Share2Teach

Open Educational Resources

FUNCTIONAL & TECHNICAL SPECIFICATIONS

John Klerck

04 July 2024

Version 0.5

VERSION HISTORY				
VERSION	**APPROVED BY**	**REVISION DATE**	**DESCRIPTION OF CHANGE**	**AUTHOR**
0.1	Johan Venter	21/05/2024	Functional Draft	John Klerck
0.2	John Klerck	29/05/2024	Technical Draft	Johan Venter
0.3		11/06/2024		
0.4		02/07/2024		

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 the initiation of work to proceed. Based on our authority and judgement, the continued operation of this system is authorized.

Name: ______________________________	______________________________
Project Manager	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, technical expectations, and responsibilities.

Intended Audience and Pertinent Sections

Project Sponsor - Preface, Introduction

Project Manager - Preface, Introduction, Requirements

Development Team - Preface, Introduction, Requirements

Project Scope

This project aims to extend upon the Share2Teach platform by implementing the following features:

1. Account Creation and Secure Sign-in
2. File Uploading & Storage
3. File Moderation (Gate keeping, reviewing, approving/denying of documents)
4. File Reporting
5. Pre-pending Watermark/License to files
6. Allow tagging of Documents when uploading
7. Allow Searching of Documents
8. Implement Analytics to monitor user engagement and behaviour
9. Facilitate document ratings by Users.
10. Create an FAQ page

Introduction

Overview

Share2Teach is introduced as a vibrant open educational resource (OER) project crafted to nurture a global community of learners and educators. At its core lies the principle that knowledge should be accessible, collaborative, and freely available. Share2Teach is a testament to the power of collective endeavour, co-crafted by students under the guidance of their facilitators.

The project was initiated by Dr. Chantelle Bosch, a dedicated lecturer and sub-area leader for Blended Learning to Enhance Self-Directed Learning within the Research Unit Self-Directed Learning at the North-West University (NWU). Alongside her, Prof. Dorothy Laubscher, the chair-holder of the UNESCO Chair on Multi-modal Learning and OER, has played a pivotal role in shaping the vision and trajectory of Share2Teach.

Together, a platform has been cultivated where diverse educational resources are brought to life, crafted by students for students. From comprehensive semester planning documents to topic-specific insights, Share2Teach offers a wide array of materials tailored to enhance self-directed learning through cooperative learning and project-based teaching strategies.

Share2Teach serves as a beacon for educational innovation, extending an invitation to educators and learners worldwide to contribute, explore, and evolve within this open, inclusive community. Joining this journey means participating in the endeavour to transform learning into a shared adventure, dismantling barriers and erecting bridges toward a more knowledgeable and interconnected world.

Operating Environment

The system will be web-based, and only allow access through a web browser using the internet. This system will make use of technologies supported by most of the latest desktop and desktop-like browsers and should allow for a seamless user experience. Mobile compatibility might be supported, but the system will not be optimised for mobile usage.

User Roles

Four 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. The only unique component that this user has access to is the analytics component. Access to this user role is tightly controlled.
2. Moderator
    a) This user role represents a subset of subject experts as selected by the project owner and/or sponsor. This user has access to the following features:
        i. Document Searching
        ii. Document Viewing
        iii. Document Contribution

iv. Document Rating

v. Using the FAQ

vi. Moderating Documents

3. Educator

a) This user role represents any user who registers for an educator's account. This user has access to the following functionality:

i. Document Searching

ii. Document Viewing

iii. Document Contribution

iv. Document Rating

v. Using the FAQ

4. Open Access User

a) This user role represents any user that accesses the site and does not sign in. This user has access to the following functionality:

i. Document Searching

ii. Document Viewing

iii. Document Rating

iv. Using the FAQ

Overview of Functional Requirements

The following Functional Requirements have been stipulated by the client:

·Account Creation and Secure Sign-in
·File Uploading & Storage
·File Moderation (Gate keeping, reviewing, approving/denying of documents)
·File Reporting
·Pre-pending Watermark/License to files
·Tagging of Documents when uploading (Metadata)
·Searching of Documents
·Analytics to monitor user engagement and behaviour
·Document Ratings
·FAQ page

Overview of Data Requirements

The following Data Requirements have been stipulated by the client:

·File Storage
·File Metadata
·File Ownership
·Moderation History
·User Data - Account Creation
·User Analytics

Overview of Product and Technical Requirements

This section is meant to outline the technical requirements for the development of a web application designed to provide free access to educational resources for students from various backgrounds. Roles including educators, moderators, open access users and admins will manage and interact with the content, based on the assigned permissions. It details the technologies, architecture, and implementation strategies to be used by the development team.

Application

This project will deliver an application that users can interact with in the form of a web app. This application will replace an existing Google pages application that has a familiar look and feel. This application will form the main way in which users will interface with the system.

Data Storage

The system needs to store different types of data:

- Document Storage: Uploaded documents will be kept in a file storage system.
- Document Metadata: Information like subject, grade, and keywords, along with document ratings and storage locations, will be stored in an SQL database.
- User Data: User login details, roles, and activity logs for analytics and audits will be securely stored.
- FAQ: Frequently asked questions and answers will be managed in a database.

Security

1. **Verification and Validation**: This system will make use of HTTP-only cookies, JWTs and encrypted passwords for making sure that users are only able to access and modify the content that they are authorized to, and to ensure that user accounts are secured.

Backend

1. **API**: This system will provide a server listening for requests from the application. Request behaviour will be moderated using an API that will abstract away the complex interactions from the application with the data. It should be documented using a framework like Swagger/OpenAPI.

Architectural Requirements

1. **Hosting**: This application will have to be hosted on a web hosting platform that could either be an in-house bare metal server or a 3rd party service such as AWS, Microsoft Azure, Google cloud.

Constraints

Compatibility: The application must be a web app, accessible via modern browsers (Chromium/V8 support required, WebKit/Firefox optional).

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.

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.

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. Each member needs to prove a similar level of contribution to the project.

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.
- The existing web platform will be extended, but not replaced.
- 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 gateway. The addition of load balancers may become required.

Stakeholder Input: Regular feedback on 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.

FAQ: The main site will contain an FAQ section that will provide users with access to frequently asked questions and their suggested answers. FAQ's should be stored in a database and be retrieved for the site.

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

The following table contains a preliminary list of the identified User Interfaces required in the system. This list is non-exhaustive, and subject to change. The "flow" through the system as described in the "Links to" column is considered to be the optimal flow through the system as currently described. As Developers develop the system, more interfaces may be identified, which could lead to changes in the flow through interfaces. It is also important to note that the "flow" is only 'positive' and does not account for when a user may encounter errors or issues with the system ('negative' flow). It is up to the developers to account for this 'negative' flow.

No	**User Interface Name**	**Description**	**Links to**
1	Landing	The main page the user sees. Presents the user with hyperlinks to the majority of the screens listed below.	All
2	Subject View	This page is subject specific. It allows further searching/viewing of documents within a subject.	3,4,13,
3	Search (Results)	This page allows for searching of documents based on tags created when the documents are uploaded. It provides optional filters along with the search results.	4,13
4	Contribute	This page allows a user to contribute documents and to tag them while doing so.	13,
5	Other useful OER's	This page directs the user to other OER resource pages.	All
6	Contributors	This page displays all the contributors to the project. This page needs to be updated to be dynamic.	All
7	About Us	This page provides information about the project, the project website, and the project founders.	All
8	Self-Directed Learning	This page provides links to Self-Directed Learning resources available to users.	All
9	Moderate	This page will present a user with all the files uploaded by users, that need to be moderated.	
10	Account Creation	This page allows users to create accounts.	
11	Password Reset	This page allows users to reset their passwords should they forget them.	
12	Analytics	This page allows administrators to view the	

		analytics information gathered on the website.	
13	FAQ	This page provides users with answers to some frequently asked questions.	All
14	User Management	This page allows an administrator to change the access levels of users as well as view all the users on the system	

Hardware Interfaces

The system requires the following Hardware interfaces.

Type	Description	Interactions
Keyboard-like HID	A HID that is used to input keystroke data into the website's text fields and interact with buttons.	
Mouse-like HID	A HID that is used to select buttons, fields, and files on the website.	Selecting (Focusing) on buttons and fields.

Software and 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, back up and restore.
File Storage System	The primary file storage for any documents uploaded to the system.	Read/Write, back up and restore.
API	RESTful API: Primary access to services and features of the system will be accessed through this API gateway.	The front-end 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.
	The git version control platform will be used on GitHub to	Branching/Push and pull

Version Control	facilitate version control and developer collaboration.	requests will be made via a shell or on GitHub's website.

Functional Requirements

This list contains the functional requirements for the system to be developed during the project's execution. This list is non-exhaustive and should be expanded upon as the developers identify a need.

Purpose/Description	Account Creation
Inputs	FName, LName, Email, Password, Affiliation (opt), Credentials (Opt)
Processing	Required Field check, Email-type check, password match check
Outputs	Failure: Required Fields empty, invalid email type, passwords don't match.
	Success: Account creation successful

Purpose/Description	Sign In
Inputs	Email, Password
Processing	Required Field Check, email type check, user exist check, password match check
Outputs	Failure: Email or password does not match; required fields empty
	Success: sign in successful.

Purpose/Description	File Uploading & Tagging
Inputs	User File, File type, file name, subject, grade, date created,
Processing	Required field check, file size check, file type check,
Outputs	Failure: File too large, incorrect file type, required fields empty, file upload failed,
	Success: file upload successful.

Purpose/Description	File Storage
Inputs	none
Processing	File storage
Outputs	Failure: Server-side logging
	Success: Server-side logging

Purpose/Description	File Moderation
Inputs	File approval/disapproval, comments
Processing	updating file approval status
Outputs	Failure: Could not complete action
	Success: Action Complete

Purpose/Description	File Rating
Inputs	0-5 star rating for document
Processing	Rating entry made in database
Outputs	Rating visible to user

Purpose/Description	File Reporting
Inputs	Report button selected, reason selected
Processing	Report entry made in database
Outputs	“Report submitted” message

Purpose/Description	Watermark/License adding
Inputs	File uploaded
Processing	Watermark/License is Prepended
Outputs	File with watermark/license is saved in file storage

Purpose/Description	Document Search
Inputs	Keywords entered by user
Processing	Database tags searched for keywords.
Outputs	Files presented to user.

Purpose/Description	User Analytics
Inputs	User actions on system
Processing	Data relating to actions written to database.
Outputs	none

Purpose/Description	FAQ
Inputs	none

Processing	FAQ's Fetched from Database
Outputs	List of FAQ’s with answers provided

Purpose/Description	Password Reset
Inputs	Password Reset request, email address, verification cookie/session/token, new password
Processing	Email verification, token verification and password match verification
Outputs	Failure: Failure message.
	Success: Email Verification Message to generate token, success Message

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 modularizing the code, following design patterns, and keeping dependencies up to date. Document the architecture and design decisions to aid future maintenance efforts.

Product and Technical Requirements

This section is meant to outline the technical requirements for the development of a web application designed to provide free access to educational resources for students from various backgrounds. Roles, including educators, moderators, open access users, and admins, will manage and interact with the content, based on the assigned permissions. It details the technologies, architecture, and implementation strategies to be used by the development team.

The system will provide a web application that the primary users will interact with for downloading, uploading, and approving documents. These documents can be tagged with descriptions and other relevant information that will be used to index and identify the documents in a central data store.

The system will implement a user access and upload policy that will allow only authenticated users to approve document upload requests. In this way, document quality can be ensured. This will require users to sign up to the system and apply for moderator privileges. This user data will be stored in the data store, and any sensitive user information will be encrypted.

For all data processing and preparation, a backend server will be hosted that will be responsible for all data processing and aggregation. This backend server will provide access to data in the form of an API gateway.

The entire system will need to be hosted on a platform that will give all the users access to the system. Some options are available for in-house hosting as well as cloud providers.

Application

This project will deliver an application that users can interact with in the form of a web app. This application will replace the existing Google pages application but retain the familiar look and feel. This application will form the main way in which users will interface with the system.

Data Storage

The system will require persistent storage for the following:

1. **Document Storage:** All uploaded documents will need to be stored in a file storage system.
2. **Document Metadata:** When a user is in the process of uploading a document, they will add some metadata about the document such as the subject group, grade, key words etc. This data, the document ratings and a reference to the location of the actual document in the file storage system will need an entry in a database. The database paradigm suggested for this is SQL, since we are dealing with highly structured data, where data aggregation, sorting and grouping will be very important when users search and filter for documents.
3. **User Data:** All user-related content, such as their login information and the associated roles need to be stored for user verification and identification. For analytics and auditing purposes, all user actions will be recorded in a database with timestamps and actions.
4. **FAQ:** Frequently asked questions and answers to these will be stored in a database.

Security

2. **Verification and Validation**: This system will make use of HTTP-only cookies, JWTs and encrypted passwords for making sure that users are only able to access and modify the content that they are authorized to, and to ensure that user accounts are

secured.

Backend

2. **API Gateway**: This system will provide a server listening for requests from the application. Request behaviour will be moderated using an API gateway that will abstract away the complex interactions from the application with the data.

Architectural Requirements

2. **Hosting**: This application will have to be hosted on a web hosting platform that could either be an in-house bare metal server (your laptop) or a 3rd party service such as AWS, Microsoft Azure, Google cloud.

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) except the hosting costs must be fully free, open-source.

Scalability: The constituents of the 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.
- The existing web platform will be extended, but not replaced.
- 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 gateway. 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.

FAQ: The main site will contain an FAQ section that will provide users with access to frequently asked questions and their suggested answers. FAQ's should be stored in a database and be retrieved for the site.

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

MoSCoW Breakdown

Categories	**Item**	**Category Weight**
Must Have	User Verification File Uploading File Storage Document Reporting File Metadata capturing Document Searching Document ratings FAQ Page	40%
Should Have	File Moderation Automatic Metadata generation PDF Conversion Pre pending Licence	35%
Could Have	Docker/Podman Deployment Google Analytics integration Full WebKit & Firefox Support	25%
Won't Have	Firebase/Superbase Backend	- <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
        - https://www.youtube.com/watch?v=SqTdHCTWqks

- Django
    - 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 pertain 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	

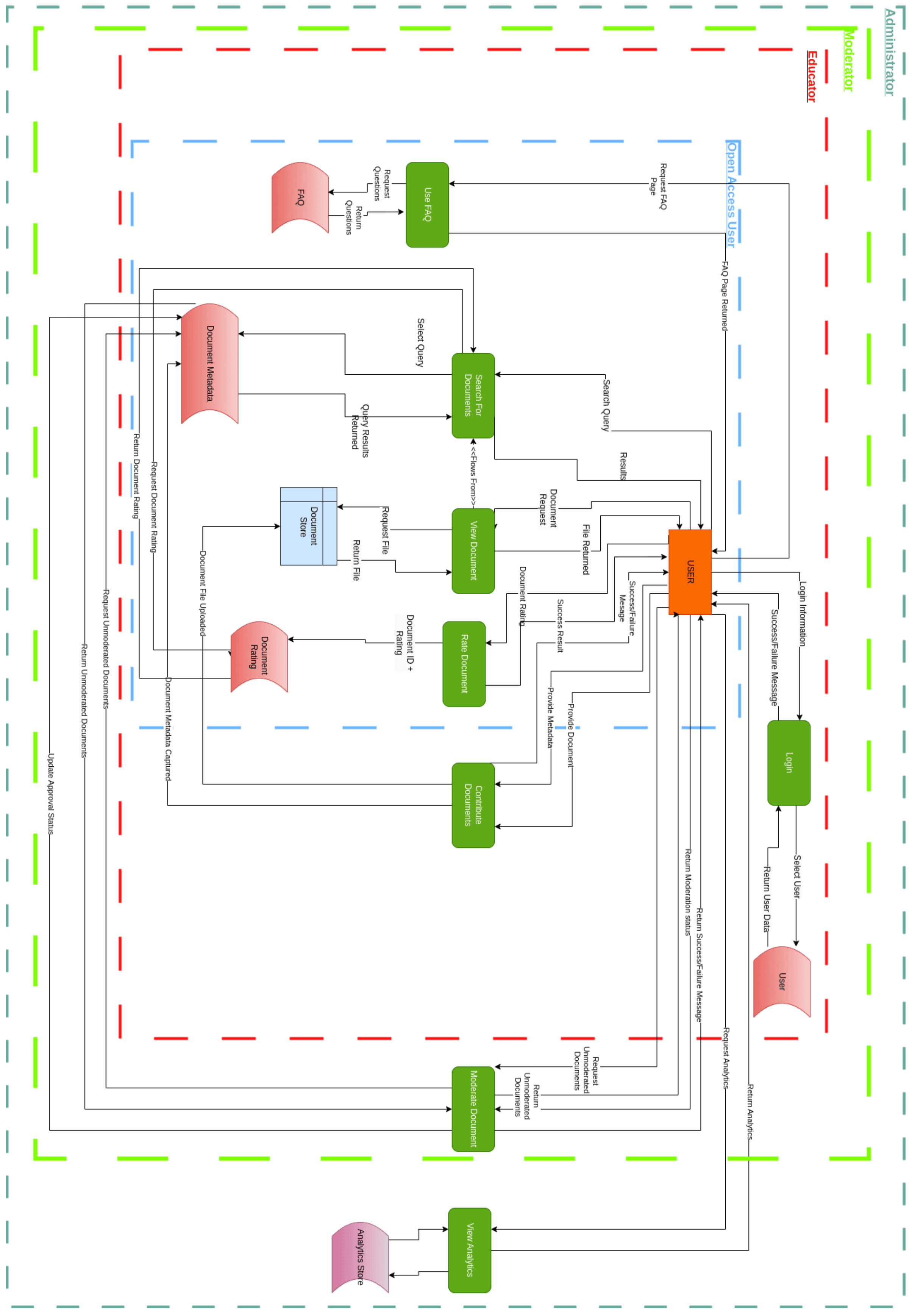
Appendix A: Glossary of Terms

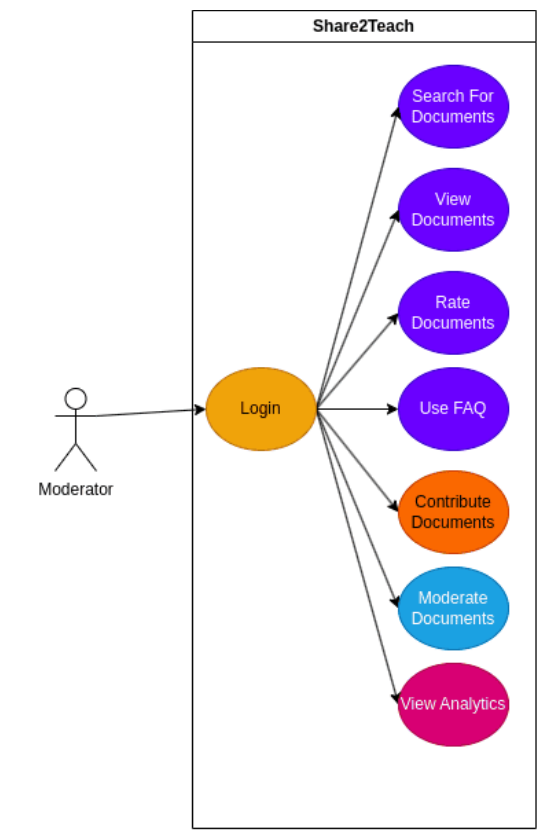
Term	**Definition**
HID	Human Interface Device. A method by which a human interacts with an electronic information system either by inputting data or receiving output.
Metadata	Data that provided information about one or more aspects of data. Used to summarise basic information about data to make tracking and manipulate data easier.

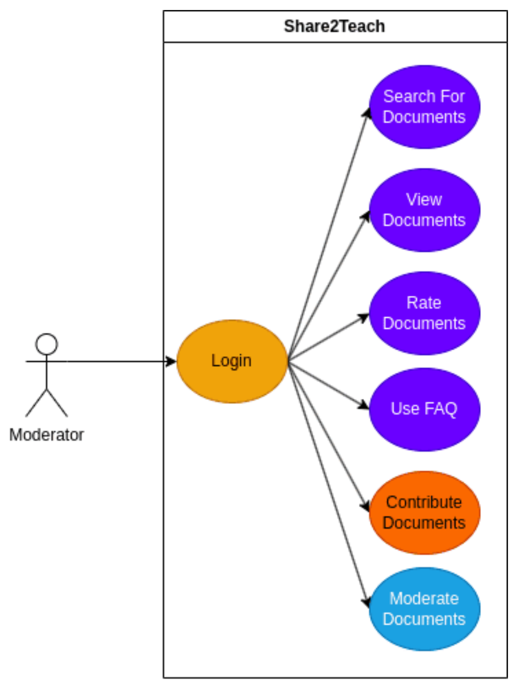
Appendix B: Issue List

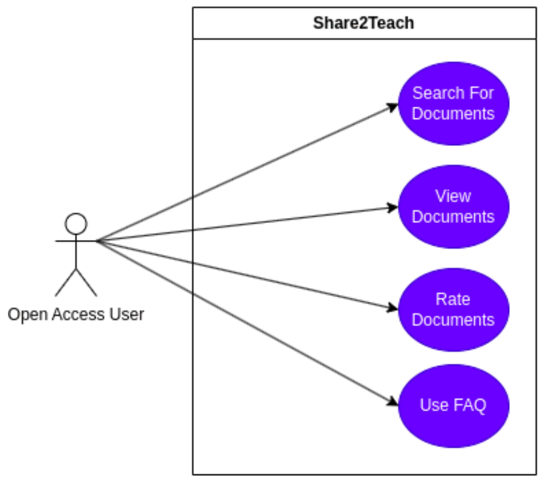
Date	Issue	Description	Version Resolved

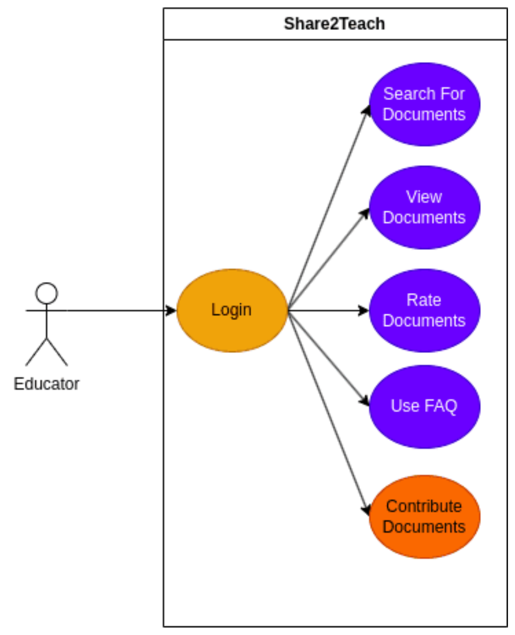
Appendix C: Analysis Models




Share2Teach
Moderator
Login
Search For Documents
View Documents
Rate Documents
Use FAQ
Contribute Documents
Moderate Documents
View Analytics


Share2Teach
Moderator
Login
Search For Documents
View Documents
Rate Documents
Use FAQ
Contribute Documents
Moderate Documents


Share2Teach
Open Access User
Search For Documents
View Documents
Rate Documents
Use FAQ


Share2Teach
Educator
Login
Search For Documents
View Documents
Rate Documents
Use FAQ
Contribute Documents