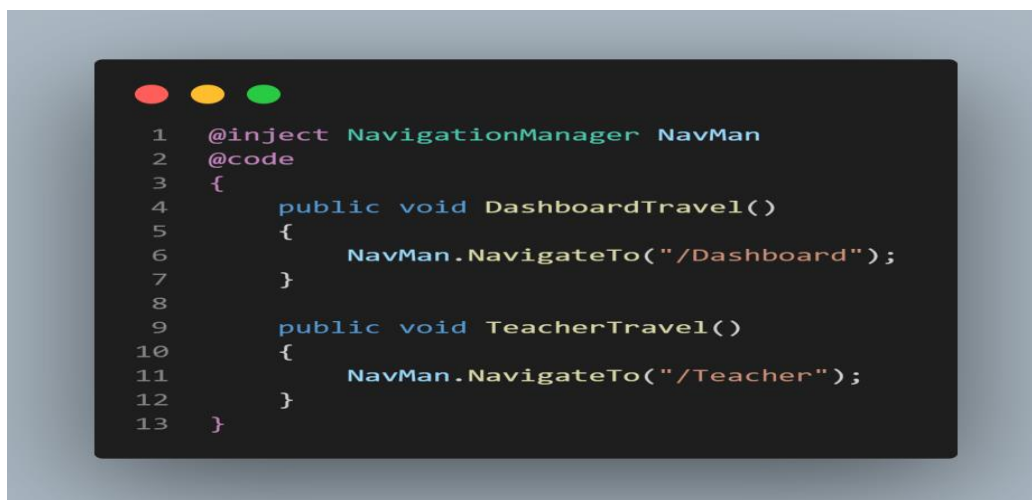


Figure 13 - C# code for navigation

Student view:

Main dashboard:

Following the completion of the login page, the focus shifted towards creating the student dashboard, which serves as the centerpiece of the website and fulfills the primary objective. The dashboard encompasses various elements, including graphs and information that were extensively discussed in meetings and previously documented in the project report.

To enhance user experience, a fixed left-hand-sided panel was implemented, enabling seamless navigation across different pages without the need to rely solely on the browser's back button. This design choice significantly improves user convenience and streamlines the overall browsing experience.



Figure 14 - Sidebar code

The student dashboard boasts an aesthetically pleasing layout, ensuring that information is presented in a visually appealing and easily comprehensible manner. This aspect is crucial for user satisfaction and ensures optimal usability. Users can readily access and interpret the displayed data, facilitating efficient monitoring of their academic progress.

Additionally, at the bottom of the page, a list of subjects was incorporated using flexbox, a layout model in CSS that provides a flexible and responsive design. The use of flexbox enables the list of subjects to adapt and adjust dynamically based on the available space and screen size. Furthermore, the subjects in the list feature a hover effect, indicating their clickable nature to users. When clicked, these subjects redirect users to dedicated pages showcasing their respective grades, allowing for a comprehensive and focused examination of subject-specific performance.

A specialist engineering project

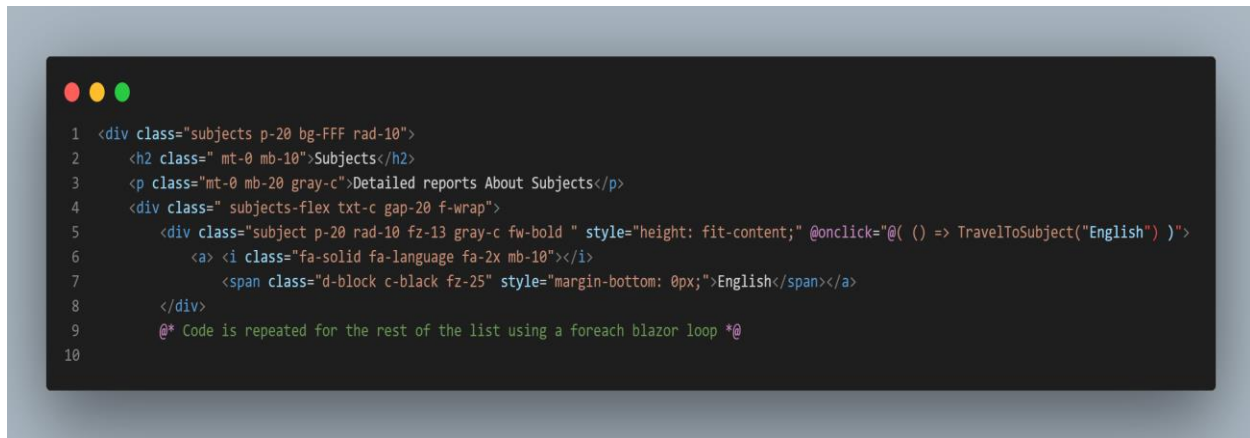


Figure 15 – List of subjects

Subject specific dashboard:

Within the student dashboard, a crucial component is the inclusion of subjects that the student is studying. These subjects are made clickable, allowing the designers to focus on developing comprehensive subject-specific pages to enrich the student's dashboard experience. An illustrative example of such a subject page is provided below.

The subject page exemplifies the meticulous implementation of the features and data outlined in the previous project report. As evident, the subject page presents information in a visually compelling manner, employing various visualization techniques. This approach enhances the readability and accessibility of student-specific information pertaining to the subject. By utilizing visualized data, students can effortlessly grasp their progress and performance within the subject, facilitating a deeper understanding of their academic journey.

In addition, the subject pages incorporate dynamic functionality achieved through the integration of C# code and JavaScript runtime Dependency Injection (DI). By leveraging C# code, the system examines the URL of the page using the Navigation Manager DI, allowing for targeted data retrieval and presentation. This intelligent routing mechanism ensures that the appropriate chart corresponding to the subject is dynamically rendered on the page. The seamless integration of C# and JavaScript runtime DI enables the system to deliver personalized and contextually relevant visualizations, enriching the student's interaction with the subject page.

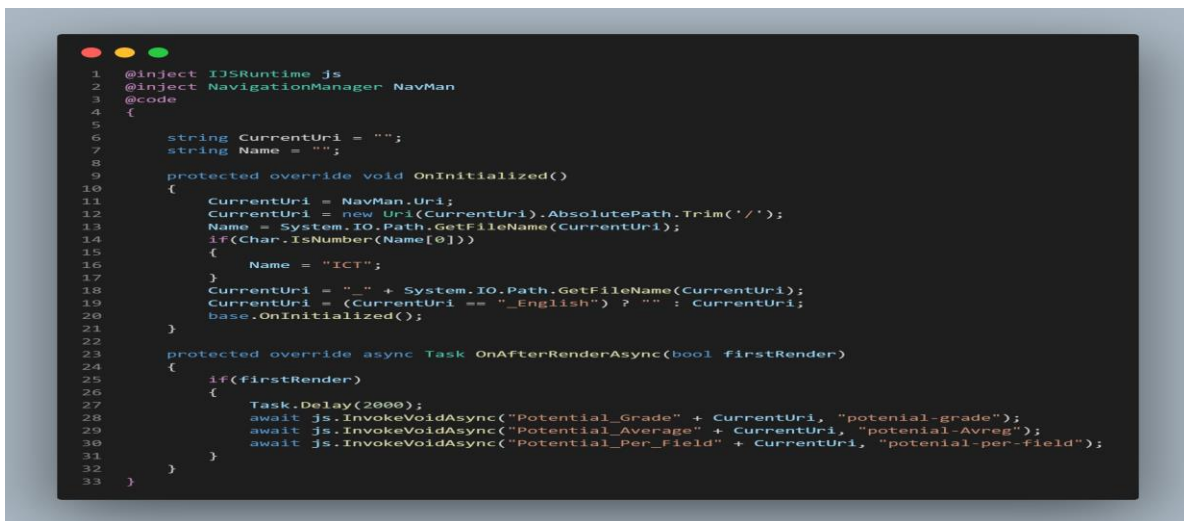


Figure 16 - Chart checking functionality

Teacher view:

The teachers' dashboard serves as the culminating feature of the website's dashboard project. Upon logging in as a teacher and accessing their dashboard, they are presented with a user-friendly interface that includes two essential buttons. The first button leads to the class list page, which provides an overview of the classes the teacher instructs. This page serves as a central hub for managing and organizing class-related information. The second button grants the teacher access to the student list, where they can view and analyze data about the students they teach. Through visually represented data, the teacher gains valuable insights into student performance and progress, facilitating informed decision-making and personalized instruction. This comprehensive teachers' dashboard empowers educators with the tools and information necessary to effectively manage their classes.

Teacher's landing page (Classes / Students option):

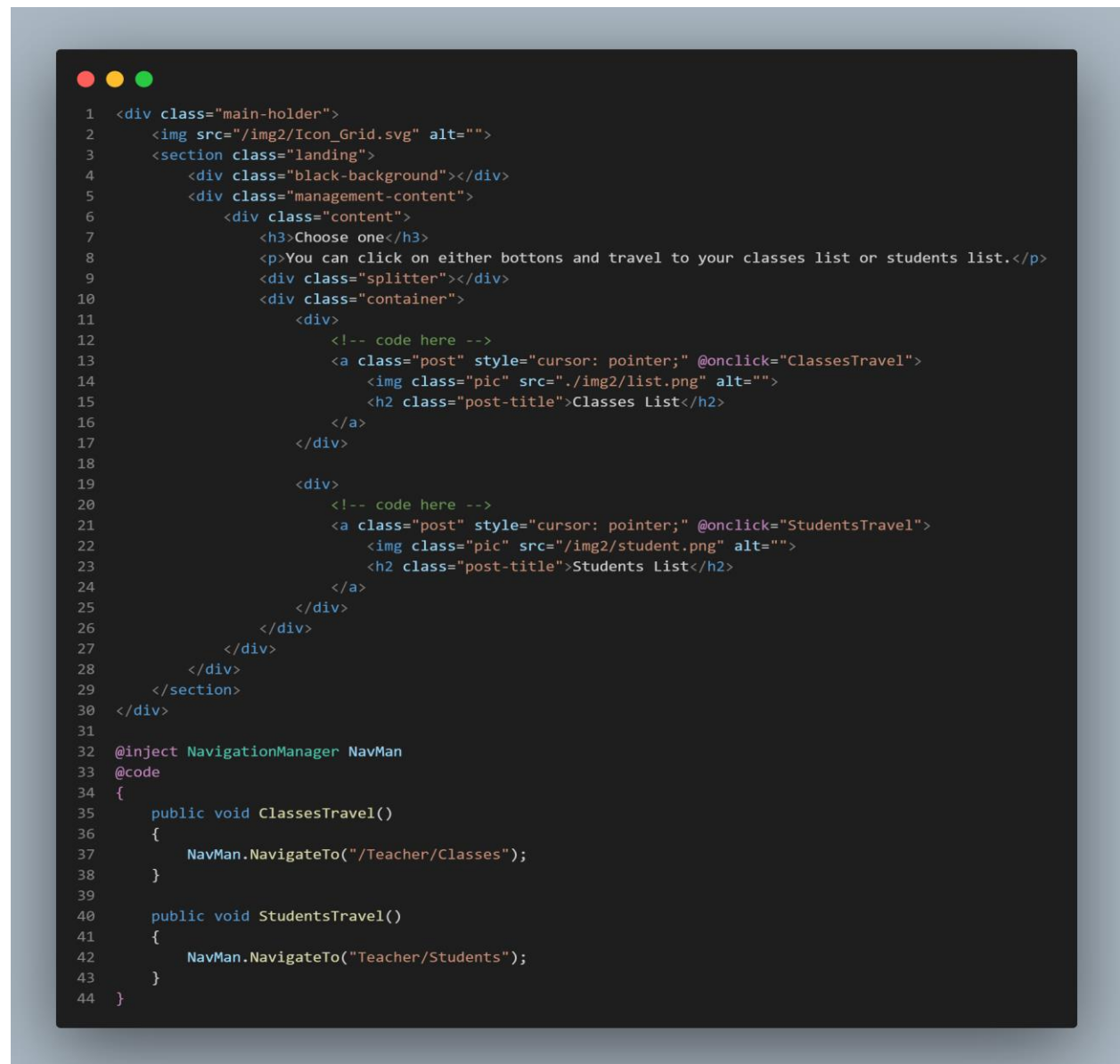


Figure 17 - Teacher's choices page

Classes list:

This code snippet figure 18 uses a Razor syntax used in a Blazor component. It iterates over a collection of classes (TheClasses.Classes) using a foreach loop. For each class, it generates an HTML div element with a CSS class of "box-book". The div element is clickable, and when clicked, it triggers the ClassTravel method with the class name as a parameter.

Inside the div element, there is an image, some information about the class (such as the class name), and a paragraph displaying the names of the first two students in the class. The ClassTravel method uses the NavigationManager service to navigate to a specific URL, which in this case includes the class name as part of the URL.

Overall, this code generates a dynamic list of clickable class boxes that allow teachers to navigate to individual class pages when clicked.



```
1      @foreach (var item in TheClasses.Classes)
2      {
3          <div class="box-book" @onclick="@() => ClassTravel(item.Name) ">
4              
5              <div class="info-book" style="cursor: pointer;">
6                  <h4>
7                      @item.Name
8                  </h4>
9                  <p style="color: black;">
10                     @item.StudentsList[0].Name, @item.StudentsList[1].Name
11                  </p>
12                  <div class="stata">
13                      </div>
14                  </div>
15              </div>
16          }
17      </div>
18  </div>
19  </div>
20
21  @inject NavigationManager NavMan
22  @code
23  {
24      public void ClassTravel(string Class)
25      {
26          NavMan.NavigateTo("Teacher/Classes/"+Class);
27      }
28  }
```

Figure 18 - Classes list page

Class view:

The class view is a page that showcases the students available within a given class that the teacher teaches. It is simply comprised of two charts and a list of the students. Said charts showcase the top three students within the class and a chart which separates the students based on performance. This allows the teacher to easily find out which students are performing poorly and which ones are doing alright, allowing them to easily understand which students require more help. The performance indicator is also available on the list of students itself next to each student's name.

A specialist engineering project

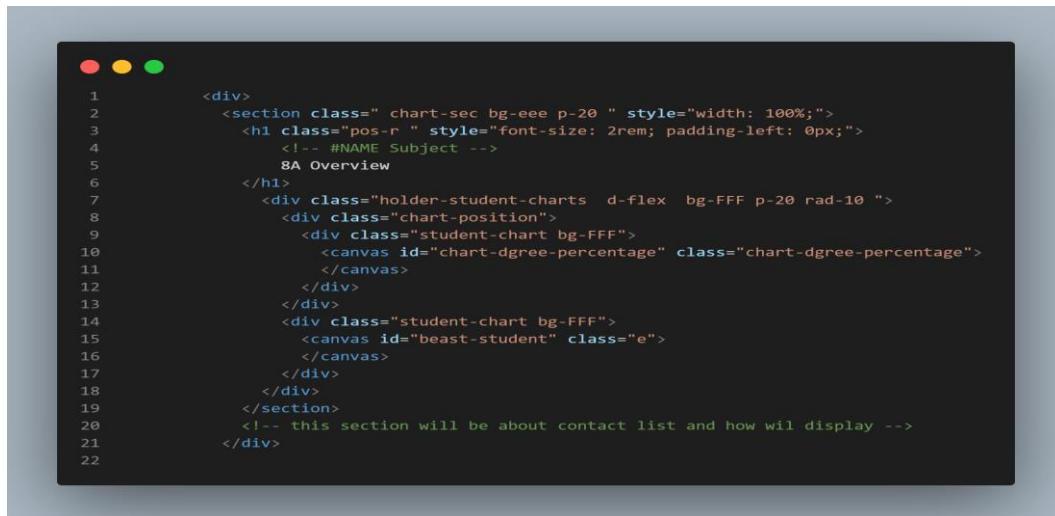


Figure 19 - The div holding the charts



Figure 20 - List of students in the class view

Students list:

This page simply showcases all the students that the teacher teaches, and is the page that the application navigates to when the teacher clicks on “Students list” on the landing page and essentially uses the same exact code snippet as the one shown in figure 19 to display the list of students, except that it uses a wider list of students rather than the one for that class specifically

Development Process – Backend logic & APIs

Here, the overall process related to working with the backend of the application will be discussed. This includes configuring and setting up the database and storing heavier data on the cloud automatically. Moreover, the development process for the Representational State Transfer Application Programming Interfaces (abbreviated as REST APIs) is going to be gone through as they are the standard methodology in computer networking for transferring data between the user's local machine and the server hosting the application.

Moreover, this section will go over the backend code used to run the graphing and visualization algorithms and the process and path taken in order to deploy the application to the public. Of course, it should be noted that this section will not encompass every single detail of the backend codebase. Since anything that is not shown in the frontend can technically be considered backend code, however, it will go through the major aspects of the project, not to mention that the fully documented and commented code has been open sourced and is available on the GitHub repository of the application (Found in the references).

Main classes and database:

Prerequisite to making a database:

Before being able to create an instance of a database, there are a couple of prerequisites that need to be met. This includes creating an instance of SQL Server by simply downloading it from the SQL Server web page alongside the SQL Server Management Studio. Certain NuGet packages (libraries) would need to be downloaded in order to interact with the database. Said NuGet packages will be discussed in the following section. Moreover, a Microsoft Entity Framework (database) tool would need to be installed for the Powershell command line in order to actually update and create the database. This is simply the process of setting up the database and getting the necessary tools to work with it.

However, the most important aspect is deciding on the datatypes that would actually go into the database. In order to do this, classes would need to be created which store the datatypes and names of the database tables as their variable properties. As such, the following two sections will discuss the two aforementioned points of setting up the database and classes.

Initializing the database:

NuGet packages to install: NuGet packages are libraries or open source tools that can be downloaded from the internet into one's project in order to add functionality that otherwise would be too complicated if added manually. For the application specifically, the NuGet packages that would need to be installed are ones related to manipulating and having control of the database. This would be done through a Microsoft made database framework and library known as Entity Framework Core (abbreviated as EFC). EFC gives us dedicated methods in order to transfer data from input in the codebase to the database through CLI functions. With that being said, the packages that need to be installed are the following:

Names	Purpose
Microsoft.EntityFrameworkCore	Provides the main classes for DB mapping
Microsoft.EntityFrameworkCore.SqlServer	Mapping functionality for MSSQL specifically
Microsoft.EntityFrameworkCore.Design	To update the DB after a migration is created

A specialist engineering project

From there, in order to initialize the database and update it with the newly created values, the entity framework Command Line Interface (CLI) tool had to be installed. This was simply done by typing the following command in Powershell (Window's CLI): (`dotnet install --global dotnet-ef`). Said tools allows the developer to initiate what is known as a database migration. Said migration is a C# class which contains all the information about the database such as information about every table found in the database including its rows and columns and its primary key as well as potentially foreign keys.

Setting up the connection string:

Each SQL database or account uses a special connection string that can be used in order for Entity Framework to identify to which SQL instance should a new table be created. This connection string can be copied right after the database is initially created.

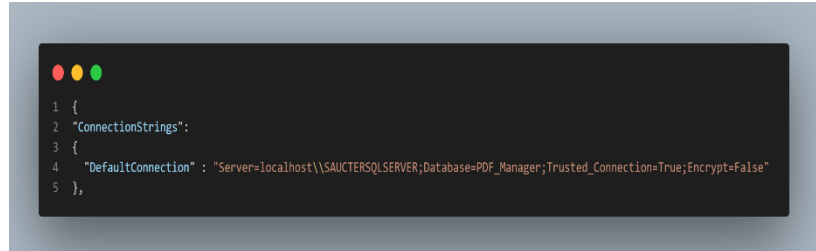


Figure 21 - Connection string

However, to actually use the connection string in the codebase two things need to be initially done. The first is going to the "appsettings.Development.json" file and adding a variable called "ConnectionStrings" to the json file, in which another variable is contained which can be named any suitable name. Said variable would then contain the connection string. This process can be seen in figure [number](#)

From there, the connection string would need to be referenced to Entity Framework Core with the following line in figure [number](#) of code being written in the "Program.cs" file which can be found in the Server directory of Blazor WASM.

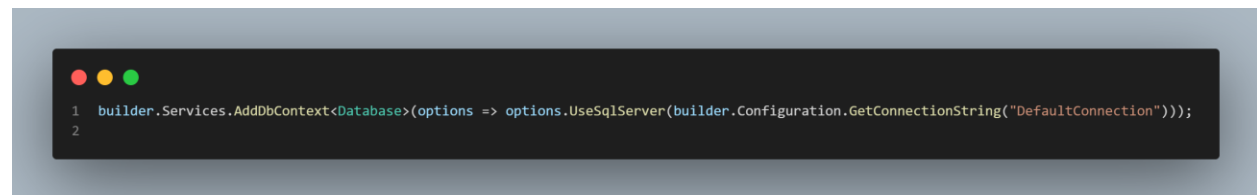


Figure 22 - Referencing connection string to EF

Setting up the classes:

Brief: In order to decide upon the different tables and the data (columns) that they are going to contain, classes need to be created, with each class being a distinct table and its variable properties being the columns that would be created in the database. As such, said classes need to encapsulate any data that would be needed further down the line during the development of other features in the application.

However, in the case that something was forgotten during the initial creation then that can be added later on, although this could cause issues with the previously inputted datasets as the added column would be "null" for them. Meaning that the functionality which requires the newly added information would not work with the old datasets. Meaning that choosing the correct properties and columns to include for each table is an important process that needs to be done carefully even during initial development.

Choosing datatypes: The code for the classes would be added within the "Shared" automatically created by Blazor WASM as the classes would be used both on the server and client side (Hence "Shared"). An example of how a class can be created with its properties can be below in figure [number](#).


```

1  //Libraries
2
3  namespace Dashboard_Project.Shared.Entities
4  {
5      public class Classes
6      {
7          protected readonly SubjectSelector Selector = new SubjectSelector();
8          public Classes(){}
9          public Classes(int Year, string Letter, DateTime EnrollmentDate, DateTime GraduationDate)
10         {
11             this.Year = Year;
12             this.Letter = $"{Year.ToString()}{Letter}";
13             this.EnrollmentDate = EnrollmentDate;
14             this.GraduationDate = GraduationDate;
15             ClassSubjects = Selector.Selector(Year);
16         }
17         public int id {get; set;}
18         public int Year;
19         public string Letter;
20         public DateTime EnrollmentDate {get; set;}
21         public DateTime GraduationDate {get; set;}
22         public List<Students> ClassStudents {get; set;}
23         public List<Teachers> ClassTeachers {get; set;}
24         public List<SubjectsList> ClassSubjects;
25         public List<TeachersClasses> TeachersClasses {get; set;} = new List<TeachersClasses>();
26     }
27 }

```

Figure 23 – Example of creating a class

The code snippet above showcases how a normal class might look like. This includes a constructor at the top which initiates the data when an instance of the class is created. Below that are the properties of the class. In the shown example, this includes an integer represent the year, a string represent the letter (8"A"/"B"/"C"), and other information that might used throughout the application. However, what has been shown is only that for that the Classes class which displays and gives the information for a given class in the school. But a separate class is also available for each table or set of information that might be needed for the functionality of the application. The properties of said classes are the following:

Classes and their properties	
Classes	Properties
Admin	int id – string name – string username – string password
Executive	int id – string name – string username – string password
Classes	int id – int Year – string Letter – DateTime EnrollmentDate – DateTime GraduationDate – List<Students> ClassStudents – List<Teachers> ClassTeachers – List<TeachersClasses> TeachersClasses
Teachers	int id – string name – string username – string password – List<TeachersClasses> TeachersAndClasses – List<Subjects> TeacherSubjects – List<Classes> TeachersClasses
Students	int id – string name – int CandidateNumber – string username – string password – Classes Class – List<Grades> Grades
Grades	int id – int Classwork / Behavior / Assignments / etc.
Subjects	int id – SubjectsList SubjectName – Grades Grade
TeachersClasses	int id – int TeacherID – int ClassID – Teachers Teacher – Classes Class

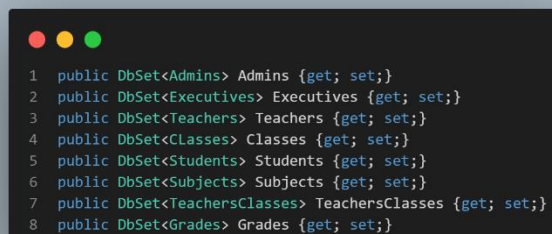
A specialist engineering project

Many-to-Many relationships: In software development, a term commonly used is “Many-To-Many relationships”. These indicate the relationship between a class and another. As an example if we have two classes. Class one has multiple instances of class two in its properties, while at the same time class two has multiple instances of class one in its properties. This is what is known as Many-To-Many relationship.

In order to indicate that there is a Many-To-Many relationship between two classes to Entity Framework core as it is mapping the database the developer would need to create an entirely new class (Meaning also a new table) which simply contains the IDs for both classes. And this is what can be seen in the TeachersClasses class, as it stores the ID for a Classes instance and for a Teachers instance as a single teacher can have multiple classes and a single class can have multiple teachers.

Adding the migrations and updating the database:

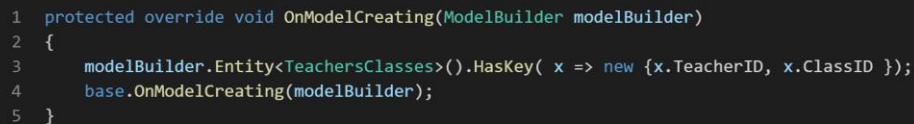
Adding the database class: Before adding the tables into the MSSQL database, a new class would need to be created. Said class would utilize the Microsoft.EntityFrameworkCore NuGet package in its library imports. From there it would inherit from a class called “DbContext”, a class that was provided by the NuGet package. This class would need to contain mainly two things. The first being an a set of properties which have “DbSet” as their datatype. Said datatypes are generic and take references to the classes that need to be added to the database. This can be seen in figure number. This tells Entity Framework the classes that need to be added as tables into the database



```
1 public DbSet<Admins> Admins {get; set;}
2 public DbSet<Executives> Executives {get; set;}
3 public DbSet<Teachers> Teachers {get; set;}
4 public DbSet<Classes> Classes {get; set;}
5 public DbSet<Students> Students {get; set;}
6 public DbSet<Subjects> Subjects {get; set;}
7 public DbSet<TeachersClasses> TeachersClasses {get; set;}
8 public DbSet<Grades> Grades {get; set;}
```

Figure 24 - Adding classes to the DB

Setting the Many-To-Many relationship: In order to indicate to Entity Framework that the class being added is simply used as a connector for a Many-To-Many relationship, a method from the DbContext class needs to be added and overridden using polymorphism. This method takes in a special class called “ModelBuilder” as its parameter and what through said class it is possible to add the Many-To-Many relationship indicator using the following code:



```
1 protected override void OnModelCreating(ModelBuilder modelBuilder)
2 {
3     modelBuilder.Entity<TeachersClasses>().HasKey( x => new {x.TeacherID, x.ClassID });
4     base.OnModelCreating(modelBuilder);
5 }
```

Figure 25 - Using OnModelCreating method to create special class relationship

What the code essentially does is that it indicate that the TeachersClasses class actually has two different IDs which need to be updated automatically, rather than the traditional “id” property, with said IDs being the TeacherID and ClassID.

Final steps: Lastly, in order to add the database, two command need to be ran in the terminal. With them being “dotnet ef migrations add Initial“, which would automatically create a new file called “Migrations” and add a class to it which would contain the code that Entity framework core would read when mapping the database. Basically, what this does is that it writes the class for the developer automatically. From there, simply running “dotnet ef database update“ should update the MSSQL DB that was linked with the mapped DB.

APIs:

API definition and meaning:

An Application Programming Interface (API) is a key component of your project, facilitating the seamless transfer of data between the database and the browser. It serves as a standardized set of rules and protocols that govern the communication and information exchange between various software components.

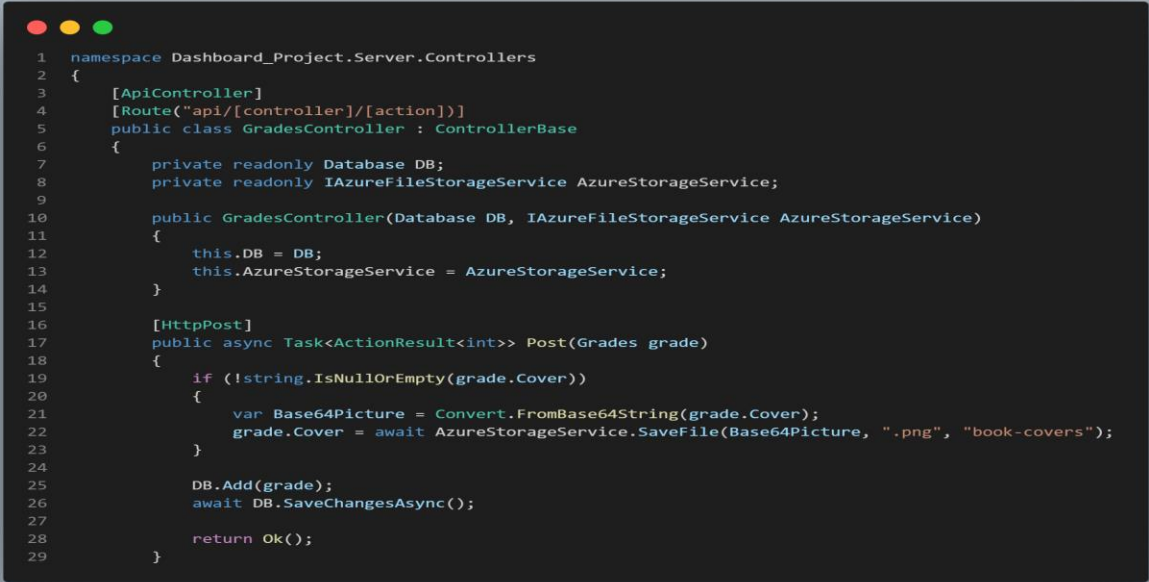
The API employs the HTTP protocol, a widely adopted standard for data communication on the web. It leverages the HTTP methods such as GET, POST, PUT, and DELETE to enable the front-end application to request specific data or perform operations on the server-side. The API defines clear endpoints that correspond to specific functionalities or data entities within the project.

Through the API, the front-end application can send well-structured HTTP requests to the server, specifying the desired data or operations. The API processes these requests, accessing the relevant data from the database, and subsequently transmits the response back to the front-end application.

The way that an API or HTTP Protocol works is that it creates a specific URL that can be searched for on the web. When this URL is utilized over a given browser, data – in the form of a JSON string – is sent to the database, which then checks the validity of the request. If the request is valid, then it returns the requested data from the database based on the functionality that the API provides. Otherwise, it would return an error specified by the developer.

Implementing API on the server side:

Incorporating the API into the server side involves the creation of classes known as “controllers”. Said controller create the varying URL in which the API would function from. Since – as stated – the API function by transmitting data by searching for a specific URL over a browser. For that, code written in the controller class using the Entity framework core library is used to facilitate the search point and the URL of the API, and to control how the requested and sent data is handled.



```
1 namespace Dashboard_Project.Server.Controllers
2 {
3     [ApiController]
4     [Route("api/[controller]/[action]")]
5     public class GradesController : ControllerBase
6     {
7         private readonly Database DB;
8         private readonly IAzureFileStorageService AzureStorageService;
9
10        public GradesController(Database DB, IAzureFileStorageService AzureStorageService)
11        {
12            this.DB = DB;
13            this.AzureStorageService = AzureStorageService;
14        }
15
16        [HttpPost]
17        public async Task<ActionResult<int>> Post(Grades grade)
18        {
19            if (!string.IsNullOrEmpty(grade.Cover))
20            {
21                var Base64Picture = Convert.FromBase64String(grade.Cover);
22                grade.Cover = await AzureStorageService.SaveFile(Base64Picture, ".png", "book-covers");
23            }
24
25            DB.Add(grade);
26            await DB.SaveChangesAsync();
27
28            return Ok();
29        }
30    }
```

Figure 26 - API Controller example

A specialist engineering project

The shown code snippet is an example of a controller might look like when using C# and Entity framework core. The constructor in the class initializes any variable that would be used, and then after that varying HTTP protocols are utilized in order to create the required URL. For example, the first one is when a new grade is posted. The URL for that one would be the name of the website, followed by the “Grades” and followed by “Post”. Anything sent to that URL would go into this API, and in turn follow the logic of the code, which adds a new grade to the database and saves it. Moreover, other APIs are used in the same controller such as one for deleting and retrieving back grades (using `HttpDelete` and `HttpGet`). The same process would be done any other aspects of the project such as students or admins.

Implementing API on the client side:

Although an API has been implemented on the backend, the client does need a way to communicate with the server. This is where `Http Services` come in play through the use of a class within the C# and ASP.NET Core framework called `HttpContext` which allows for calling and receiving data in JSON format to the client side. A class is created which uses the functionality of the `HttpContext` class alongside other required logic to make the code functional. Moreover, another class is created and is known as a “repository”, said repository is created for each one of the classes or values that would need to be edited in the front-end. This repository is used to actually call the aforementioned class and ensure that the specific required API is utilized.

```
1 namespace blazorTestApp.Client.Classes_FE
2 {
3     public class HttpService : IHttpService
4     {
5         private readonly HttpClient httpClient;
6         private JsonSerializerOptions defaultJsonSerializerOptions => new JsonSerializerOptions {PropertyNameCaseInsensitive = true};
7         public HttpService(HttpClient httpClient)
8         {
9             this.httpClient = httpClient;
10        }
11
12        public async Task<HttpResponse<T>> GET<T>(string url)
13        {
14            var response = await httpClient.GetAsync(url);
15            if(response.IsSuccessStatusCode)
16            {
17                var DeserializedResponse = await Deserialize<T>(response, defaultJsonSerializerOptions);
18                return new HttpResponse<T>(DeserializedResponse, response, true);
19            }
20            else
21            {
22                return new HttpResponse<T>(default, response, false);
23            }
24        }
25
26        public async Task<HttpResponse<object>> POST<T>(string url, T data)
27        {
28            var JsonData = JsonSerializer.Serialize(data);
29            var StringContent = new StringContent(JsonData, Encoding.UTF8, "application/json");
30            var ResponseMessage = await httpClient.PostAsync(url, StringContent);
31            return new HttpResponse<object>(null, ResponseMessage, ResponseMessage.IsSuccessStatusCode);
32        }
33
34        public async Task<HttpResponse<object>> PostGeneric<T, TResponse>(string url, T data)
35        {
36            var JsonData = JsonSerializer.Serialize(data);
37            var StringContent = new StringContent(JsonData, Encoding.UTF8, "application/json");
38            var ResponseMessage = await httpClient.PostAsync(url, StringContent);
39            if(ResponseMessage.IsSuccessStatusCode)
40            {
41                var responseDeserializer = await Deserialize<TResponse>(ResponseMessage, defaultJsonSerializerOptions);
42                return new HttpResponse<object>(responseDeserializer, ResponseMessage, true);
43            }
44            else
45            {
46                return new HttpResponse<object>(default, ResponseMessage, false);
47            }
48        }
49
50        private async Task<T> Deserialize<T>(HttpResponseMessage responseMessage, JsonSerializerOptions options)
51        {
52            var responseString = await responseMessage.Content.ReadAsStringAsync();
53            return JsonSerializer.Deserialize<T>(responseString, options);
54        }
55    }
56 }
```

Figure 27 - Http Service class

A specialist engineering project


The `HttpService` class is responsible for handling HTTP requests from the client side to the API using the HTTP protocol. It encapsulates the functionality of sending GET and POST requests and handling the responses. The class utilizes the `HttpClient` class for making the actual HTTP requests.

In the GET method, the `HttpService` sends a GET request to the specified URL and checks if the response is successful. If the response is successful, the received data is deserialized using JSON deserialization and returned as a `HttpResponse<T>` object, indicating a successful response. If the response is not successful, a `HttpResponse<T>` object is returned with a default value, indicating a failed response.

The POST method sends a POST request to the specified URL with the provided data. The data is serialized to JSON and sent as the request payload. The method then returns a `HttpResponse<object>` object indicating the success or failure of the request.

The `PostGeneric` method is similar to the POST method but is used for cases where a specific response type is expected. It sends a POST request, deserializes the response body to the specified `TResponse` type using JSON deserialization, and returns a `HttpResponse<object>` object with the deserialized response and the success status.

The `Deserialize` method is a helper method that performs JSON deserialization of the response body using the provided `JsonSerializerOptions`.



```
1 namespace Dashboard_Project.Client.Repositories
2 {
3     public class GradesRepo
4     {
5         public IHttpService httpService;
6         public string url = "api/Grades";
7
8         public GradesRepo(IHttpService httpService)
9         {
10             this.httpService = httpService;
11         }
12
13         public async Task<object> AddGrade(Grade grade)
14         {
15             var response = await httpService.PostGeneric<Grade, int>(url, grade);
16             if (!response.Success)
17             {
18                 throw new ApplicationException(await response.GetBodyString());
19             }
20             return response.Response;
21         }
22
23         public async Task<object> GetGrades()
24         {
25             var response = await httpService.GET<List<Grade>>(url);
26             if (!response.Success)
27             {
28                 throw new ApplicationException(await response.GetBodyString());
29             }
30             return response.Response;
31         }
32     }
33 }
34
```

Figure 28 - Grades repository class

A specialist engineering project

The `GradesRepo` class serves as a repository for interacting with the API's Grades resource. It relies on the `IHttpService` interface to communicate with the API using the defined HTTP methods.

The class contains methods such as `AddGrade` and `GetGrades` that correspond to the operations related to grades. The `AddGrade` method sends a POST request to the specified URL with the provided `Grade` object as the payload. It uses the `PostGeneric` method of the `httpService` to handle the request and obtain the response. If the response is successful, the method returns the response, indicating the added grade. Otherwise, an exception is thrown.

The `GetGrades` method sends a GET request to the specified URL to retrieve a list of grades. It uses the `GET` method of the `httpService` to handle the request and obtain the response. If the response is successful, the method returns the response, indicating the list of grades. Otherwise, an exception is thrown.

These repository methods encapsulate the API calls and provide a simplified interface for interacting with the Grades resource, abstracting away the details of HTTP communication.

Cloud migration:

Database with Azure DB:

Migrating an MSSQL database from a local environment to Azure is a relatively straightforward process that involves a few key steps. The first step is to launch SQL Server Management Studio (SSMS), a powerful tool for managing SQL Server databases. Within SSMS, there is a built-in Migration Wizard that simplifies the migration process. By following the prompts and providing necessary information, such as the source database details and Azure destination settings, the wizard guides users through the migration process.

Once the migration wizard completes the transfer of the local database to Azure, the next step is to update the application's connection string. The connection string is a crucial configuration setting that specifies the database's location and other connection parameters. In this case, the connection string needs to be updated to reflect the Azure database's details, such as the server name, database name, and security credentials. By replacing the local database's connection string with the Azure database's connection string, the application will now connect to the migrated database hosted in the Azure cloud environment.

Images with Azure storage:

To store images and other files in a scalable and efficient manner, object storage is commonly used. Object storage is a type of cloud storage that stores data as objects rather than in a traditional file hierarchy. It offers several advantages, including virtually limitless storage capacity, high durability, and easy access to data through HTTP-based protocols. In the context of a website, object storage can be used to store various images, such as student photos, that are required for displaying content on the site.

To utilize object storage in Azure, the first step is to create an object storage account using the Azure portal. Once the storage account is set up, the next step involves utilizing an Azure NuGet package that provides the necessary APIs for interacting with the object storage. In the provided code, the class `AzureFileStorageService` is responsible for handling the file storage operations.

```
1 using Azure.Storage.Blobs;
2 namespace blazorTestApp.Server.Helpers
3 {
4     public class AzureFileStorageService : IAzureFileStorageService
5     {
6         private readonly string ConnectionString;
7
8         public AzureFileStorageService(IConfiguration Config)
9         {
10             ConnectionString = Config.GetConnectionString("AzureConnectionStorage");
11         }
12
13         public async Task<string> SaveFile(byte[] Content, string extension, string ContainerName)
14         {
15             var Client = new BlobContainerClient(ConnectionString, ContainerName);
16             await Client.CreateIfNotExistsAsync();
17             Client.SetAccessPolicy(Azure.Storage.Blobs.Models.PublicAccessType.Blob);
18             var FileName = $"{Guid.NewGuid()}({extension})";
19             var blob = Client.GetBlobClient(FileName);
20             using (var ms = new MemoryStream(Content))
21             {
22                 await blob.UploadAsync(ms);
23             }
24             return blob.Uri.ToString();
25         }
26
27         public async Task DeleteFile(string ContainerName, string FileRoute)
28         {
29             if(!string.IsNullOrEmpty(FileRoute))
30             {
31                 var Client = new BlobContainerClient(ConnectionString, ContainerName);
32                 var blob = Client.GetBlobClient(Path.GetFileName(FileRoute));
33                 await blob.DeleteIfExistsAsync();
34             }
35         }
36
37         public async Task<string> EditFile(byte[] Content, string extension, string ContainerName, string FileRoute)
38         {
39             await DeleteFile(ContainerName, FileRoute);
40             return await SaveFile(Content, extension, ContainerName);
41         }
42     }
43 }
```

Figure 29 - Azure storage class

The `AzureFileStorageService` class accepts an `IConfiguration` object in its constructor, which allows it to retrieve the connection string for the Azure object storage from the application's configuration. The `SaveFile` method takes in the file content as a byte array, the file extension, and the container name where the file will be stored. It creates a `BlobContainerClient` using the provided connection string and container name, ensuring that the container exists. The method then generates a unique file name, creates a `BlobClient` for the file, and uploads the file content to the blob storage using the `UploadAsync` method. Finally, the method returns the URI of the uploaded file.

The `DeleteFile` method removes a file from the storage. It takes the container name and the file route (or path) as input, retrieves the corresponding `BlobClient`, and deletes the file using the `DeleteIfExistsAsync` method.

The `EditFile` method allows for updating a file in the storage. It first deletes the existing file using the `DeleteFile` method, and then calls the `SaveFile` method to upload the updated file content with the specified file extension and container name.

The graphing algorithm:

Switch from Blazorise to Charts.JS:

The decision to switch from Blazorise components to Charts.js for creating charts was driven by several advantages offered by the Charts.js library. Charts.js not only provides a more comprehensive and feature-rich charting library with a wide range of chart types and customization options, but it also offers support for various types of charts such as bar charts, line charts, pie charts, and more. This allows for greater flexibility in visualizing data and presenting it in a more meaningful way. Where in comparison, Blazorise has a much more limited selection.

Another advantage of using Charts.js stems from its overall simplicity and ease of use. The library provides a straightforward API that simplifies the process of creating and configuring charts. Moreover, it contains intuitive options for setting labels, data points, colors, and other chart properties. Making it easier to integrate charts into the applications without significant complexity.

Additionally, Charts.js has a thriving community and extensive documentation, making it easier to find resources, examples, and support when working with the library. The availability of community-driven plugins and extensions further enhances the capabilities of Charts.js, allowing for additional chart features and functionality.

Although extra steps would need to be taken due to the fact that Chart.JS is written in JS, JS to C# interoperability is possible through the use of the IJSRuntime dependency injection.

JS interoperability:

As mentioned, an additional step would need to be taken in order to use Charts.JS due to the interoperability issues related to the fact that Blazorize runs using C# rather than JS, and simply utilizing the “script” HTML element is not possible within Blazor components (.razor files).

To allow for interoperability, the JS code would need to be written as a function within a separate .js file rather than the .razor file. However, the function would need to be written as shown in figure 17, where the name after the “window” keyword is the name of the function. This is done opposed to the traditional way which simply uses the “function” keyword followed by the name.

A screenshot of a code editor with a dark background and light-colored text. The code is as follows:

```
1 window.my_function = (message) =>
2 {
3     console.log("My function: " + message);
4 };
```

The code is numbered 1 through 4 on the left side of the editor. The editor has three colored dots (red, yellow, green) in the top left corner.

Figure 30 - JS function writing method

After this has been done, the “IJSRuntime” dependency injection would need to be injected into the .razor file that the chart should be added in. After injecting the scoped dependency and then initializing it with a variable name, it can then be used as shown in figure 18 to actually call the method, where the first string is the name of the function and the second one and whatever follows is the parameters that the function takes.

A specialist engineering project

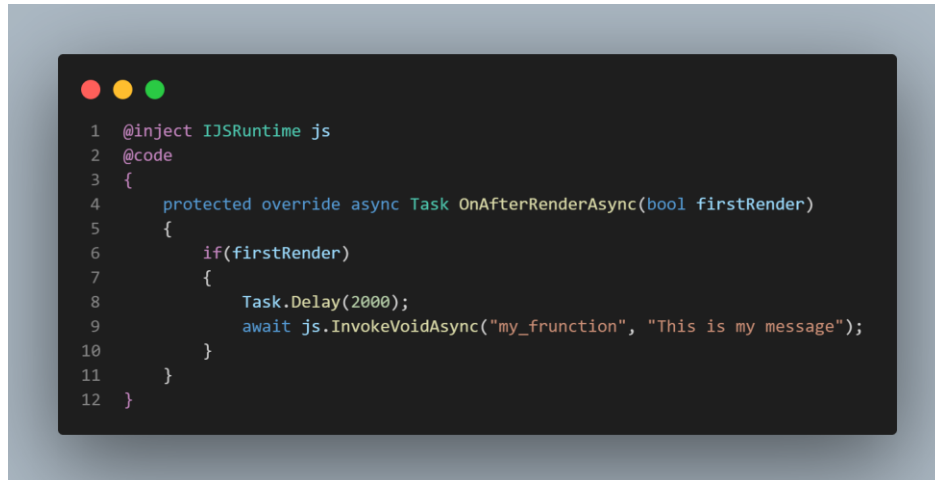


Figure 31 - IJSRuntime DI example

Chart examples:

Due to the large amount of charts that were used within the application, only three would be showcased here in depth simply to demonstrate the process of adding a chart to the application. At its core, the process follows the same exact steps as the one before with the only difference being that the parameter being sent to the function is the “id” of the HTML element that the chart would be sent to, with said element being a “canvas”, an element that specially allows for the addition of Chart.JS charts.

ChartPotential: This snippet creates a doughnut chart with two segments. It represents data with percentages, where one segment is labeled as 4% and the other as 96%. The data array [4, 96] specifies the values for the two segments, and the backgroundColor property defines the colors for each segment.



Figure 32 - ChartPotential JS code snippet

A specialist engineering project

ChartPotentialTwo: Similar to the previous snippet, this code also creates a doughnut chart with two segments representing percentages. However, the colors of the segments are different, defined by the `backgroundColor` property.



Figure 33 -ChartsPotentialTwo JS code snippet

AverageChart: This snippet creates a line chart to display the average grade over time. The labels array specifies the x-axis labels representing different months. The data array contains the corresponding average grades for each month. The line chart has a single dataset labeled as "Average grade," and the `backgroundColor` property defines the color of the line.



Figure 34 - AverageChart JS code snippet

Major technical issues faced during development

The upcoming section will discuss some of the major issues encountered while developing the application, along with their corresponding solutions. This analysis aims to provide insights into the challenges faced during the development process and highlight the steps taken to resolve them. Although many issues were faced throughout the entire development process, only one of which stands out and caused actual problems that resulted in major delays, with the rest of the issues being rather insignificant. For that reason, only the stated issue would be discussed.

Database dictionary mapping:

Problem:

One of the challenges that were encountered during the development of the application was the requirement to use a dictionary to map the subject's name with the student's grade for that subject. Initially, utilizing a dictionary seemed convenient and would have made it easier to navigate and manipulate data within the application. However, difficulty when attempting to map the dictionary directly into the database using Entity Framework Core. EF Core does not provide native support for mapping dictionaries to database columns, posing a roadblock to our desired implementation.

Solution:

To overcome the limitation of mapping a dictionary into the database, a workaround was devised that involved modifying the data structure. Instead of using a dictionary, a new class was introduced called "Subjects" within the entities of the application. The Subject class encapsulated the name of the subject and the corresponding grades. By representing each subject as an individual instance of the Subject class, a similar functionality to that of a dictionary was achieved. From there, a list of instances / object of the aforementioned "Subject" class was initialized within the Student's class.

However, a list of class instances cannot be established within a class and mapped to a database as is, since the contents of said list cannot be encapsulated within a single database entry. As such, a relationship between the Student class and the list of subjects had to be established within the confines of the database, where they would not be within the same table within the database but rather would be connected through foreign keys. To do so, using EF Core a one-to-many relationship was created between the two classes. This relationship was added to the OnModelCreating function in Entity Framework Core, ensuring the proper mapping of the Student class to the list of subjects. This approach manipulates EF core and the MSSQL to achieve the desired functionality of mapping subjects with grades the same way as if it would have initially bene through the use of a dictionary.



```
1 using Dashboard_Project.Shared.Entities;
2 using Microsoft.EntityFrameworkCore;
3
4 namespace PDF_Reader_APIs.Server
5 {
6     public class Database : DbContext
7     {
8         public Database(){}
9         public Database(DbContextOptions<Database> options) : base(options){}
10        protected override void OnModelCreating(ModelBuilder modelBuilder)
11        {
12            modelBuilder.Entity<Students>().HasMany(x => x.Subjects).WithOne(y => y.Student).HasForeignKey(z => z.StudentId);
13            base.OnModelCreating(modelBuilder);
14        }
15        public DbSet<PDF> Students {get; set;}
16
17        //The rest of the entities
18    }
19 }
```

Figure 35 - Adding a one-to-many relationship

Potential improvements

In the subsequent section, the potential improvements that could have been added to the project to further enhance its functionality and user experience would be discussed. Said improvements are suggestions that were not implemented during the project development due to the various constraints imposed upon the team. However, they nonetheless would be beneficial if integrated into the application. By exploring these potential improvements, we hope to identify areas of the project that could be improved in future iterations or implementations. Additionally, this section aims to provide insight into the possibilities of future development and expansion of the application.

Using docker and Kubernetes:

What is Docker and Kubernetes:

Docker: Docker is a tool designed to create, deploy, and run applications in the cloud by running them on what is known as a "container" which utilizes a special kernel that is able to run a software application no matter the environment it is in. To elaborate, a container is an isolated and lightweight environment that includes all the necessary software and dependencies to run a given application on any given machine. Now, Docker is a set of services which is used to simplify this process of creating a containerized environment.

Kubernetes: Kubernetes, on the other hand, is a container manager platform. It allows developers to manage and automate the deployment, scaling, and management of containerized applications (in this case docker container). Kubernetes provides features which allow easier management and control over a set of container. Such features include load balancing, automatic scaling, and self-healing capabilities, which would be too complicated to delve into here. However, overall Kubernetes is the management software for the aforementioned docker containers.

How implementing them would have improved the application:

Implementing Docker and Kubernetes in has various different improvements that could affect the final quality of the application positively. First, Docker provides a consistent environment for development, testing, and production, which can help to reduce the time and effort required to deploy the application, meaning that during the testing process if the application were to be tested in several different environments, it can work effortlessly as the docker container would be the one running the application irrelevant of the environment its in. This could be a potential issue if the application is being initially developed on windows machine but is the tested on a Linux system, which in certain cases during the development process where Al-Hosni had to switch from his Linux system to a windows machine.

It also ensures that the application runs consistently across different systems, reducing the risk of unexpected errors and issues. Kubernetes can help to scale your application automatically based on demand, which can help to improve performance and reduce downtime. Additionally, Kubernetes provides a robust set of features for managing and monitoring various aspects of the website, which can help identify and resolve issues quickly. A major example is Kubernetes's Automatic Scaling feature, where it can automatically reduce and increase the resources such CPU and memory usage of the containers running the application based on the demand from the client base at a given time. Overall, implementing Docker and Kubernetes can help you to improve the reliability, scalability, and performance of your ASP.NET core and Blazor web application.



Figure 36 - Docker and Kubernetes logos

Better security measures:

Adding authentication to the API header:

What is an API header: An API header is a part of the HTTP protocol and is a set of information that is sent alongside the HTTP request or response. When an API request is sent, the server responding will request for a header. Based on the information transmitted in said header, a developer is able to specify what occurs in the functionality of the API, it can be thought of as variables for the API which a developer can use as they deem fit.

Adding authentication to the header: Adding authentication to the API header is a security measure that involves adding a layer of protection to API endpoints by requiring users to provide authentication credentials, such as a username and password, or an arbitrary and complex string of characters known as a Bearer token or API key. With this measure in place, only authorized users with valid credentials can access the API endpoints and make requests.

Implementation process: This can be implemented through specifying which API keys are able to access certain API functionality, with said keys only being given to specified users such as Admins.

Benefit: Implementing authentication for API endpoints is an important security measures for any given application utilizing APIs as it helps prevent unauthorized access to sensitive data or functionality that the API endpoints work with. This can be critical for protecting sensitive data or ensuring that only authorized users are able to perform certain actions. Additionally, authentication can help improve the overall security posture of the application by reducing the risk of malicious attacks, such as brute-force attacks or unauthorized access attempts.

Implementing regular vulnerability testing:

What is vulnerability scanning: Vulnerability scanning is the process of scanning a network for potential vulnerabilities that could be exploited by attackers such weaknesses in the implemented security measures.

Implementation process: This can be implemented by having a separate version of the application on a non-production server in which the application and a copy of the database can be tested and utilized with no potential consequence of breaking the actual application for the end user.

Benefit: By regularly scanning for vulnerabilities, teams are able to evaluate and find threats and take proactive steps to address them before they can be exploited. This can help reduce the overall risk of a successful attack and minimize the potential impact on the application or network.

Logging and monitoring:

Implementing robust logging and monitoring mechanisms into the application is crucial for enhancing its security. As it allows admins to gain valuable insights into the application's activities and detect potential security breaches.

To implement logging, several logging frameworks or libraries could be implemented into the application in order to capture important events, errors, and exceptions occurring within the code. By strategically placing logging statements, a log file that records critical information such as user actions, system events, and security threats can be generated.

Monitoring the application's behavior is equally important. security monitoring tools to track network traffic, access logs, and system performance could be implemented by using additional libraries and frameworks. Through monitoring said factors, any unusual patterns or anomalies that may indicate a security breach or unauthorized activity could be detected.

Real-time alerts and notifications can further enhance security monitoring. Whenever specific security events or triggers are detected, such as multiple failed logins attempts or unusual data access patterns, alerts can be sent to designated personnel or security teams such as the IT department within the school. This allows for immediate investigation and response to potential security incidents.

Testing plan

This section focuses on conducting tests for the developed website, leveraging insights from previous reports while considering the dependency on them. The showcased test plan is the one that was developed and created in the second report. However, it is important to note that not all aspects covered in the previous reports will be included in this section, as certain sections of the project remain incomplete.

Graphing algorithm:

Purpose of test:

Thorough testing was conducted to validate the accuracy and functionality of the graph algorithm implemented on the dashboard website. The primary objective was to ensure that the algorithm produced expected results and performed optimally across diverse scenarios, thereby verifying its functionality. A comprehensive test suite was devised, encompassing various test cases that explored different graph structures, sizes, and complexities. The test suite took into account factors such as nodes, edges, and their relationships within the graph data, which were calculated and visualized by the algorithm to assess its accuracy and ability to handle graph data effectively.

Carried test:

The testing phase encompassed multiple dimensions, including correctness, efficiency, and scalability of the algorithm. Carefully selected inputs were used to represent real-world scenarios and edge cases, thoroughly evaluating the algorithm's capabilities. Rigorous analysis of the test results was carried out to identify any deviations or errors, with a strong focus on ensuring the algorithm's reliability and accuracy. Detected issues were promptly addressed and resolved, further enhancing the algorithm's reliability and accuracy.

Testing loading times:

Purpose of test:

To assess the loading performance of the dashboard website, thorough testing of loading times was conducted. Recognizing the significance of loading time for any website, including dashboard applications, the objective was to measure the time required for the dashboard website to load and render its content effectively.

The testing process involved conducting tests under various conditions, simulating different network speeds and traffic levels to obtain accurate results. The focus was on identifying any bottlenecks or inefficiencies that might impact the loading speed of the website as a whole.

Carried test:

Specialized tools and techniques were employed to measure and analyze the loading times of individual components, such as scripts, stylesheets, and images. Additionally, the overall loading time of the entire website was evaluated. Through this analysis, the team aimed to identify areas for optimization and improvement.

Based on the findings from the tests, a range of optimizations were implemented to enhance the loading speed of the website. These optimizations included code optimization, implementation of caching mechanisms, and

A specialist engineering project

minimizing file sizes. The goal was to deliver a fast and responsive dashboard website, ensuring an excellent user experience. By addressing loading time issues, the team aimed to improve overall website performance and provide users with a seamless and efficient experience.

Test of usability:

Purpose of test:

Usability testing played a crucial role in evaluating the user experience of the dashboard website. The main objective was to assess the ease of navigation, interaction with features, and successful completion of tasks, ultimately focusing on the overall usability of the interface.

Carried test:

To conduct the usability testing, a diverse group of users representing the target audience was selected. These users were assigned specific tasks to perform on the dashboard website while their interactions were carefully observed and recorded.

In addition to task completion, feedback from the users, including their observations and comments, was collected and analyzed. The aim of this usability evaluation was to identify any issues related to the website's usability, such as confusing layouts, unintuitive navigation, or unclear instructions.

The collected data, including user feedback and observations, was thoroughly analyzed to gain insights into areas of improvement. Based on the findings, adjustments were made to the design and interface of the website to enhance its usability. The goal was to create a more user-friendly experience, ensuring that users could navigate the dashboard website effortlessly and accomplish their tasks efficiently. By addressing usability issues, the team aimed to optimize the overall user experience and satisfaction.

Note on security testing:

As part of the testing process, it is important to note that specific security tests, including DDoS (Distributed Denial of Service) and breaching tests, were not performed on the dashboard website. This decision was made by the college, as the application was not deployed to the internet.

The college examined the test environment and determined that there was no immediate need to conduct these tests. Since the dashboard website was not exposed to the internet during the development and testing phases, the risk of DDoS attacks or breaches during this period was eliminated.

However, it is essential to emphasize that security testing is a crucial aspect of any web application. While these tests were not performed during the development and testing stages, it is highly recommended that comprehensive security testing is conducted when the dashboard website is deployed in a live production environment. This will ensure that the website is resilient against potential threats and vulnerabilities, providing a secure user experience for its users.



Final Portfolio



Conclusion

Experience and technical effort:

In conclusion, the development process of our student dashboard project has been a journey filled with learning, collaboration, and continuous improvement. The project aimed to provide students, teachers, executives, and parents with a comprehensive platform to track and monitor academic progress through a dashboard tracking system. By using the skills the team acquired while learning web development in both the front and backend, the team was able to create a great foundation for the application to be further expanded upon.

Through our backend development efforts, the team utilized various techniques and processes to enhance the user experience and functionality of the website. One of the key aspects of our backend development involved implementing cloud migrations for the database and leveraging object storage for efficient image storage. Moreover, the team prioritized creating a well-structured database that allows for easy navigation and future scalability, ensuring that even future teams can seamlessly work on the application. Additionally, the team focused on optimization techniques such as caching and graphing to enhance the responsiveness of the website and provide a smooth user experience.

On the frontend, the team dedicated its efforts to crafting an intuitive and interactive dashboard that can be easily navigated and accessed by students and teachers. As such, the team aimed to create a user-friendly interface that provides a comprehensive view of grades, progress, and areas where assistance may be needed. Through thoughtful design choices and user testing, the team strived to deliver an engaging and efficient user experience.

Project's incompleteness:

However, it is important to acknowledge that the final project is not as complete as what was initially outlined for the project. Although with greater effort the team could have potentially finished the project, we believe that we gave our all and contributed as much as we could have while balancing school and personal life in order to get the project to the state it is currently at.

The primary contributing factor to the incompleteness of the project can be mainly attributed to the time constraint, as it was not only limited by the overall timeframe but also the significant amount of time devoted to learning web development. As a team, we recognized the importance of acquiring the necessary skills and knowledge to effectively implement the project. Consequently, a considerable portion of our allocated time was dedicated to learning and familiarizing ourselves with web development practices and technologies. While this investment in learning was crucial for our long-term growth and competence, it did impact the amount of time available for direct project implementation and completion, not to mention that without the skills we have acquired through learning web development, the team would not have been able to get the project to the state it is currently at. Despite this challenge, we approached the project with enthusiasm and perseverance, successfully incorporating the learned concepts and techniques into the development process.

Despite these missing aspects, the development process has been a valuable learning experience, showcasing the power of collaboration and problem-solving. The completed components, including the intuitive frontend dashboard and the optimized backend functionalities, demonstrate our commitment to delivering a high-quality user experience. With the existing foundation in place, future teams or our own team can easily pick up where we left off and continue expanding upon the project. The student dashboard holds great potential to serve as a valuable tool for students, teachers, executives, and parents, enabling them to monitor and support academic progress effectively.

While we were unable to fully realize the initial plan within the given timeframe, the progress made and the solid foundation laid out provide a strong basis for further development. The project highlights the importance of balancing constraints and prioritizing core functionalities. With future iterations and continued effort, we are confident that the missing aspects, such as admin and executive accounts, deployment, and additional minor features, can be addressed to bring the project to its intended state. The development process has been a testament to our dedication, collaboration, and adaptability, and we look forward to seeing the student dashboard evolve and make a positive impact on the educational community.

Website's pages

This section will showcase the main pages developed for the website and provide a brief overview of their functionality. Each page has been designed with a specific purpose in mind to enhance the user experience and fulfill the objectives of the student dashboard. Through this section, it would be possible to gain an understanding of how the website would be utilized by a student, parent, or teacher, in order to access relevant information and track progress effectively.

Landing and login pages:

Login page:

The page is simply where a student or teacher would be able to login. It is relatively simply and does not include an account creation functionality as accounts would need to be manually added to the database or by the admin account.

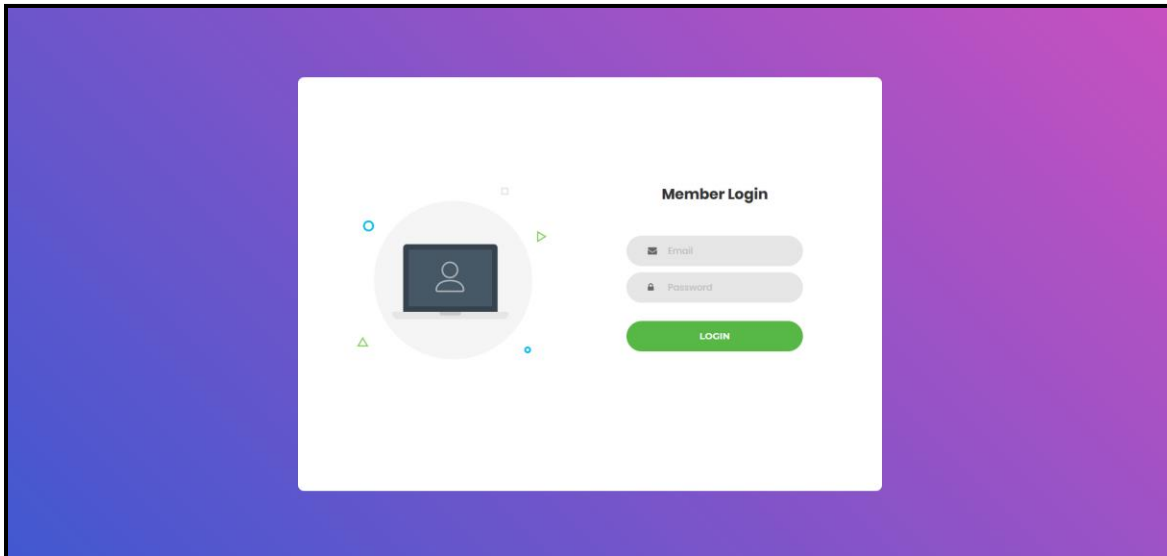


Figure 37 - Login page

Landing page:

The landing page was created with the idea in mind being that anyone could access it and view its contents even people unrelated to the school or parents that would like to enroll their students. For that, general information about the school was provided including what skills students might gain, the varying curriculums, and the departments within the school. Alongside that, the landing page also contains additional contact information and the buttons that would take the student or teacher to their dashboards.

A specialist engineering project

The initial landing page that would greet the user as they log in or open the website.

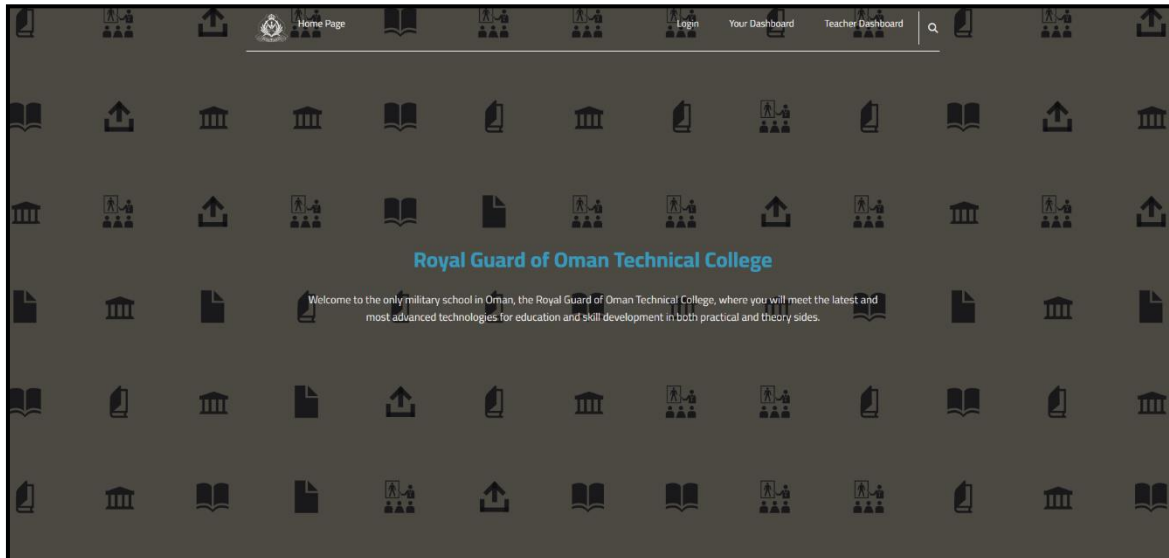


Figure 38 - Initial landing screen

Scroll down

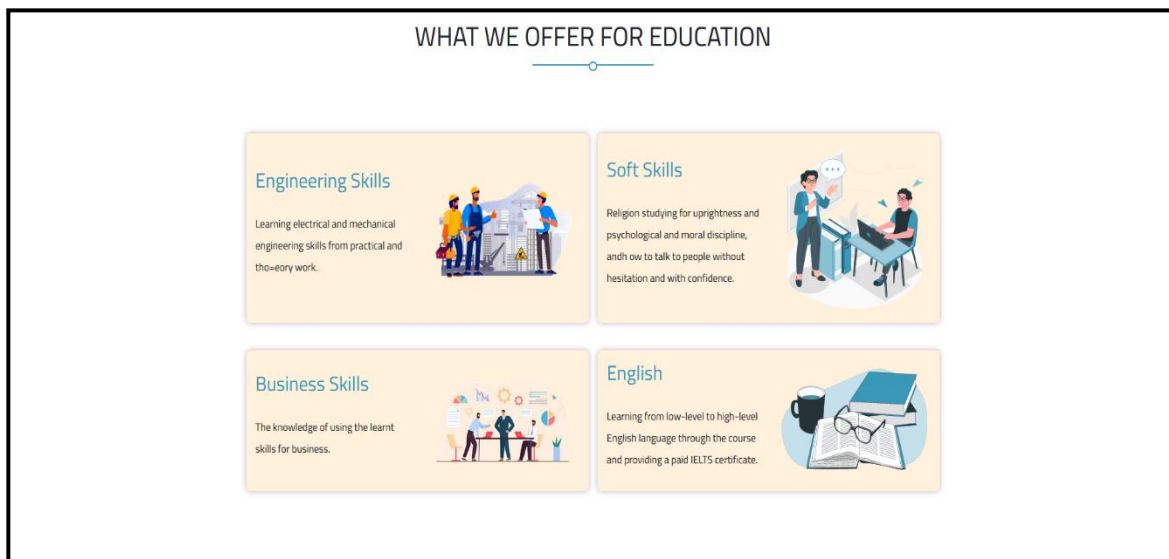


Figure 39 – Skills students will learn

A specialist engineering project

A showcase of the certificates that they student might earn through studying at the school.

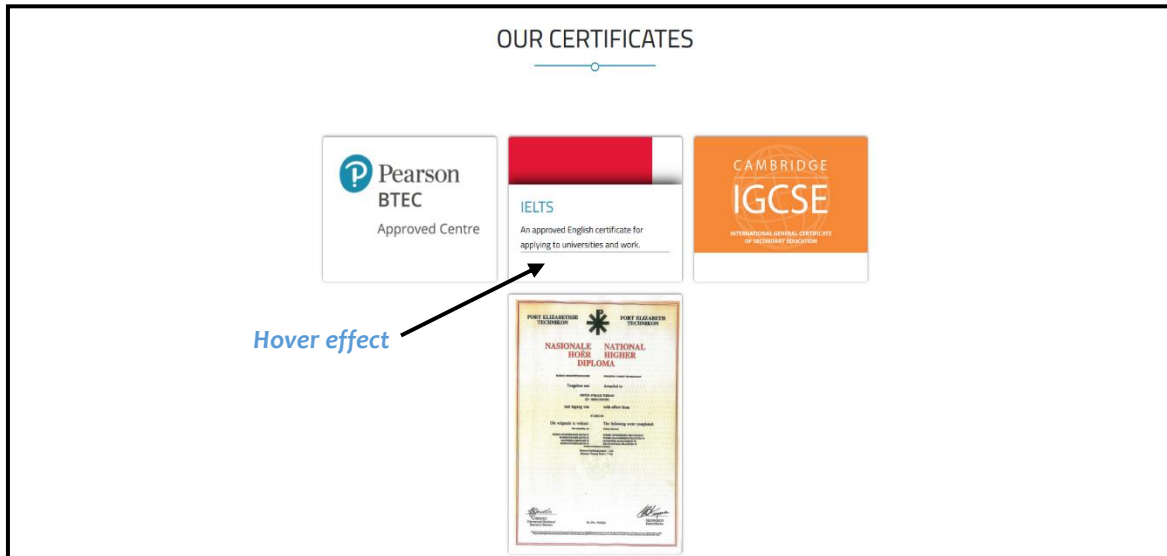


Figure 40 - Certificates showcase

Scroll down

A showcase of the available departments at the school.



Figure 41 - Departments showcase

A specialist engineering project

A footer with additional information for viewing and finding the school.

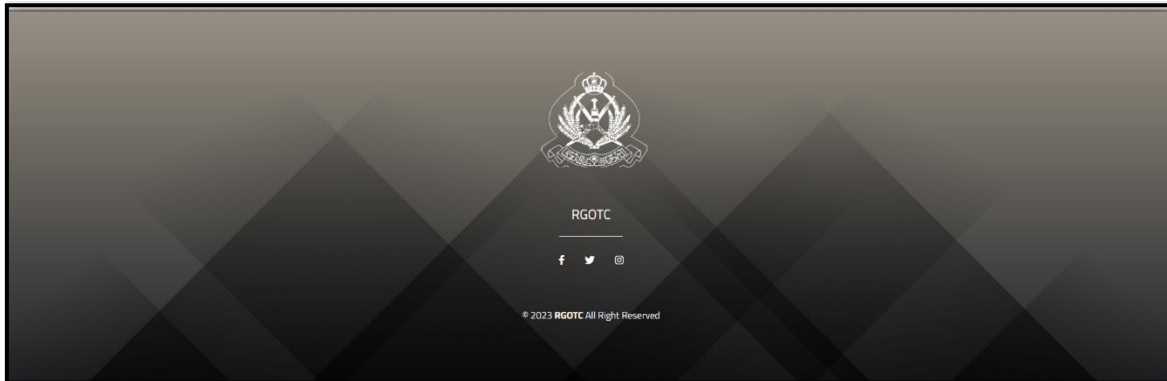


Figure 42 – Footer

Student view:

Main dashboard:

The following is the initial view that the student might see when launching their dashboard. It includes a log of all the events and grades that have been added to their database as well as their final possible grade, their rank among their peers in terms of grades, and the grades they have in the various aspects from homework, assignments, and so on.

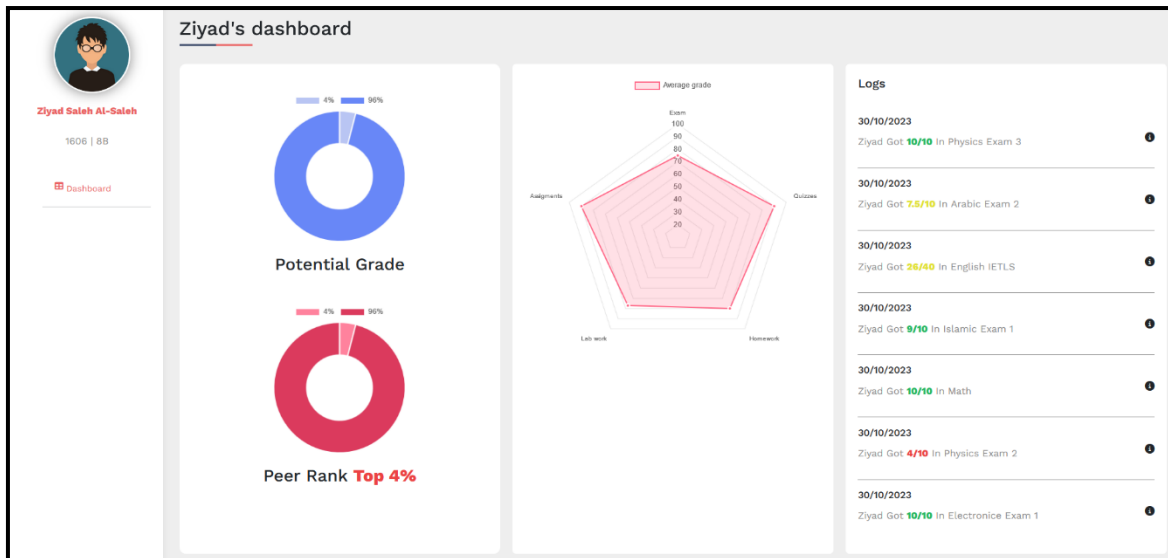


Figure 43 - Initial dashboard view

A specialist engineering project

The following showcases the student's grades out of the possible 100% that they could have achieved throughout the entire year showcased by month

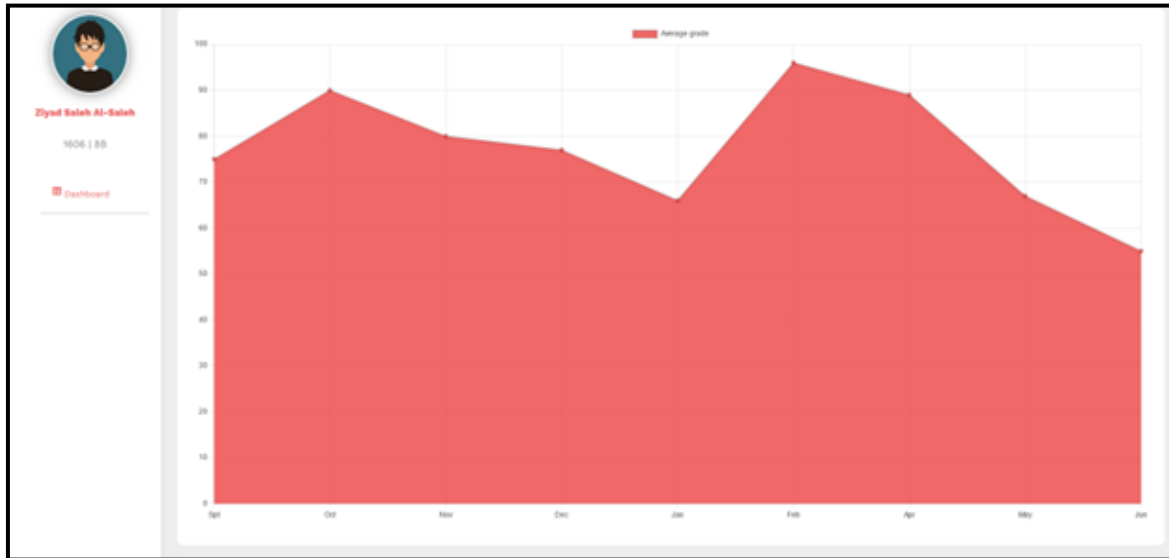


Figure 44 - Grades throughout the year

Scroll down

In the shown screen, the student is able to select which subject they want to view their grades in in more detail. The chart above simply shows the grade out of 100% throughout all subjects



Figure 45 - Selection screen

Subject specific:

After clicking on a subject in the previous screen, this screen will be presented, showing details about the subject that was clicked specifically.

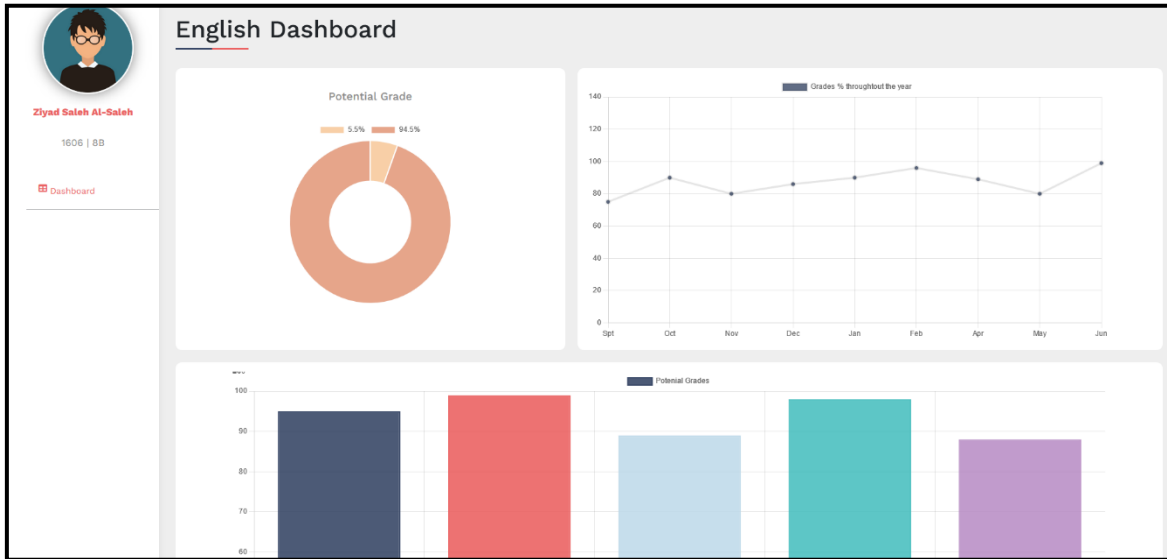


Figure 46 - In depth subject showcase

Scroll down

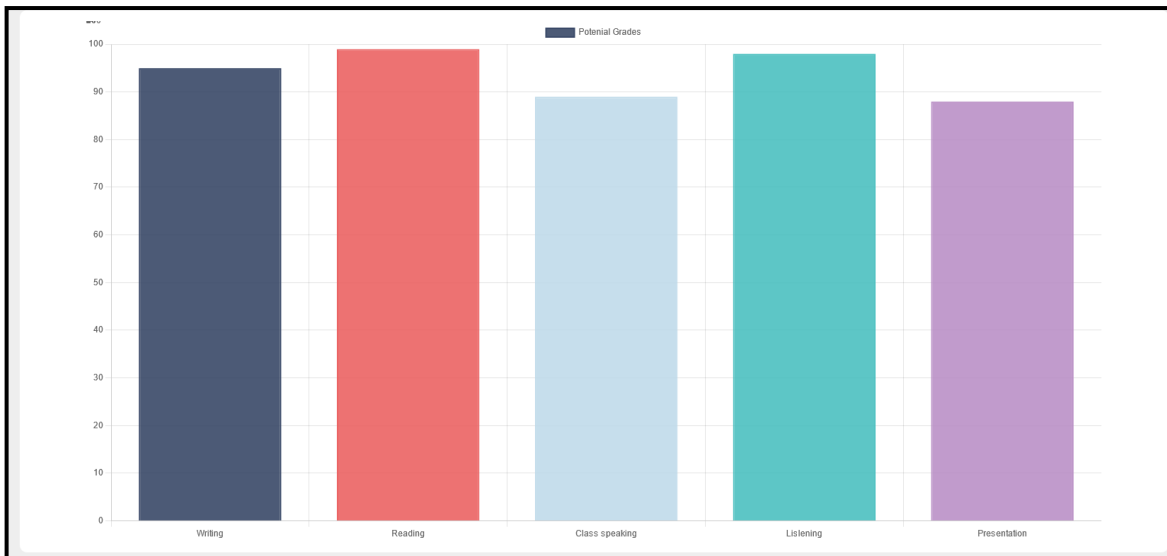


Figure 47 - Full bar chart (Detailed subject view)

Teacher view:

Here, the teacher has the option to choose whether they want to see a list of their classes or a list of their students. This is simply an aesthetic screen meant to act as a landing page for the teacher and then never to be navigated to.

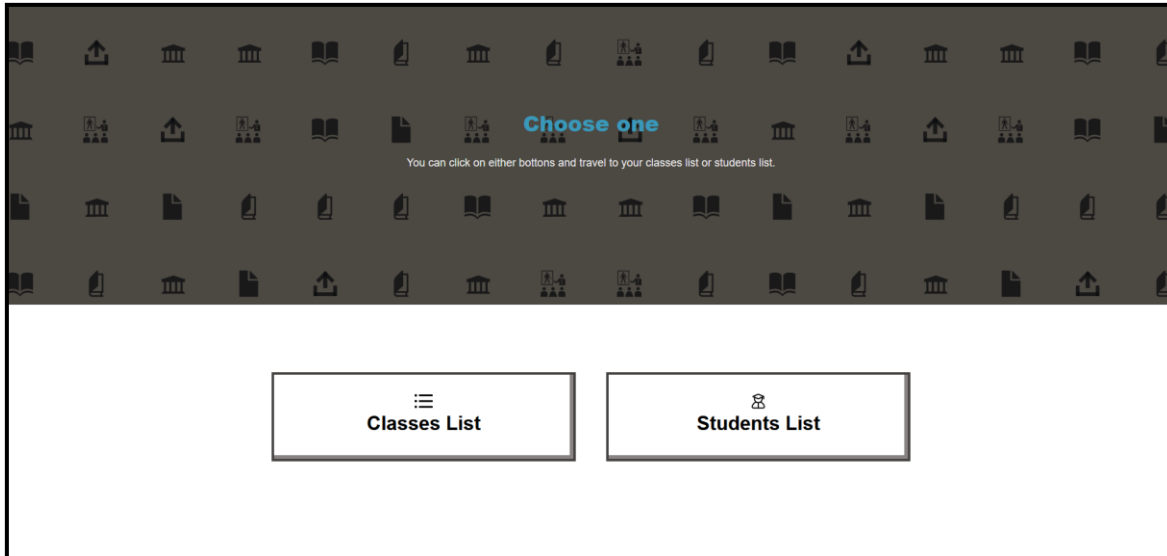


Figure 48 - Teacher landing screen

Clicked on classes list

Classes list:

A list of all the classes that a teacher would have. The teacher would be able to click on a given class's icon which would lead them to a detailed view of said class.



Figure 49 - Classes list

A specialist engineering project

Class specific view:

After clicking on a specific class within the classes list view, the teacher would be presented with this view, showcasing the students and two charts, showing the performance of the students as well as the top students within the class. The teacher can then click on a student to find more information about that student specifically.

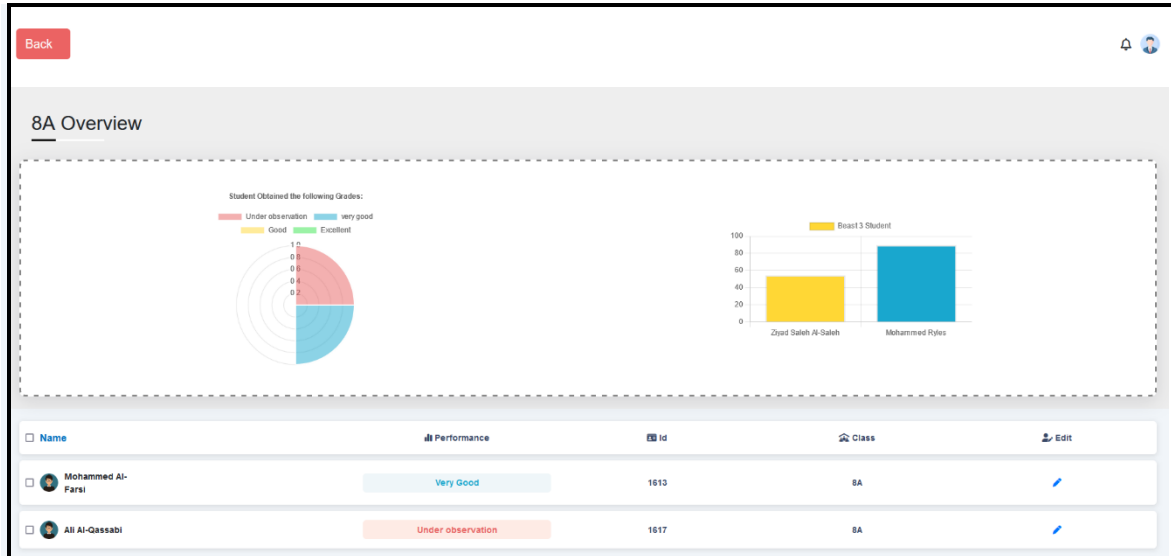


Figure 50 - Class detailed view

Clicked on specific student

Student specific view:

After clicking on a student, the teacher would be able to see that student's subject-based dashboard for the subject that the teacher teaches said student.

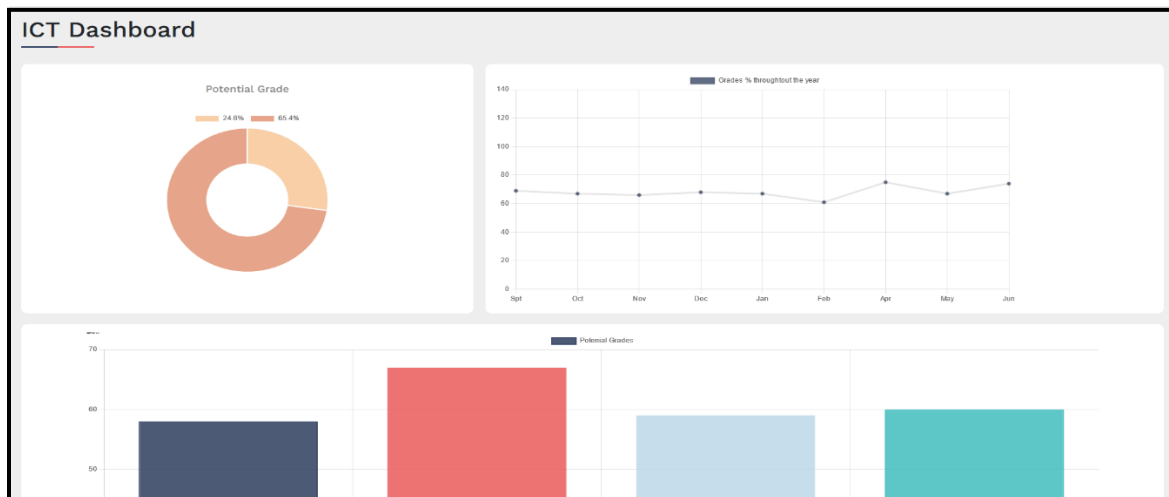
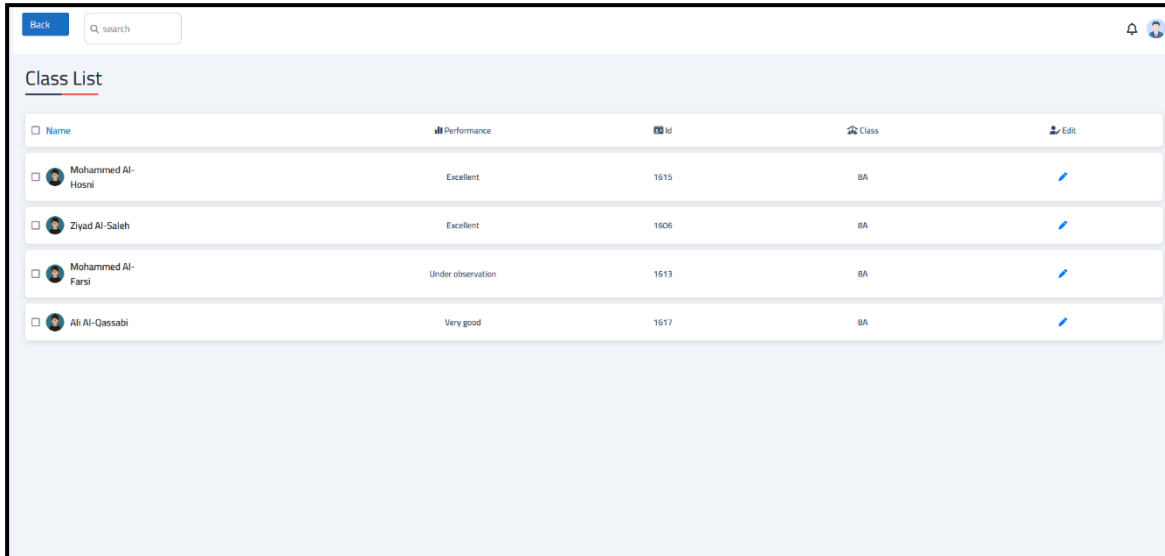


Figure 51 - Student subject dashboard - Teacher view

A specialist engineering project

Students list:

At the teacher's landing screen, if they chose the students list, then they would be presented with this screen, where teacher's can view all their students and in turn click on them to view that student's subject-based view for the subject the teacher teaches them (The same as the one seen in the previous page).



<input type="checkbox"/> Name	Performance	ID	Class	Edit
<input type="checkbox"/> Mohammed Al-Hosni	Excellent	1515	BA	
<input type="checkbox"/> Ziyad Al-Saleh	Excellent	1506	BA	
<input type="checkbox"/> Mohammed Al-Farsi	Under observation	1513	BA	
<input type="checkbox"/> Ali Al-Qassabi	Very good	1517	BA	

Figure 52 - Students list

Meeting minutes

In the following section, we will provide a showcase of the meeting minutes conducted during the development of the website and the progress of the project. These meeting minutes serve as official records documenting the discussions, decisions, and actions taken during each meeting. By presenting these minutes, the team aims to provide insight into the collaborative efforts, milestones achieved, challenges faced, and strategies devised throughout the project's lifecycle.

Date: 15/3]/2023	(Meeting 1)
Agendas:	Discussion
<ul style="list-style-type: none">• Design ideas• Front end kick off	At first, the team met and discussed about how to start making the project, everyone was confused about it so they went and done some research about the initial ideas, which were taken from the internet. They found out some nice designs and combined the ideas to produce a template to work on. Also, they thought about the structure, taking ideas from the previous reports and from the internet.
Action points for the next session:	Conclusions

A specialist engineering project

<ul style="list-style-type: none"> • Reviewing the designs and feedback • Discuss the fundamentals of back end tasks 	<p>It was initially apparent that the team was confused as to how to proceed with the project during the meeting. They conducted research and explored various initial ideas obtained from the internet as a means of addressing this issue. By combining these ideas, they were able to develop a cohesive framework for their work. In addition, they carefully considered the structure of the project, drawing inspiration from both previous reports and online sources. As a result of this collaborative effort, the team was able to establish a clear direction and laid the groundwork for them to proceed with confidence in the development of the website.</p>
--	---

Date: 10/4/2023	(Meeting 2)
Agendas:	Discussion
<ul style="list-style-type: none"> • Front end review • Back end and algorithms 	<p>In the course of the discussion, the team delved into the realm of backend integration for the website project. Several options were explored for seamlessly integrating backend functionality into the website's architecture. In addition, they discussed the API requirements in detail, ensuring that the APIs would support the desired functionality of the website. Furthermore, the team emphasized the importance of smooth data transfer and synchronization between the frontend and backend components of the system. As a result of this comprehensive exploration and discussion, we were able to make an informed decision regarding the backend integration strategy for the project.</p>
Action points for the next session:	Conclusions
<ul style="list-style-type: none"> • Testing plan 	<p>It can be concluded that the team, through thorough exploration and discussion, was able to determine the optimal backend integration strategy for the website project as a result of thorough exploration and discussion. As a result of this informed decision, seamless data transfer between the front-end and back-end components, as well as seamless functionality, will be ensured, ultimately contributing to the overall success of the project.</p>

Date: 26/4/2023	(Meeting 3)
Agendas:	Discussion
<ul style="list-style-type: none"> • Testing strategy • Actual testing 	<p>As part of the discussion on the testing strategy, the team defined test scenarios and cases in order to ensure that the website is thoroughly tested. To validate the functionality and performance of different features, the team identified a variety of scenarios, both typical and edge cases. In addition, the team discussed the importance of automating the testing process to streamline the process and improve efficiency. Due to the incomplete website, the testing will be conducted in accordance with the reports and previous notes. A robust testing strategy was developed by actively discussing and evaluating both manual and automated testing approaches to deliver a high-quality and reliable website.</p>
Action points for the next session:	Conclusions
<ul style="list-style-type: none"> • Finalizing the website • Making a back up copy 	<p>As a result of the team's discussions on the testing strategy, a comprehensive approach was developed to ensure the quality and reliability of the website. As a result of defining test scenarios and cases, the team strives to achieve thorough test coverage as well as validate the functionality and performance of various features. Considering automation testing tools and frameworks demonstrates the team's commitment to streamlining and improving the testing process. While the website is incomplete, the team will make use of reports and previous notes in order to guide the testing process. Through the active discussion and evaluation of manual and automated testing approaches, a robust testing strategy has been developed in order to deliver a high-quality and reliable website.</p>

A specialist engineering project

Date: 15/5/2023	(Meeting 4)
Agendas:	Discussion
<ul style="list-style-type: none">• Reviewing the final website• Back up versions of the website• Modifications and final test	The discussion initiates with the team giving all of their work and tasks and gather them together. The final website has already been tested before with specifics, but the team discussed about testing the finalized website and review it again to make sure that there are no problems with the plan that they are following. If the team noted any issues while testing the final website, they discussed about doing modifications to the website. Also, after doing everything, the team thought of making a back up version of the website in case the original files are corrupted or has been destroyed with any type of harm.
Action points for the next session:	Conclusions
N/A	After reviewing their work and tasks for the final website, the team engaged in a thorough discussion. A final round of testing was conducted to ensure that the website's functionality and identify any potential issues, although it had already undergone specific testing during the development process. A discussion was held regarding the implementation of necessary modifications to address any problems that were discovered during this testing phase. Moreover, the team stressed the importance of creating a backup version of the website in order to minimize the possibility of file corruption or damage. This proactive approach demonstrated the team's commitment to delivering a high-quality and resilient product.

“Data is a precious thing and will last longer than the systems themselves.”

- Tim Berners-Lee

The inventor of the World Wide Web