HMS T&L System

FUNCTIONAL & TECHNICAL SPECIFICATIONS

John Klerck

21/05/2024

Version 1.1

VERSION HISTORY				
VERSION	**APPROVED BY**	**REVISION DATE**	**DESCRIPTION OF CHANGE**	**AUTHOR**
1.1		26/04/2024	Corrected Supabase Spelling mistake Added Google Drive to disallowed data storage Corrected Faculty of Education	John Klerck

Functional & Technical Specifications Document
Authorization Memorandum

I have carefully assessed the Functional & Technical Specifications Document for Share2Teach.

MANAGEMENT CERTIFICATION - Please check the appropriate statement.

_____ The document is accepted.

_____ The document is accepted pending the changes noted.

_____ The document is not accepted.

We fully accept the changes as needed improvements and authorize initiation of work to proceed. Based on our authority and judgment, the continued operation of this system is authorized.

Name: _____ _____

Project Manager Date

Name: _____ _____

Project Sponsor Date

Name: _____ _____

Project Sponsor Date

Name: _____ _____

Project Sponsor Date

Name: _____ _____

Project Sponsor Date

Preface

Purpose

This Technical and Functional Specifications document serves to inform all project stakeholders of the agreed upon project scope, functionality expectations, and technical expectations and responsibility.

Intended Audience and Pertinent Sections

Project Manager - Preface, Introduction

Project Sponsor - Preface, Introduction, Requirements

Development Team - Preface, Introduction, Requirements

Project Scope

This project comprises two user-facing applications supported by a singular backend. The first application is a mobile application built for iOS and/or Android that has the following functionality:

- Browse File-system to select a video and/or
- Record a Video
- Compress video
- Upload video to system
- Browse submissions
- View feedback on submissions

The Second application is a web-app that has the following functionality:

- Create assignments
- View assignments
- View submissions
- Stream/download submission videos
- Providing feedback on video
- Downloading feedback & marks in .csv/.xlsx

The Backend needs to support all the functionality listed above, along with the following functionality for both interfaces:

- Secure Login
- Data stores
- User administration

Introduction

Overview

This project aims to create a multi-platform application system that will assist lecturers in the Faculty of Education Department of Human Movement Sciences at the North-West University to provide better and faster feedback to their students.

This system should enable students to upload a video through an application interface, where it will be stored in a database and tagged. The lecturers should be able to log into a web interface for the system, stream the video, and provide feedback to the student in the form of a text paragraph.

This system is required to enable students who have poor cellular and WiFi connections to be able to record their video assignments, and upload them at a later stage when they have internet access.

Operating Environment

The system should have two user-facing components, one being a website for primarily lecturers and administrators to use, and the other being an installable app for mobile platforms, including iOS and Android devices.

User Roles

Define groups, and describe user characteristics.

Three user groups have been identified. They are presented as follows.

1. Admin
    a) This user role represents the project owner and/or sponsor and the developers maintaining the system. This user has unrestricted access to all components of the system.
2. Lecturer
    a) This user role represents the lecturers within the faculty that will make use of this system.
3. Student
    a) This user role represents the students within the faculty that will make use of this system.

Overview of Functional Requirements

A mobile application should be developed that will run on Android as well as Apple platforms. A student should be able to record and upload a video of them doing some sort of exercise that will notify the corresponding lecturer so they can deliver prompt feedback on the app, having reviewed the footage (like a chat app eg. WhatsApp). Feedback will be generated by the lecturer via comments directly on the platform. The lecturer must be able to assign a mark for the video.

Overview of Data Requirements

This system will require the storage of personal videos that will need to be safeguarded. These videos need to stored in a manner that will allow for video streaming and

downloading. All the commentary provided on a set of videos, grouped by assignment, should be downloadable, along with the mark allocated to the video, and the identity of the user that uploaded the video.

Overview of Technical Requirements

This project will require the upload, storage, compression and streaming of video files. A file storage system will need to be implemented that can effectively index the video files and serve them as requested.

The system will require a backend with an API that will provide access to all the resources available to each user and allow for interfacing with the file storage system. Other data will be stored in a database.

Video files can be quite large; therefore, file size limiting will have to be implemented for restricting file sizes in uploads. Additional video file compression will have to be implemented for efficient file storage. Video files must be saved in a different format for efficient streaming (eg. Conversion from mp4 to H.264 using ffmpeg). For users with weak internet connections, consider using variable bitrate streaming.

Instant feedback and communication are the core of this project, therefore, some instant two-way communication implementations like websockets will be required. Note that a lecturer and a student should have different interfaces that have different permissions. (Eg. A lecturer should be allowed to upload a mark for the video and the student should only be allowed to upload a video).

NOTE: These video files will contain personal footage. Implement sufficient security measures.

Constraints

Compatibility: The application must be a web app, accessible via modern browsers.

Open-source 3rd party libraries: All technologies and libraries used for functional applications (such as architectural, framework and library technologies) with the exception of the hosting costs must be fully free, open-source.

Scalability: Design the software in such a way that it could be deployed in a scalable way using technologies such as Docker Swarm or Kubernetes.

Modularity, Extendibility, Expandability: The constituents of the code architecture must be modular and loosely coupled in support of high loads and future expansion.

Security: The system must adhere to common security standards for secure data storage and transmission.

Budget: The budget allocated will determine the hosting platform for the project.

Time: The current iteration of the project has one semester allocated for completion which will conclude on the 16th of October 2024.

Version Control: The software will use git and GitHub for version control and collaboration.

Assumptions

- The client will be available for regular demonstrations and feedback.
- The requirements outlined in this document dictate the scope of the project that will not undergo significant change.
- There will exist at least one stable branch that will always be ready for the production environment.

Dependencies

3rd party libraries: This project will make use of a vast array of 3rd party software for each part of the system.

Hosting platform: The project will require servers for hosting the application, the data stores and the API. The addition of load balancers may become required.

Stakeholder Input: Regular feedback on the stakeholder requirements will direct the project throughout Its lifecycle.

Guidelines

Coding Standards: Code must follow language-specific conventions.

Commenting and Documentation: Concise, descriptive comments are expected in the code. All functionalities must be documented thoroughly.

Version Control: GitHub will be used to host the project source code. One branch will be named something like ‘stable’ that can be easily identified and will always be production ready.

Unit Testing: Write unit tests for non-trivial source code.

User Documentation

User Manuals: The user will be provided with a detailed guide on how to use the application that covers all the previously discussed features.

Video Demonstration: The stakeholders and users will be provided with a video that demonstrates how to use the system and its features.

Requirements

External Interface Requirements

User Interfaces

Describe product /user interface characteristics, including standards, style guides, constraints, functionality, and sample screens if applicable.

The User Interfaces are split between two components, the mobile application and the web-app interface. The listed interfaces, along with the "flow" described in the "Links To" column, should not be considered exhaustive, and developers should feel free to add additional links to pages and new pages as they see fit. The list should be seen as a guideline representing the optimal path through the system, and negative paths (when errors occur) have not been modeled.

Web-App Interfaces				
No	**User Interface Name**	**Description**	**Data Requirements**	**Links to**
1	Landing Page	This is the first page that is presented to the user when the application is launched.		2,3,4,5
2	Login Page	This page allows a user to log in to the system		1,3,4,5
3	User Administration Page	This page allows a high-level user to manage other users.		1,4,5
4	List Assignments	This page displays a list of assignments per subject.		1,3,4,5
5	Create Assignment	This page allows a user to create a new assignment and link it to a subject.		1,6
6	List Assignment Videos (submissions)	This page allows an administrator and lecturer to view all videos submitted to a particular assignment.		7,
7	Watch Video & Provide Feedback	This page allows a user to stream and/or download a video submission and provide a mark and commentary feedback.		1

Mobile App Interfaces				
No	**User Interface Name**	**Description**	**Data Requirements**	**Links to**
1	Landing Page	This is the first page that is presented to the user when the		2,3

		application is launched.		
2	Login Page	This page allows a user to log in to the system.		1
3	View Assignments	This page will show all the assignments available to the user.		1,4,5
4	Upload Video	This page will allow the user to upload a video as a submission to the assignment.		1
5	View Video Feedback	This page allows the user to view the feedback left on their submission.		1

Communications Interfaces

The system requires the following types of Software interfaces. This list is non-specific, non-exhaustive, and subject to change as the development process proceeds.

Type	Description	Interactions
Database	Relational or non-relational databases used to store documents, user data, transaction logging etc.	Read/Write, backup and restore
File Storage System	The primary file storage for any documents uploaded to the system	Read/Write, backup and restore
API	RESTful API: Primary access to services and features of the system will be accessed through this API gateway.	The frontend application will make requests and receive responses in JSON. OAuth2 is suggested for security
Hosting Operating System	Ubuntu LTS release: The underlying operating system for the application and related services.	The Operating System will be responsible for providing a runtime for the entire application and servers.
Framework	A web application framework for building the frontend user interface	UI rendering and user input for enabling communication with the API gateway
Tool	Docker: A platform for developing, building and shipping and running software in containers.	Builds in CI/CD pipeline, Auto-scaling for load balancing
Authentication	Users will need to be authenticated for access to the system	Users will provide their login credentials in the frontend application for authentication.
Version Control	The git version control platform will be used on GitHub to facilitate version control and developer collaboration.	Branching/Push and pull requests will be made via a shell or on GitHub's

		website.

Functional Requirements

Purpose/Description	Select Video
Inputs	User interacts with button
Processing	File selection browser opened, file type verified, file path captured.
Outputs	Success: Success Message, File path stored, log-file entry.
	Failure: Failure message, log-file entry

Purpose/Description	Record Video
Inputs	User interacts with button
Processing	Camera opened, video recorded, video file saved, file path captured
Outputs	Success: Success Message, File path stored, log-file entry
	Failure: Failure message, log-file entry

Purpose/Description	Compress Video
Inputs	File path to video
Processing	Video File fetched, file size checked against limit, file type checked against goal type. Conversions are done if necessary.
Outputs	Success: Compressed Video File saved, log-file entry.
	Failure: Failure message, log-file entry

Purpose/Description	Upload Video
Inputs	File path to compressed video file
Processing	Open connection to the API, upload file, close connection to the server.
Outputs	Success: Success Message, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Create Assignment
Inputs	User Interacts with button, User provides Assignment information IBNLT: Subject Information, Assignment Name, Due date, Assignment Information
Processing	Data verification; Database entry created.

Outputs	Success: Success Message, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	View Assignment
Inputs	User Interacts with a button
Processing	Assignment information is retrieved from the database
Outputs	Success: Assignment information presented to the user, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	View submissions
Inputs	User interacts with a button
Processing	Submission information is retrieved from the database
Outputs	Success: Submission information presented to user, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Browse your own submissions
Inputs	User interacts with a button
Processing	Submission information is retrieved from the database
Outputs	Success: Submission information is presented to user, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	View feedback on your submissions
Inputs	User interacts with a button
Processing	Feedback information retrieved from the database.
Outputs	Success: Feedback information provided to the user, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Stream video submissions
Inputs	User selects video to stream
Processing	Video stream is initialized, server connection opened, video stream provided, server connection closed
Outputs	Success: Video Stream plays in browser, log-file entry

	Failure: Failure Message, log-file entry

Purpose/Description	Download video submissions
Inputs	User interacts with Button
Processing	Video File is prepared, server connection opened, video file provided, server connection closed.
Outputs	Success: Success message, Video file downloads, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Provide feedback on video
Inputs	User provides text and mark based feedback on the video.
Processing	Feedback and mark data entered into the database
Outputs	Success: Success Message, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Download marks
Inputs	User interacts with a button
Processing	Marks data read from the database, prepared into a file, server connection opened, marks file provided, server connection closed.
Outputs	Success: Success Message,Marks file downloads, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	Secure Login
Inputs	User provides Username and Password
Processing	Password is hashed, Username and hashed password is verified, token is generated on success, token returned.
Outputs	Success: Success Message, User token, log-file entry
	Failure: Failure Message, log-file entry

Purpose/Description	User Administration
Inputs	User Information
Processing	User information is updated.
Outputs	Success: Success Message, log-file entry
	Failure: Failure Message, log-file entry

Non-Functional Requirements

Performance

Scalability: The system should be able to scale horizontally to handle a growing number of users and increased data load without performance degradation. This includes the ability to add more instances of the web server and database as needed.

Throughput: The system should be able to support multiple concurrent users without a significant drop in performance.

Caching: Caching mechanisms should be implemented to reduce server load and improve response times.

Safety

Data Integrity: Ensure all **requests** are processed accurately and data is consistently maintained across the system. Implement database constraints and transactional integrity checks.

Fault Tolerance: The system should be able to recover from hardware or software failures. This includes automatic failover mechanisms and redundancy in critical components.

Error Handling: Implement comprehensive error handling throughout the application. All errors should be logged, and appropriate user-friendly messages should be displayed to the users.

Backups: Regular backups of all critical data should be performed and stored securely. Ensure that a disaster recovery plan is in place and tested periodically.

Security

Authentication and Authorization: All users must be authenticated using a secure authentication mechanism (e.g., OAuth2, JWT). Implement role-based access control to ensure users only have access to the resources they are authorized to use.

Data Encryption: All sensitive data, both at rest and in transit, must be encrypted using industry-standard encryption protocols).

Security Audits: Security audits and penetration testing can be used to identify and mitigate vulnerabilities. Follow best practices and comply with relevant security standards and regulations.

Software Quality

Code Quality: Ensure high code quality by following coding standards and best practices. Conduct regular code reviews.

Testing: Implement a comprehensive testing strategy that includes unit tests, integration tests, system tests, and user acceptance tests. Aim for high test coverage to ensure the reliability of the codebase.

Continuous Integration/Continuous Deployment (CI/CD): Utilize CI/CD pipelines to automate the build, testing, and deployment processes. This helps in detecting and fixing issues early and ensures that changes are delivered quickly and reliably.

Documentation: Maintain up-to-date and comprehensive documentation for the entire system, including code comments, API documentation, user manuals, and operation guides.

Usability: Design the application with a user-centric approach, ensuring that it is intuitive and easy to use. Conduct usability testing to gather feedback and make necessary improvements.

Maintainability: Ensure the system is easy to maintain by modularising the code, following design patterns, and keeping dependencies up to date. Document the architecture and design decisions to aid future maintenance efforts.

MoSCoW Breakdown

The Functionality of the system will be broken down into the following categories to focus your time and energy.

Category	Scope Item	Category Weight
Must Have	Secure Login A Mobile App A Web-App Data Store Video Uploading Video Downloading Providing Feedback on Video Assignment Creation	40%
Should Have	User Administration Video Compression on mobile Video Streaming	35%
Could Have	Download Marks & Feedback per Assignment (.csv/xlsx file) Push Notifications Docker Containerization CI/CD Pipeline Cloud Hosting	25%
Won't Have	Firebase/Supabase Backend Google Drive DataStore	- (<100%)

Suggested Technologies

Popular Tech Stacks

As third-year students delving into software engineering, it's crucial to appreciate the significance of exploring and understanding new technologies. The tech landscape is ever-evolving, and staying updated with the latest tools and frameworks can dramatically influence your ability to create efficient, scalable, and maintainable applications. Here's why it's vital and how you can navigate this exploration effectively.

Pure Frontend (No serving of templates from backend)

- **Web Based**:
    - React
    - Angular
    - Vue
    - Svelte
    - Astro
    - HTMX
    - Solid
    - Remix
- **Mobile**:
    - React native
    - Android Java
    - Android Kotlin
    - Swift IOS **(BEWARE YOU NEED A MAC AND IPHONE TO WORK IN THIS BECAUSE YOU NEED XCODE)**
- **Cross platform:**
    - Ionic
    - Flutter
    - .Net MAUI
    - Kotlin Multi platform

- **Server side or MVC frameworks** are a hybrid approach where the frontend and backend are intertwined into one codebase and there is tight coupling, which has upsides and downsides depending on the situation or context
    - Laravel
    - Django
    - Ruby on Rails
    - Spring MVC Thymeleaf (Spring has a lot of projects with the word spring, so the MVC thymeleaf is important)
    - Golang with Templ

Pure backend

- Expressjs
- Fastify
- Spring boot
- .NET
- Golang Gin
- Golang CHI
- Python Flask

Database choices

When choosing a database, it's crucial to consider how the specific use case affects our decision:

Performance

- **Pros**: Some databases, like Redis or Cassandra, are optimized for high-speed read and write operations, making them suitable for real-time applications.
- **Cons**: High-performance databases might require more resources or be more complex to manage.

Scalability

- **Pros**: NoSQL databases like MongoDB and Cassandra are designed to scale horizontally, easily handling large amounts of data and high traffic.
- **Cons**: Relational databases like PostgreSQL or MySQL might require more complex sharding or replication strategies to scale effectively.

Consistency vs. Availability

- **Consistency**: Relational databases ensure data consistency but might face availability issues under high load.
    - **Pros**: Ensures data accuracy and integrity.

    - **Cons**: Might not be as performant or available in distributed systems.
- **Availability**: NoSQL databases often prioritize availability and partition tolerance over consistency (as per the CAP theorem).
    - **Pros**: Better performance and uptime.
    - **Cons**: Potential data inconsistencies (eventual consistency).

Data Model Flexibility

- **Pros**: Document-based databases like MongoDB offer flexible schemas, allowing for easy updates and changes to the data model.
- **Cons**: This flexibility might lead to inconsistent data structures if not managed properly.

Query Complexity and Support

- **Pros**: SQL databases provide powerful query capabilities with joins, transactions, and complex queries.
- **Cons**: NoSQL databases might require more effort to perform complex queries and may lack advanced querying features.

Transaction Support

- **Pros**: SQL databases support ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring reliable and consistent transactions.
- **Cons**: NoSQL databases may not fully support ACID transactions, focusing on eventual consistency and partition tolerance instead.

Ecosystem and Tooling

- **Pros**: Mature databases like MySQL, PostgreSQL, and MongoDB have robust ecosystems, extensive documentation, and a wide range of tools for management and monitoring.
- **Cons**: Newer or niche databases might lack comprehensive tooling and community support.

Cost

- **Pros**: Open-source databases like MySQL and PostgreSQL can reduce licensing costs, while managed database services (e.g., AWS RDS, Google Cloud Firestore) can lower operational overhead.
- **Cons**: Proprietary databases or large-scale managed services might incur higher costs, particularly as data and traffic grow.

Use Case Suitability

- **Pros**: Some databases are tailored for specific use cases, such as graph databases (e.g., Neo4j) for relationship-focused data or time-series databases (e.g., InfluxDB) for time-stamped data.
- **Cons**: General-purpose databases might not be optimized for specific use cases, leading to suboptimal performance.

Analytics and Data Analysis

- **Pros**: Databases like PostgreSQL and MySQL offer strong analytical capabilities with advanced SQL functions, while specialized databases like Amazon Redshift or Google BigQuery are designed for large-scale analytics.
- **Cons**: General-purpose databases may struggle with performance under heavy analytical workloads compared to specialized solutions.

ETL (Extract, Transform, Load) Processes

- **Pros**: Databases designed for ETL, like Apache Hadoop or Amazon Redshift, can efficiently handle large volumes of data transformation and loading. Tools like Apache NiFi or Talend integrate well with various databases for seamless ETL workflows.
- **Cons**: Using a general-purpose database for ETL can lead to performance bottlenecks and increased complexity in managing the data pipeline.

Understanding these trade-offs is crucial for us to make informed decisions. The right database choice can significantly impact our application's performance, scalability, and maintainability. We need to evaluate our project's specific needs, such as data volume, query complexity, and consistency requirements, to select the most suitable database technology. By considering these factors, we can ensure that we make the best choice for our particular use case. For instance, if our company wants to perform extensive data analytics and ETL processes, we might prioritize databases with strong analytical capabilities and efficient ETL support.

Things to consider

When deciding on a Technology it is key to look at usage first of all, because industry usage means there are big corporates backing a piece of technology, and in the case of frontend this is more crucial because there are many many many ways of taking on problems, and each framework/library comes with its own set of problems and challenges.

Example:

React is simple to pick up and simple to use, but complexity arises quickly when following a pure component-based design, which then causes state management issues where you will need to learn Redux. Conversely a Framework like Next or Angular is harder to just pick up , but has more features baked into to framework and follows a very opinionated approaches.

You know how fast you learn and you know what your appetite is for challenges, pick accordingly and read about different frameworks, as a software engineer you will be put in situations where you will need to make decisions based on how a project is evolving, and your project manager and product owners have no technical knowledge and will look to engineers to make educated decisions, and in smaller companies you will not have the luxury of a solutions architect or competent experienced technical lead. Situations such as caching, database choices, cloud provider choices , logging monitoring, metrics, Test driven development, domain driven development all have upsides and downsides and understanding the tool and the various implementations that are aiming at solving the same problems allows you to make good educated choices and expands your capabilities as an engineer.

When deciding on a technology stack, several factors come into play:

1. **Usage and Industry Support**
    a. Opt for widely-used technologies as they typically have robust community support and backing by large corporations. This ensures better resources, regular updates, and reliability.
2. **Learning Curve**
    a. Assess the complexity of the technology. Some, like React, are straightforward initially but become complex with advanced usage. Others, like Angular, might be harder to pick up but offer extensive built-in features.
3. **Project Requirements and Context**
    a. Consider the specific needs of your project. For instance, server-side rendering (SSR) might be crucial for SEO in web apps, making frameworks like Next.js desirable.
4. **Personal and Team Capabilities**
    a. Reflect on your learning speed and willingness to tackle challenges. Your choice should align with your team's strengths and the project's requirements.
5. **Future Scalability and Maintainability**
    a. Technologies should not only meet current needs but also scale with your project. Consider long-term maintenance and potential for future enhancements.

Resources:

- **Web Based**:
    - React
        - https://www.youtube.com/watch?v=MHn66JJH5zs&list=PLSsAz5wf2lkK_ekd0J__44KG6QoXetZza
    - Angular:
        - https://www.youtube.com/watch?v=3qBXWUpoPHo&t=30601s
    - Vue:
        - https://www.youtube.com/watch?v=pgWZLS75Nmo&t=16s
    - Svelte:
        - https://www.youtube.com/watch?v=wWRhX_Hzyf8

    - Astro:
        - https://www.youtube.com/watch?v=F2pw1C9eKXw&list=PLoqZcxvpWzzeRwF8TEpXHtO7KYY6cNJeF&index=1
    - Solid:
        - https://www.youtube.com/watch?v=uPXn9S31o7Q&list=PL4cUxeGkcC9gU_GvFygZFu0aBysPilkbB
- **Mobile**:
    - React native:
        - https://www.youtube.com/watch?v=ZBCUegTZF7M
    - Android Java
        - https://www.youtube.com/watch?v=cGi9wL8Esw4&list=PL6Q9UqV2Sf1i4eRuXtfWU9nPJ5YldLXbG
    - Android Kotlin
        - https://www.youtube.com/watch?v=BxM2DayeOBE
    - Swift IOS **:**
        - https://www.youtube.com/watch?v=fTGA8cjbf5Y
- **Cross platform:**
    - Ionic
        - https://www.youtube.com/watch?v=K7ghUiXLef8
    - Flutter
        - https://www.youtube.com/watch?v=VPvVD8t02U8&t=21341s
    - .Net MAUI
        - https://www.youtube.com/watch?v=n3tA3Ku65_8
    - Kotlin Multi platform
- **Server side or MVC frameworks** are a hybrid approach where the frontend and backend are intertwined into one codebase and there is tight coupling, which has upsides and downsides depending on the situation or context
    - Laravel (PHP)
        - https://www.youtube.com/watch?v=SqTdHCTWqks

- Django (Python)
    - https://www.youtube.com/watch?v=ZpKl3U-arN8&list=PL4cUxeGkcC9iqfAag3a_BKEX1N43uJutw
- Ruby on Rails
    - https://www.youtube.com/watch?v=Z0Xn1iiiEZE
- Spring MVC Thymeleaf (Spring has a lot of projects with the word spring, so the MVC thymeleaf is important)
    - https://www.youtube.com/watch?v=VqptK6_icjk&list=PL82C6-O4XrHejlASdecIsroNEbZFYo_X1

Pure backend

- Expressjs
    - https://www.youtube.com/watch?v=P6RZfI8KDYc&list=PL_cUvD4qzbkwjmjy-KjbieZ8J9cGwxZpC
- Spring boot
    - https://www.youtube.com/watch?v=Nv2DERaMx-4&list=PLzUMQwCOrQTksiYqoumAQxuhPNa3HqasL
- .NET
    - https://www.youtube.com/watch?v=AhAxLiGC7Pc&t=1674s
- Golang Gin
    - https://www.youtube.com/watch?v=oiPdFkMZ58Q&list=PLDZ_9qD1hkzMdre6oedUdyDTgoJYq-_AY
- Golang CHI
    - https://www.youtube.com/watch?v=JBrF5yviZKE
- Python Flask
    - https://www.youtube.com/watch?v=z3YMz-Gocmw

Database

- Choosing the right database:
    - https://www.youtube.com/watch?v=kkeFE6iRfMM
    - https://www.youtube.com/watch?v=W2Z7fbCLSTw
    - https://www.youtube.com/watch?v=9mdadNspP_M

SOME EXTRAS FOR THOSE INTERESTED

- Designing good API's
  - https://www.youtube.com/watch?v=_gQaygjm_hg
- Software acronyms:
  - https://www.youtube.com/watch?v=cTyZ_hbmbDw
- What is an API?
  - https://www.youtube.com/watch?v=ByGJQzlzxQg
- What is full stack development:
  - https://www.youtube.com/watch?v=7NaeDBTRY1k
- Vertical vs horizontal scaling of an app for higher load:
  - https://www.youtube.com/watch?v=dvRFHG2-uYs
- How git actually works:
  - https://www.youtube.com/watch?v=e9lnsKot_SQ
- What is Docker:
  - https://www.youtube.com/watch?v=Cs2j-Rjqg94
- Docker crash course:
  - https://www.youtube.com/watch?v=pg19Z8LL06w&t=23s
- AWS cloud practitioner Course:
  - https://www.youtube.com/watch?v=NhDYbskXRgc&t=6919s
- What is Cloud?
  - https://www.youtube.com/watch?v=mxT233EdY5c
- What is AWS?
  - https://www.youtube.com/watch?v=a9__D53WsUs
- What is Azure:
  - https://www.youtube.com/watch?v=oPSHs71mTVU
- Azure cloud fundamentals certification course:
  - https://www.youtube.com/watch?v=5abffC-K40c

- Devops vs Site reliability vs Platform engineering
    - https://www.youtube.com/watch?v=an8SrFtJBdM

Free STUFF for students and other free stuff:

- Github Developer pro student pack
    - https://github.com/education/students
- Intellij free student licenses:
    - https://www.jetbrains.com/community/education/#students
- Free AWS skill badges from completing courses:
    - https://aws.amazon.com/education/awseducate/
- Azure student Resources:
    - https://azure.microsoft.com/en-us/resources/students?activetab=pivot:githubtab
- Google cloud skill badges:
    - https://cloud.google.com/learn/training/credentials

Example Stacks, but this is interchangeable

Frontend	Backend	Database	File Storage	Comments
React	Node.js with Express	MongoDB	SeaweedFS	**Pros**: Highly popular, large community support, fast development with JavaScript. **Cons**: Can become complex with state management, callback hell in Node.js.
Angular	Spring Boot (Java)	PostgreSQL	Ceph	**Pros**: Full-featured framework, strong typing with TypeScript, robust backend with Spring Boot. **Cons**: Steeper learning curve, heavy-weight compared to other frameworks.
Vue.js	Laravel (PHP)	MySQL	Nextcloud	**Pros**: Easy to learn, flexible, Laravel offers elegant syntax and tools. **Cons**: Less corporate backing compared to React/Angular, Laravel can be slower than some alternatives.
Flutter	Node.js with Express	MySQL	Garage	**Pros**: Cross-platform mobile development, fast performance, real-time database with Firebase. **Cons**: Still relatively new, less mature than React Native.
React Native	Node.js with Express and SSR using Next.js	MongoDB	SeaweedFS	**Pros**: Cross-platform development, server-side rendering with Next.js, popular and widely used. **Cons**: Complexity with SSR, learning curve for Next.js.
Svelte	Python Flask	MySQL	Nextcloud	**Pros**: Small and fast, easy to learn, lightweight backend with Flask. **Cons**: Less mature ecosystem, less corporate backing.

Ionic with Angular	.NET Core	SQL Server	Ceph	**Pros**: Cross-platform development, enterprise support with .NET, powerful SQL Server. **Cons**: Angular has a steeper learning curve, .NET can be heavy-weight.
HTMX	Django (Python)	PostgreSQL	Nextcloud	**Pros**: Simple frontend integration, beginner-friendly Django, efficient and scalable. **Cons**: Limited features compared to modern JS frameworks, tight coupling.
Astro	Golang with Fiber	PostgreSQL	Garage	**Pros**: New but straightforward, highly performant backend with Go. **Cons**: Less mature ecosystem, limited community support.
Vue.js	Ruby on Rails	MySQL	Ceph	**Pros**: Easy to learn, convention over configuration in Rails, fast prototyping. **Cons**: Performance may not match that of other backend frameworks, less flexible.
React Native	Golang Gin	PostgreSQL	SeaweedFS	**Pros**: Cross-platform mobile apps, highly performant Go backend. **Cons**: Learning curve with Go, fewer libraries and tools compared to Node.js.
Angular	ASP.NET Core	SQL Server	Garage	**Pros**: Enterprise-level support, robust and scalable, TypeScript. **Cons**: Steeper learning curve, heavy-weight framework.
Svelte	Rust with Rocket	PostgreSQL	Nextcloud	**Pros**: High performance, memory safety with Rust, easy frontend with Svelte. **Cons**: Steeper learning curve for Rust, smaller community.

Flutter	Golang with Chi	PostgreSQL	Garage	**Pros**: Cross-platform mobile, simple and performant backend. **Cons**: Flutter's learning curve, less mature ecosystem.
Ionic	PHP with Symfony	MySQL	Ceph	**Pros**: Cross-platform development, flexible PHP backend with Symfony. **Cons**: PHP may be less performant, steeper learning curve with Symfony.
Vue.js	Node.js with Fastify	MongoDB	SeaweedFS	**Pros**: Fast and lightweight, easy integration with Vue. **Cons**: Smaller community compared to Express, learning curve with Fastify.
React	Kotlin with Ktor	PostgreSQL	Ceph	**Pros**: Modern JVM language, highly performant, strong typing. **Cons**: Newer ecosystem, fewer resources and libraries.

Other Technologies to consider:

- **Frontend**: Svelte.
- **Backend**: Flask (Python), Springboot (Java, Kotlin, Groovy, you can choose).
- **File Storage**:, Ceph, Garage: https://garagehq.deuxfleurs.fr/
- **Full Stack**: Ruby on Rails, ASP .NET MVC.
- **Android**: Kotlin
- **Apple**: Swift

Please consider using some of these. Otherwise, at least use some combination of these technologies.

Understanding the trade-offs when choosing a tech stack is crucial for several reasons:

1. **Informed Decision Making**

    a. Engineers often face decisions that impact the entire project lifecycle. Being aware of the pros and cons helps in making choices that align with project goals and constraints.

2. **Adaptability**
    a. The tech landscape changes rapidly. Familiarity with multiple technologies enhances your ability to adapt and integrate new solutions as needed.

3. **Optimization**
    a. Different stacks offer varying levels of performance, scalability, and maintainability. Knowing these nuances allows you to optimize your application for better user experience and resource management.

4. **Leadership and Guidance**
    a. As you progress in your career, you'll be expected to guide less experienced team members. A broad understanding of tech stacks empowers you to mentor effectively and lead projects confidently.

In summary, exploring new technologies enriches your skill set and equips you to tackle diverse challenges. Embrace this exploration with curiosity and critical thinking to become a versatile and proficient software engineer.

Timeline Outline

Date	Work to be evaluated
27 October	Swagger/OpenAPI BackEnd Database File Storage Testable Endpoints + Best Practices + Use of Project Management Tools
16 October	UI Design (Wireframes) UI Creation UI & API Interacts User Analytics + Best Practices + Use of Project Management Tools

Appendices

Appendix Number	Document Name	Description	Location
A	Glossary of Terms	A Glossary containing all unique terms and acronyms used within this document that pertains to this project.	
B	Issue List	A list of outstanding issues within this document, to be rectified before the next revision.	
C	Analysis Models	A document containing various diagrams generated during the systems analysis and design phases	

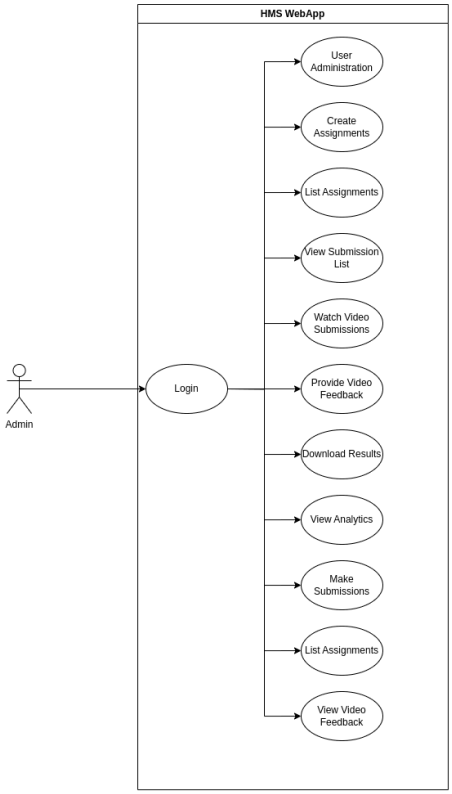
Appendix A: Glossary of Terms

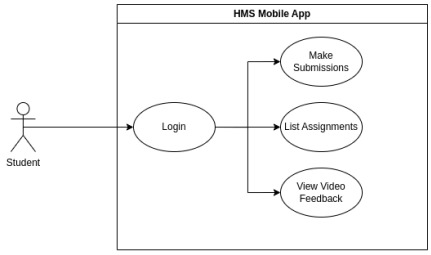
Term	Definition
HID	Human Interface Device. A method by which a human interacts with an electronic information system either by inputting data or receiving output.
Metadata	Data that provided information about one or more aspects of data. Used to summarise basic information about data to make tracking and manipulate data easier.
IBNLT	Including But Not Limited To

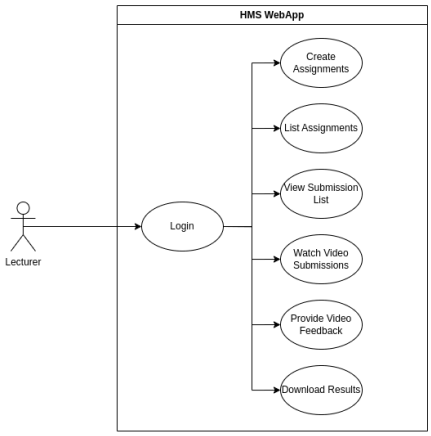
Appendix B: Issue List

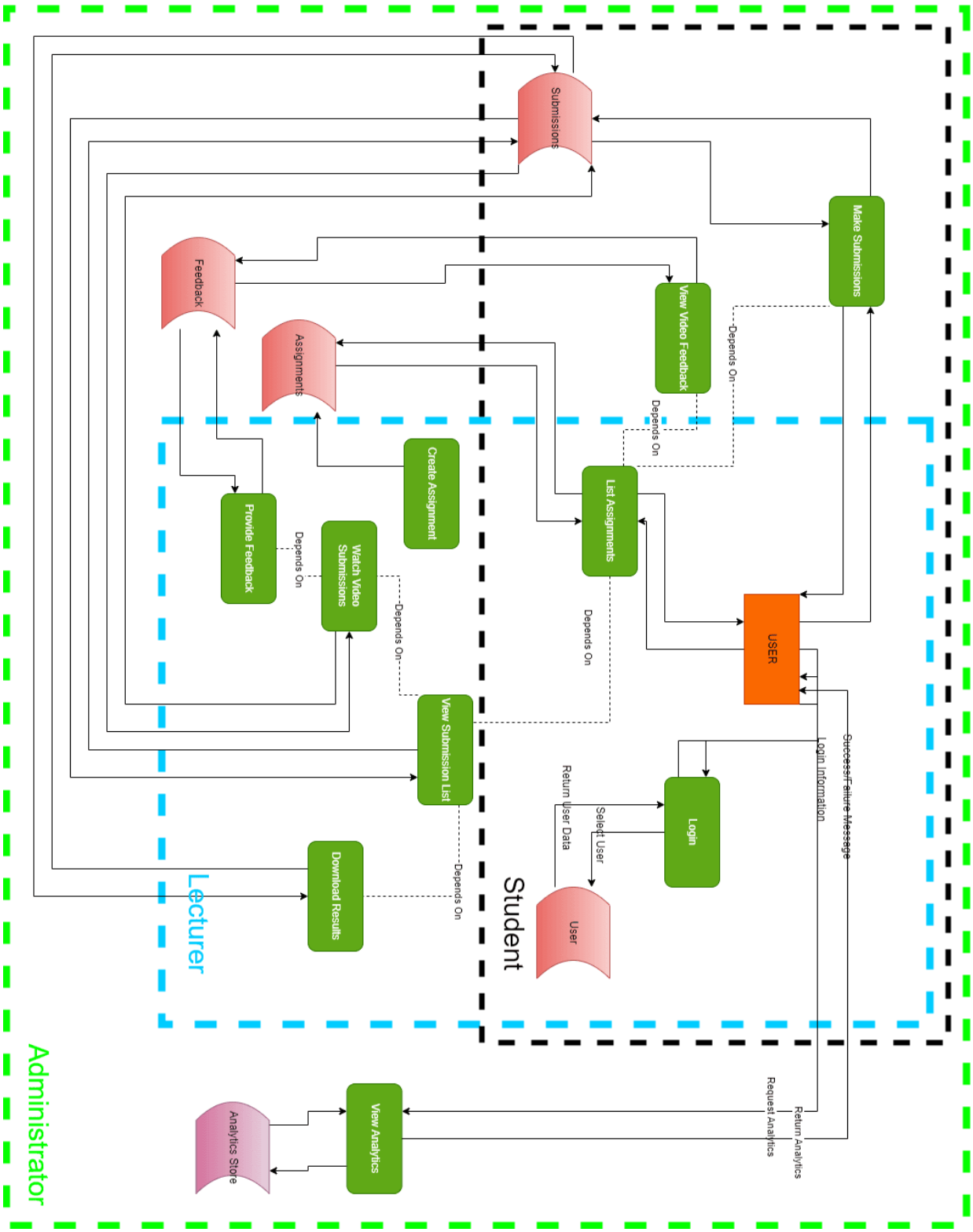
Date	Issue	Description	Version Resolved
26/07/2024	Faculty Incorrect	Faculty of Education Department of Human Movement Sciences was incorrectly identified as Faculty of Human Movement Sciences on page 7.	1.1

Appendix C: Analysis Models








Submissions
Make Submissions
Feedback
View Video Feedback
Assignments
Depends On
Depends On
Create Assignment
List Assignments
Provide Feedback
Watch Video Submissions
Depends On
Depends On
Depends On
USER
View Submission List
Login
Return User Data
Select User
Login Information
Success/Failure Message
User
Download Results
Depends On
Lecturer
Student
Administrator
Analytics Store
View Analytics
Request Analytics
Return Analytics