Software Requirements Specification

For:

General Education Database Management System (GEDM): A Horizontally scalable Database Management system and Web App for the Educational Industry.

Version 1.0

Prepared by:

Vivek Joy

Synopsis

Product Title:

General Education Database Management System (GEDM): A Horizontally scalable Database Management system and Web App for the Educational Industry.

Proposed product description

GEDM is a horizontally scalable database built using MongoDB. Database management systems used in the education industry have very limited scope for accommodating new types of data beyond the predefined data model that the DBMS is based on. Most institutions collect more data than they can deal with. This is because most Relational DBMS Software are not equipped to handle data outside of its schema, and a lot of data is thus not catalogued.

The General education database system is a NoSQL Database where Users can create a database with a schema that fits their particular needs and use case. Users can choose to incorporate a few pre-defined entities commonly found in the education industry, then pick and choose the attributes that they would like to have in their database. Users will be able to add new attributes to these entities, in addition to creating new entities all together.

The GEDM schema-less architecture is implemented using MongoDB. The database is hosted on MongoDB Atlas.

Users will be able to access, upload, edit and retrieve all of their data through the GEDM Website. The Website will also allow users to edit and map their datasets to the database.

The GEDM allows users to upload any kind of data to their database while still organising the data into structured collections for ease of access for further use.

Table of Contents

Introduction

Purpose

GEDM aims to provide a flexible data management tool that overcomes the shortcomings of commercially available DBMS used in education.

It is specifically designed to allow the users to expand the database according to the data that they collect each year. GEDM is built on a schema-less database management system, thus granting each user the ability to make the model for their database interactively using the Web app.

The web app also allows the user to edit their datasets directly, and also has features to map columns in csv/xlsx files to their dataset entities.

SRS Overview:

Version 1.0 :

- Prototype of Web app and implementation of the basic functionalities.
- Backend Code for creating dynamic schemas in MongoDB using mongoose.
- Integration of backend and frontend.

Product Scope

This software will allow users to interactively build databases for their specific purpose.

Since it has no underlying schema, users will be able to ingest all types of data that they may collect at their institutions. It also provides some template data models for common entities like student, employee, institution, course, assessment, etc. Each database will be specifically designed for the user thus ensuring that there are no redundant entities/attributes in the database.

Since many of the admins/teachers find commercially available DBMS complicated to use, the accompanying Web App will be easy to use and will allow data modelling to be done using simple dropdowns, buttons and input fields, and the backend scripts will do the job of creating the MongoDB database.

References

1. Commercially available SQL DBMS for education: https://ceds.ed.gov/

Overall Description

Product Perspective

Today, the amount of data collected far exceeds the capability to correctly store and utilize that data. It is not feasible to predict what kind of data will be relevant/collected in the future. This is a major shortcoming of SQL databases where schemas have to be strictly followed. Further, the users can not alter the underlying architecture of the database.

By allowing users to dynamically create and expand databases, GEDM provides users with more flexibility in data storage than commercially available software targeted at the education industry.

Product Functions

1. **Frontend:**
   - Web app where users/admins can register/login and view databases that they have created or have access to.
   - Upload and view data files before pushing to the MongoDB server.
   - Edit data entries.
   - Create MongoDB collections(entities), and allows the user to add nested sub-attributes in those collections.
   - Map data attributes to collections in the data model.
   - Upload data to server.
2. **Backend:**
   - Create MongoDB schemas based on user defined collection names.
   - Create a database based on the data model specified by the user.
   - Convert data sets (csv/xlsx files) to JSON objects and then upload to the data server.
   - Preserve relationships between different collections if they belong to the same data instance.

User Classes and Characteristics

1. **Administrators:** Logistics personnel for each institution. They have privileged access to the databases for their institution, allowing them to modify the underlying schema, upload new data, and create new databases. Administrators will also be able to modify user privileges.
2. **Teachers/Employees:** Users who will be allowed to upload/read data to the database, but cannot modify the schema of the database.
3. **Users:** Users within the institution who wish to view some data. They will be given read only access by default.

Operating Environment

GEDM is built using the MERN Stack:

- MongoDB — document database
- Express(.js) — Node.js web framework
- React(.js) — a client-side JavaScript framework
- Node(.js) — a JavaScript web server

Thus, the Web App will work on any modern web browser.

The Data Store is hosted on MongoDB Atlas and can thus be accessed by anyone with an active internet connection.

Design and Implementation Constraints

1. Hard to predict size of data entry, MongoDB has a limit on document size (16 megabytes), may have to design a way to split larger entries into smaller chunks.
2. Customers may want to use their own data servers for data storage, currently the product can only host on the cloud.

2.6 Assumptions and Dependencies

1. Assumtions:

    - Datasets are free of malicious code.
    - Database admins will not create invalid schemas.

2. Dependencies: (JavaScript Libraries):

    i. Client:

```
"@emotion/react": "^11.10.4",
"@emotion/styled": "^11.10.4",
"@mui/icons-material": "^5.10.6",
"@mui/material": "^5.10.8",
"@mui/system": "^5.10.8",
"@testing-library/jest-dom": "^5.16.5",
"@testing-library/react": "^13.4.0",
"@testing-library/user-event": "^13.5.0",
"axios": "^1.1.2",
"bootstrap": "^5.2.2",
"form-data": "^4.0.0",
"fs": "^0.0.1-security",
"mui": "^0.0.1",
"react": "^18.2.0",
"react-dom": "^18.2.0",
"react-router-dom": "^6.4.2",
"react-scripts": "5.0.1",
"reactstrap": "^9.1.4",
"rsuite": "^5.19.0",
```

```
"web-vitals": "^2.1.4",
"xlsx": "^0.18.5"
```

ii. Server:

```
"csv": "^6.2.0",
"csv-parser": "^3.0.0",
"dotenv": "^16.0.3",
"express": "^4.18.1",
"express-fileupload": "^1.4.0",
"generate-schema": "^2.6.0",
"mongoose": "^6.6.5",
"stream": "^0.0.2",
"xlsx": "^0.18.5"
```

External Interface Requirements

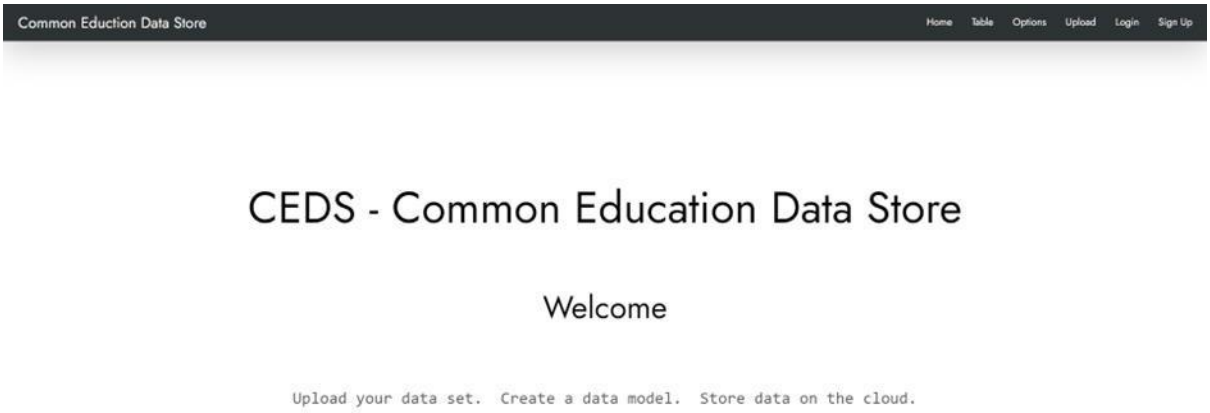
User Interfaces

The main interface between the user and the database is the Web App. Built using:

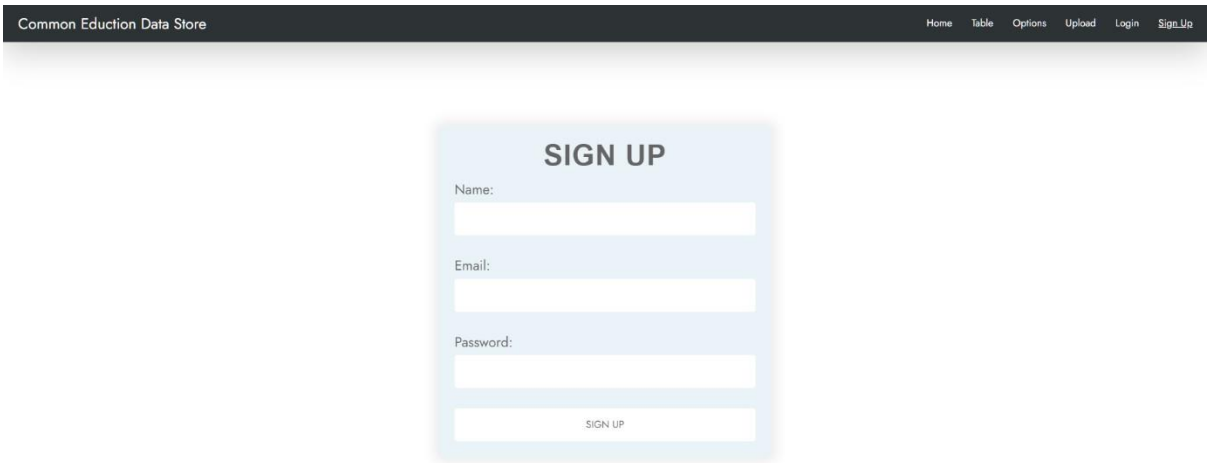
1. React.js
2. HTML/CSS
3. The following JavaScript Libraries are used for styling:
    i. Material UI interface
    ii. Emotion

Pages:

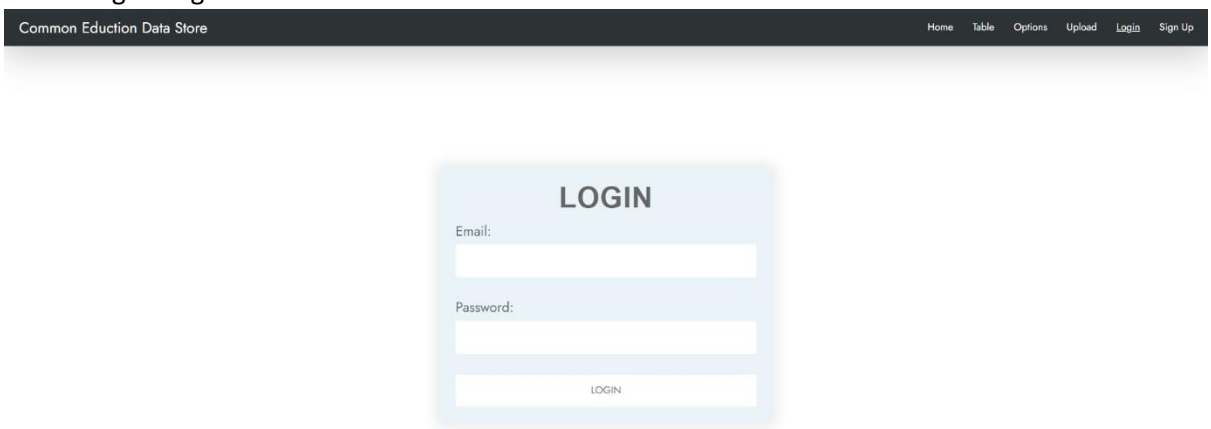
1. Home Page



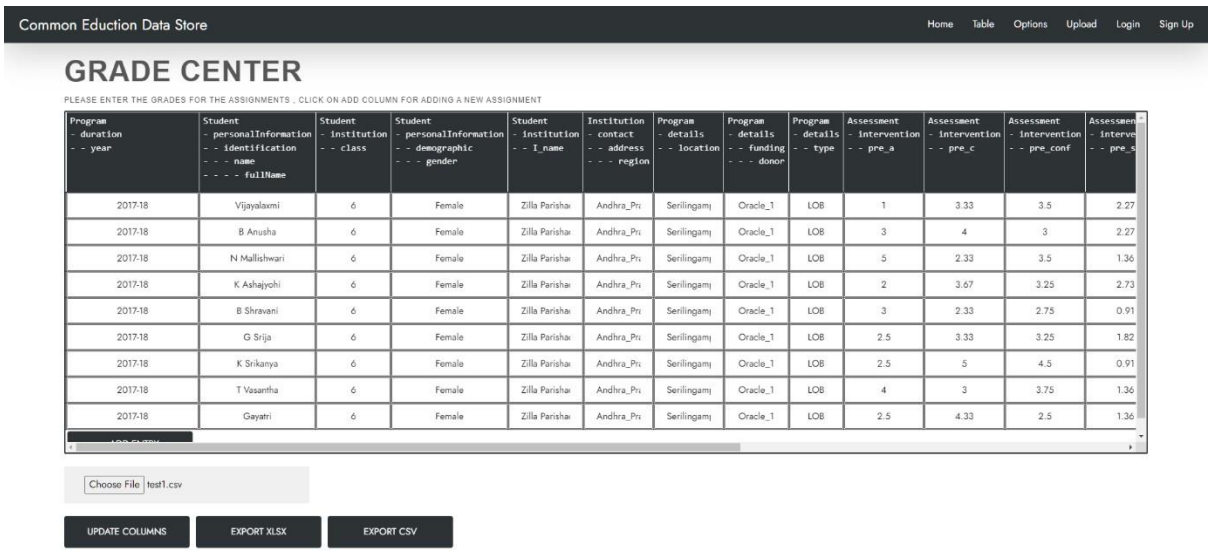
2. Sign up Page



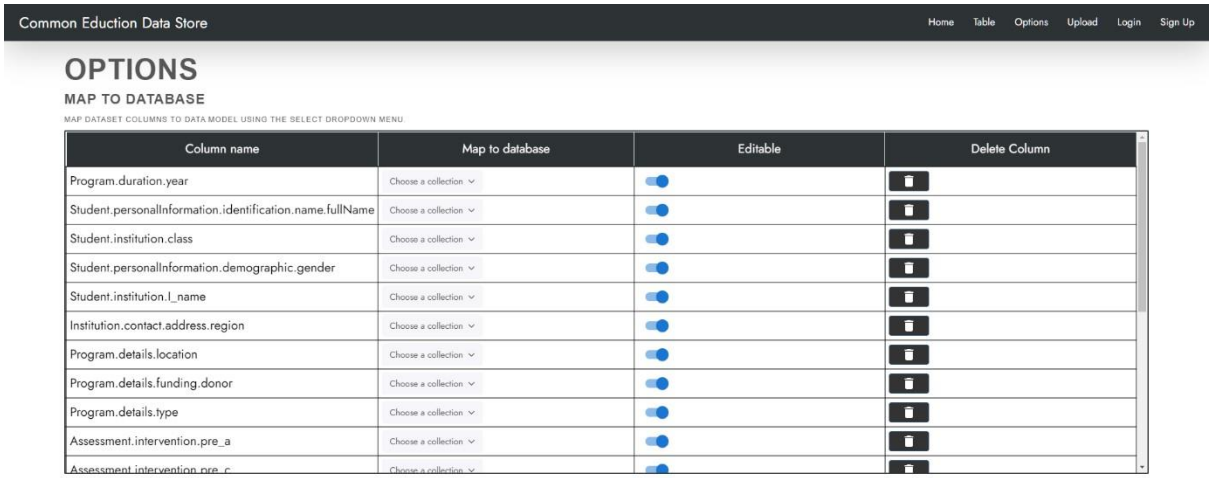
3. Log in Page



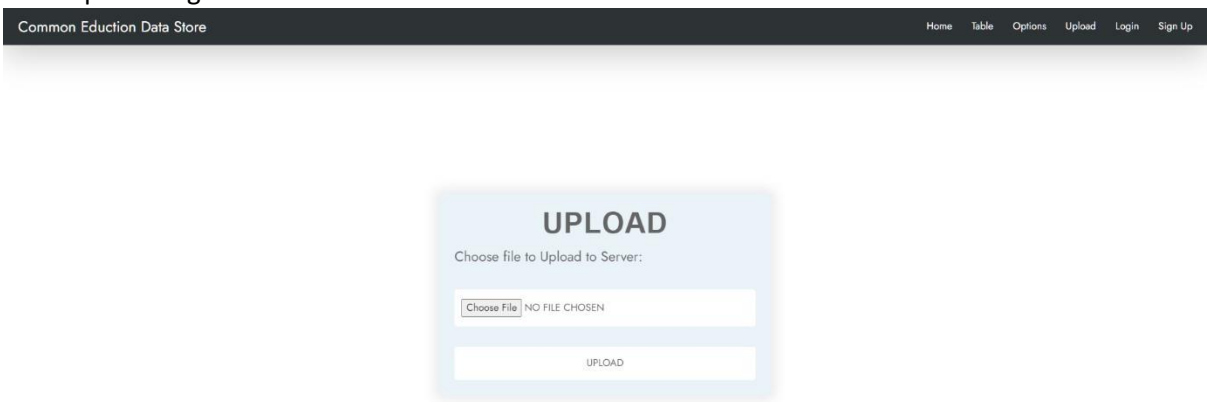
4. Table Editing page



5. Options / Map Column names to databbase collections page



6. Upload Page



Software Interfaces

The Web app runs on:

i. Node v16.16.0.
ii. MongoDB v5.0.9
iii. React v18.2.0 iv. Express v4.17.3
v. Mongoose v6.4.6

- Express is used as the backend web application framework for communication and routing of http requests and responses.
- Data transfer will be through backend JavaScript code that communicates directly with the data server.
- Mongoose is used to dynamically create data models for the database.
- The data files uploaded by the users will be shared across different components before eventually being pushed to the database. This is to allow the user to edit different aspects of the data, and also to allow the file to be used as the basis for building the data model.

Communications Interfaces

i. HTTP
ii. FTP
iii. MongoDB Wire Protocol

Analysis Models

Users will be given the option to use entities and attributes from a detailed pre-defined data model:

Overview:

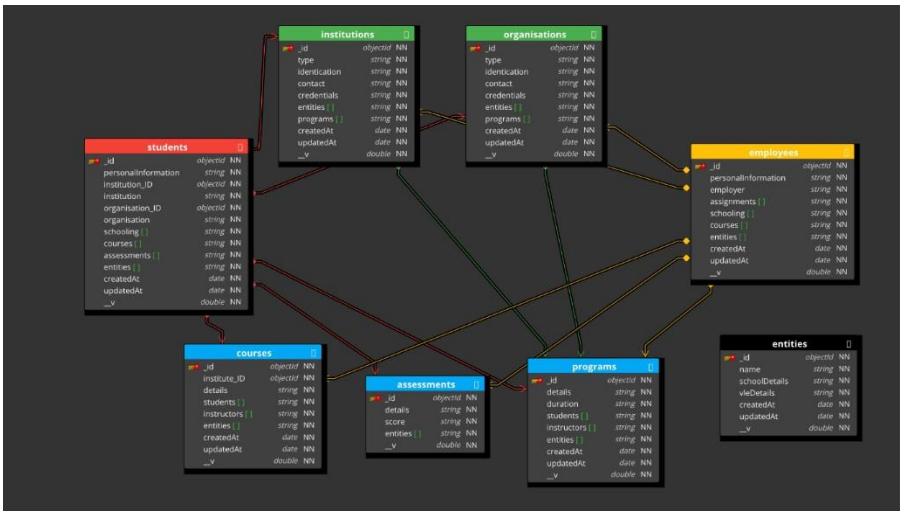


Fig i. Overview

Detailed:

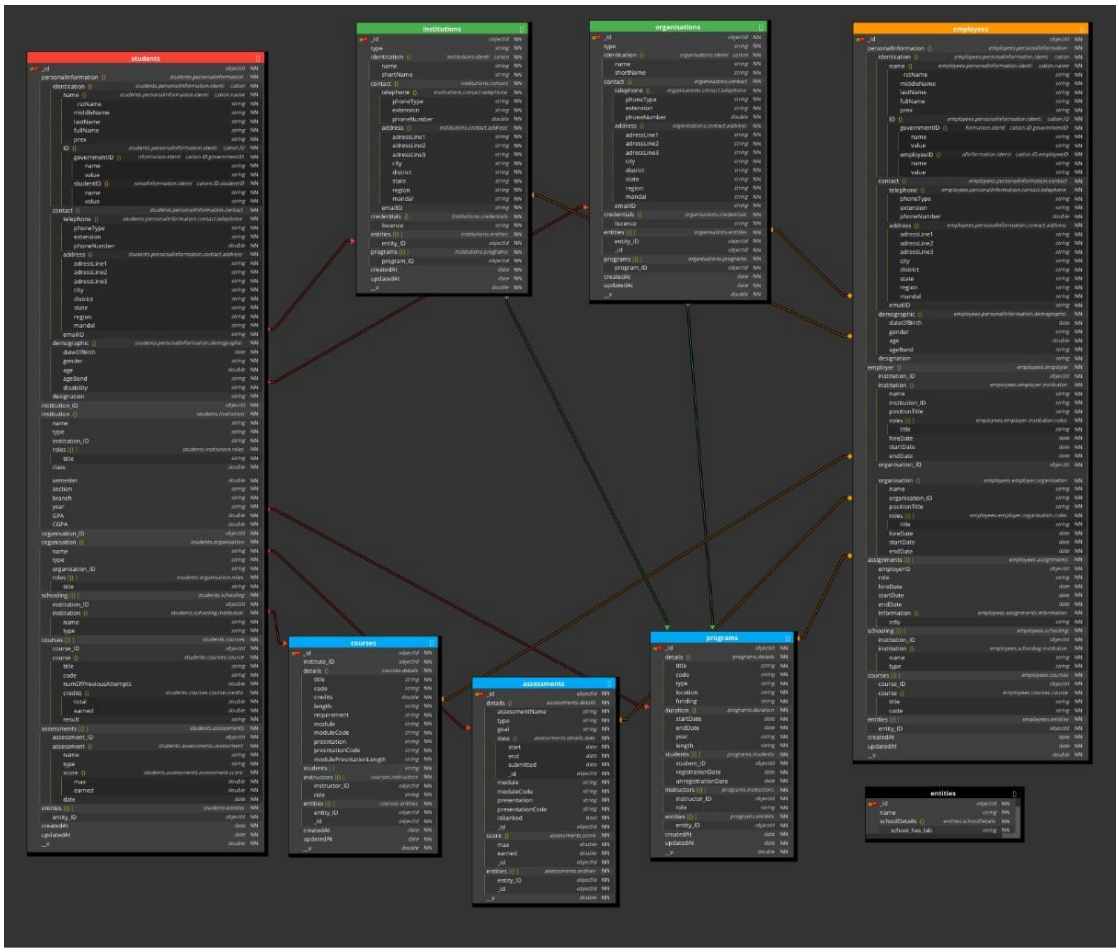


Fig ii. Detailed

System Features

The main features offered are:

i. View data files and edit data files on the web app.
ii. Map data file columns to mongoDB collections.
iii. Upload data file to database. iv. View and modify database.

i. View and Edit Data files

5.1.1 Description and Priority

- User can upload and view data sets directly on the web app and make changes to it before pushing it to the database.
- Priority: High

5.1.2 Stimulus/Response Sequences

- Upload File : Dialog box to choose file on system

5.1.3 Functional Requirements

TABLE:

•A table component to display the table on the Web App.
•Has to be able to dynamically resize based on the dataset. UPLOAD:
•A upload file button to allow the user to upload files.
•Error handling: Reject all files that are not of type .csv and .xlsx

ii. Map dataset to data base

5.1.1 Description and Priority

- User can make the data model for the database in this step by mapping data set columns to mongoDB collections with provision for nested sub attributes.
- Priority: High

5.1.2 Stimulus/Response Sequences

- Choose Collection : Dropdown that shows collections present in the database and also has an input field for new collections.

5.1.3 Functional Requirements

TABLE:

- A table component to display the table on the Web App.
- Has to be able to dynamically resize based on the dataset. CHOOSE_COLLECTION:
- Dropdown for choosing collections and creating new collections
- This component will also send data to the backend script that is going to create the mongoDB schema using mongoose.

iii. Upload data

5.1.1 Description and Priority

- User can upload their data set after mapping to database.
- Priority: High

5.1.2 Stimulus/Response Sequences

- Upload : Start uploading data to the mongoDB server.

5.1.3 Functional Requirements

UPLOAD:

- Backend script that pushes the data entries to the server.
- Must be able to process large amounts of data and avoid bottlenecks while pushing to different collections.

iv. View and modify database

5.1.1 Description and Priority

- User can view and perform queries on their database.
- Priority: High

5.1.2 Stimulus/Response Sequences

- Show Database: Display the database.

5.1.3 Functional Requirements

- TBD

Other Non-functional Requirements

Performance Requirements

For larger datasets, to ensure fast data transfer, code should be optimised to push data in batches. MongoDB Wire Protocol is preferred over HTPP in these cases as it is built to handle large volumes of data.

Safety Requirements

i. Since the user can choose any name for database entities, care should be taken to ensure that all user defined parameters can't cause breaking changes in the database.

Security Requirements

i. Privileged access must be restricted to trusted individuals.
ii. Datasets should be checked before uploading to database.

Software Quality Attributes

i. Horizontal scalability: Users are not restricted to any pre-defined data models
ii. Capable of handling data with errors instead of simply rejecting it.
iii. The Web app makes it very intuitive to upload data to the database and can help organisations structure their data for future use.

Business Rules

USER:

- Can only view database entries if he has the required authentication. ADMIN:
- Can create and edit the underlying data models and architecture of the database.

EDITOR:

- Can upload files to the database. Can edit these files on the web app before pushing, but cannot change the underlying architecture of the database.

Other Requirements

i. Data Validation: If the user wants to validate the data set before uploading it to the database, a data validation software must be used.

ii. New User validation: Must be provided by the institutions who use the product.

Appendix A: Glossary

1	**GEDM**	General Education Database Management System
2	**mongoose**	Mongoose is a Node.js-based Object Data Modelling (ODM) library for MongoDB. It is akin to an Object Relational Mapper (ORM) such as. SQL Alchemy. for traditional SQL databases. The problem that Mongoose aims to solve is allowing developers to enforce a specific schema at the application layer.

Appendix B: Figures

Figure Number	**Description**	**PDF File Name**
Fig i	Overview of pre-defined schema	overview.pdf
Fig ii	Detailed view of pre-defined schema	detailed.pdf