**Requirement Document**

**Project Name: School Database**

**Team Name: Admirable Blue Beatles**

**1 BUSINESS REQUIREMENTS**

* 1. **INTRODUCTION**

“Admirable Blue Beatles” aims to develop a comprehensive digital resource platform for school students focusing on three subjects: art, mathematics, and technology. The platform will include a database that contains reference materials, including articles with biographies of key individuals, major art objects, events, and important concepts in the specified subjects. The platform will be accessible to administrators, tutors, and students, each with varying levels of access and functionality.

**1.2 STATEMENT OF PROBLEM OR NEED**

Educational Database Solution:

Centralized Repository:

Develop a dynamic database as a centralized repository for course materials.

Enable efficient storage, organization, and retrieval of educational content.

Content Management:

Empower staff to seamlessly upload, update, and manage course materials.

Ensure a user-friendly interface for content contributors.

Information Accessibility:

Facilitate seamless browsing for students, allowing them to access reference materials with ease.

Ensure accessibility for administrators, tutors, and students through a mobile or web application.

User Empowerment:

Empower staff to contribute to the database, provide tools for tutors to modify and enhance existing content to align with evolving educational standards.

Innovation and Adaptability:

Acknowledge the evolving educational landscape and the need for innovation.

Design the solution to adapt to future educational trends and technological advancements.

User-Friendly Interface:

Prioritize the development of a user-friendly interface to enhance the overall user experience.

Ensure intuitiveness in navigation for administrators, tutors, and students.

Cross-Platform Accessibility:

Develop a mobile or web application for cross-platform accessibility.

Allow users to access the educational database seamlessly from various devices.

Security and Privacy:

Implement robust security measures to safeguard sensitive educational data.

Adhere to data privacy standards to ensure the confidentiality of user information.

Continuous Improvement:

Establish mechanisms for continuous improvement based on user feedback.

Iterate on the solution to incorporate emerging educational needs and technological advancements.

**1.3 BUSINESS REQUIREMENTS**

**1.3.1 LIST OF STAKEHOLDERS**

Main Stakeholders:

* Students
* Teachers
* Board of trustees
* School administrators

**1.3.2 CLIENTS**

Primary Client:

* Students
* teachers
* Admin

**1.3.3 LIST OF BUSINESS REQUIREMENTS**

Students:

* Browse Articles by Category:
  + Objective: Allow students to explore articles based on predefined categories such as art, mathematics, and technology.
  + Functionality: Implement a user interface that enables students to navigate through articles organized into distinct categories.
  + User Experience: Ensure an intuitive category-based browsing experience for easy content discovery.
* Browse Articles by Keyword in the Title:
  + Objective: Enable students to find specific articles quickly by searching for keywords within article titles.
  + Functionality: Implement a search feature that scans article titles for entered keywords.
  + User Experience: Provide a responsive and efficient search mechanism to enhance the precision of article discovery.

Tutors:

* Add or Modify Articles:
  + Objective: Empower tutors to contribute and enhance content within the database.
  + Functionality: Implement an interface for tutors to add new articles, including biographies, major art objects, events, and important concepts. Additionally, allow tutors to modify existing articles.
  + User Experience: Ensure a streamlined content management system for tutors, allowing for easy additions and modifications.

Administrators:

* Add, Modify, or Remove Articles:
  + Objective: Provide administrators with comprehensive control over the content within the database.
  + Functionality: Implement functionality for administrators to add new articles, modify existing ones, and remove outdated or irrelevant content.
  + User Experience: Create an administrative interface that facilitates efficient content curation, ensuring the database remains current and relevant.

**1.3.4 QUALITY REQUIREMENTS**

Non-functional Requirements:

* Reliability:

The system must provide a reliable and quick response to user queries.

* Usability:

The platform should be easy to use and user-friendly.

* Security and Privacy:

Adherence to appropriate data privacy standards, especially considering the sensitive educational context.

* Performance:

ensure the system's responsiveness and performance meet expectations.

**2 BUSINESS SOLUTION**

**2.1 OPTIONS CONSIDERED**

**Mongo DB**

We considered MongoDB as a potential database solution.

MongoDB is an open-source, no-SQL database management system. It has great documentation, a large community, and has been available since 2009.

From MongoDB’s official site: “MongoDB is an excellent choice if you need to: Support rapid iterative development. Enable collaboration of a large number of teams. Scale to high levels of read and write traffic. Scale your data repository to a massive size. Evolve the type of deployment as the business changes. Store, manage, and search data with text, geospatial, or time-series dimensions.[[1]](#footnote-1)”. Mongo DB is a great option for a school where there will be high levels of read and write traffic. It can store large amounts of data and still perform well. It is also great for ad-hoc queries.

MongoDB is a flexible database with no schema. It stores records as ‘documents’ which can include large files such as rich text documents, images, or video files. This would allow the client to store several types of educational content in the database rather than just text in the future. As many schools use video content, diagrams, and images in their lessons it is good future planning to choose a database with these capabilities. One issue with MongoDB is that data replication is a consequence of not being normalized. This does mean storage is not as efficient as a normalized/ SQL database.

MongoDB has replica sets and horizontal sharding built in and can scale horizontally across multiple servers in a way to optimize read/write functions. It is easy to do.

MongoDB uses Kerberos, X.509, and LDAP certificates to authenticate users. There is no risk of injection attacks like SQL due to the schema-free nature of the database.

**MySQL Workbench**

We considered MySQL as a potential database solution.

A main benefit of MySQL is that it is widely used and has longevity – it has been around since 1995, and many developers are familiar with SQL and MySQL as a program. There is a large and active community of users, and this would mean our client would be easily able to employ somebody who is familiar with SQL. It is also open-source and has an easy-to-use interface. As schools have limited budgets and resources this may be a key point to consider for our client.

MySQL works best with normalized, structured data. While we could organize the data required to suit these needs, the large text documents may work best in an unstructured format. When considering longevity of the database, many schools utilize video content either of recorded lessons or third-party educational resources. A MySQL database would not be suitable for this purpose. As MySQL also has a rigid schema, if the current schema needs to be changed the entire database will have to be migrated to the new schema.

Regarding scale and storage, MySQL can be vertically scaled, which would require upgrades to existing software, or horizontally scaled. Horizontal scaling can be done by creating replicas, table sharding, or functional partitioning. These solutions all have challenges involved and must be done manually. Replicas are a great option for sharding as well as backups, but they do create lag in performance.

Performance-wise, MySQL is great for queries and joins. It has slower write performance but as it is an indexed and structured database it is good for querying data quickly.

MySQL is particularly vulnerable to SQL-injection attacks and Denial of Service. Large text fields as required by the database may be at risk of SQL-injections. Correct authorization and input sanitation is important to consider.

**Single page web application or mobile application**

We considered either a single-page web application or mobile application. The key reason we choose to make a web app rather than a mobile-native app is that it allows users to select from a wider variety of devices.

It is likely that teachers will want to use their work laptops to upload to the database, and uploading substantial amounts of text can be difficult from small devices. Teachers may also want to only use work-issues devices and may not currently have a mobile phone issued by their employer.

Restricting to mobile only may also cause accessibility issues, one key issue is the need for keyboard control. The option for a physical keyboard is recommended and taking in mind users varying levels of access to accessories to their devices, having the option to use the application on desktop gives the best chance they will be able to use the database.

The National Government announced in their election campaign they intend to create laws so students must not use mobile devices in school. If this is implemented, students will not be able to access the database legally during school time if they are not allowed to have their mobile device. Ensuring the database can be accessed through devices the students will have access to is important, so the database is not quickly made unusable early in its life.

Finally, as the development team we analyzed our skills and the requirements and timeframe of the project and decided we are best suited to developing a web application.

We have decided to use React framework to develop this application as it is industry standard, and we have covered it in our course.

**Mongoose**

***Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box. – From the Mongoose Website*** (mongoose js, n.d.)

Mongoose is object document modeling layer which works with Node.js’s MongoDB driver.

One great benefit of Mongoose is the ability to make schemas. MongoDB is schema-less as it is NoSQL, which can make it difficult to retrieve data and keep consistent fields etc. The mongoose schema also allows easy validation such as required: true or type: string.

Mongoose also is generally said to be easy for beginners to MongoDB, which our team is making it a great choice.

**Node.js**

Node.js is an open-source server environment which uses JavaScript.

Node is great for building fast, scalable applications and works especially well with event driven applications. It can handle asynchronous calls and can process many requests with fast response times. Our client may have many different users making requests to and from the server, so Node is great for this use case.

NPM, Node Package Manager, is also a useful and easy way to manage packages and is another advantage of Node. Because there is a large community, there are also a lot of great packages available to use used with Node, reducing the amount of code we need to do from scratch. (Dziuba, 2023)

Node is also widely used, has a large developer community and therefore a lot of documentation and help for common issues.

**Express.js**

“Building a backend from-scratch for an application in Node.js can be tedious and time consuming. From setting up ports to route handlers, writing all of the boilerplate code takes away from what really matters, which is writing the business logic for an application. By using web frameworks like Express.js, developers can save time and focus on other important tasks.” (Codeacademy Team, n.d.)

Express.js is a great and easy solution to use for route handing, it works with Node.js and allows us to easily handle request and responses. It is almost always used with Node.js and they complement each other well. Due to a large userbase, there is a lot of help on forums and documentation as well as many tutorials available.

It also allows us to use middleware on our routes, which we will need to help implement authentication. Authentication for user login as well as user roles are required by our client and we can achieve this with express.

It also runs quickly and is faster than any node.js framework (Kinsta, 2023), which suits our clients needs for a responsive and efficient way to access their database.

**2.2 RECOMMENDED SOLUTION**

**2.2.1 SOLUTION STATEMENT**

After analyzing different potential solutions, our recommended solution is a single-page web application in React, which uses a MongoDB database.

We will use Nodejs, along with express and mongoose to help create the back end and query the database.

**2.2.2 KEY FEATURES**

#### Front End

Clean and clearly designed single-page application allows users to easily read, write and query the database.

* Web applications are accessible from a variety of devices, allowing more flexibility for each user’s needs.
* Authentication and sign-in functionality

#### Database

MongoDB database, which is a no-SQL database.

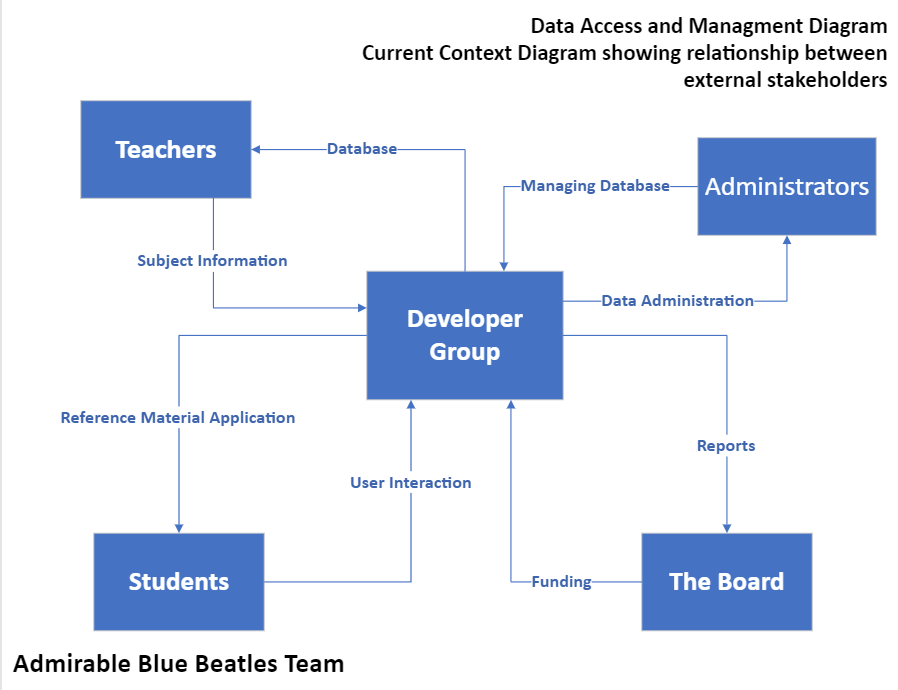
* Great read/write performance.
* Great for ad hoc queries.
* Built-in sharding and replica sets when scaling is needed.
* Stores data in documents, which works well for text files.
* Flexibility for changing future needs – can easily update to include different file types, such as video, if needed by client.

**2.2.3 UNIQUE VALUE PROPOSITION/JUSTIFICATION**

The key value of our proposed solution is flexibility. We have recommended a solution which allows our client to change and grow easily in the future, both by growth of the size of the database and by content stored in the database. Over the past few years, due to covid, we have seen a rapid change in how educational providers use technology and we appreciate room to grow and change is an important factor. Regarding our front end, we understand that different users may have unique needs – a disabled student may need a keyboard or a large screen, a teacher may only want to use work issued equipment, or a lower-income student may only have access to a mobile. By choosing a flexible solution, a web app which can be accessed from many devices, we allow our users the space to be flexible with how they access the database depending on their needs.

**3 SOLUTION REQUIREMENTS**

**3.1 CONTEXT DIAGRAM**



**3.2 USER ROLES**

* Students
* Teachers
* Administrators
* Board of Trustees

**3.3 BUSINESS DOMAIN MODEL**

Entities:

* Student
  + StudentID
  + First\_Name
  + Last\_Name
* Subject
  + SubjectID
  + SubjectName
  + article
* Article
  + type: String,
  + name: String,
  + born: { type: String, default: null },
  + died: { type: String, default: null },
  + nationality: { type: String, default: null },
  + knownFor: { type: String, default: null },
  + notableWork: { type: String, default: null },
  + about: { type: String, default: null },
  + year: { type: String, default: null },
  + medium: { type: String, default: null },
  + dimensions: { type: String, default: null },
  + location: { type: String, default: null },
  + designedBy: { type: String, default: null },
  + developer: { type: String, default: null },
  + objectID

**3.4 USER STORIES**

As a student I want to browse articles by category to have a simple way to find resources based on the subject I am studying.

As a student I want to browse articles by a keyword in a title to have a dynamic way of searching for articles without needing the whole title.

As a teacher I want to add or modify articles to keep the articles up to date with current education standards.

As a Administrator I want to add, modify or remove articles to make sure I can fully manage the articles.

**3.5 NON-FUNCTIONAL REQUIREMENTS**

**3.5.1 SECURITY REQUIREMENTS**

**3.5.1.1 Data Encryption**

The application should use https to encrypt data transmitted between the user's browser and the server in order to prevent interception of sensitive data.

**3.5.1.2 Authentication**

Use secure authentication methods, such as salting and password hashing, to prevent unauthorized access to user account information.

**3.5.1.3 Authorization**

Make sure that users can only access the information and features that are available to them based on their roles and permissions by enforcing proper authorization checks.

**3.5.1.4 Database Backup**

To ensure data recovery in the event of system failures or data breaches, the application's database should be backed up daily or weekly.

**3.5.2 OTHER QUALITY REQUIREMENTS**

**3.5.2.1 Performance**

Establish performance benchmarks, such as response times for various operations to make sure the application satisfies these benchmarks even during peak usage.

**3.5.2.2 Scalability**

Make sure the application can scale horizontally (by adding more servers) or vertically (by upgrading server resources) to handle an increasing number of users.

**3.5.2.3 Usability**

To make sure the user interface is simple and easy to use, conduct usability tests and user testing, and based on the results, make the necessary adjustments.

**3.5.2.4 Accessibility**

Make sure the web application complies with accessibility standards like the WCAG (web content accessibility guidelines) and is usable by people with disabilities.

**3.5.2.5 Maintenance**

To make the codebase maintainable by future developers and to promote collaboration within your development team, enforce coding standards and documentation best practices.

**3.5.2.6 Availability**

Make sure the application is highly available, reliable, and provides redundancy in case of server failures to reduce downtime.

**3.5.2.7 Testing**

Establish testing specifications, such as unit tests, integration tests, and user acceptance tests, to guarantee the accuracy and dependability of the application.

Good UI/UX

**4 SCOPE**

**4.1 ITERATION 1**

System Architecture

1. Provide the system structure, details of the components, and their interfaces.
2. Identify the technology stack to be used for database and application development.

Database Design

1. Define the database schema by identifying tables, fields, and relationships.
2. Explain how to format data on terms, categories, keywords, and user applications.

User Stories

1. Clearly identify the permissions associated with each user role.
2. Identify and define user roles, including students, tutors, and administrators.

Analysis

1. Review and analyze client-submitted project requirements.
2. Identify any ambiguities or differences in requirements that need clarification.

4.2 ITERATION 2

Sample Data Integration

1. Import and add the sample data from the provided SampleData.xls file to the database.
2. Ensure that the data is consistent with the design and relationships.

Database Implementation

1. Create tables defined in the System Design Document, including those for text, groups, keywords, and user roles.
2. Establish relationships between tables according to the Solution Design doc.

Data Validation

1. Set limits and rules to improve data quality
2. Implement data validation checks to ensure database integrity and stability.

4.3 ITERATION 3

Documentation Updates

1. Document any changes made to the questions during the development process.
2. Update the forms to include information on ongoing questions.

Query Development for Different Roles

1. Develop questions that allow students to analyze information as it is structured.
2. Use questions that allow students to search for information based on key words in the title
3. Develop queries that allow teachers to retrieve, edit, and add articles to the database.
4. Implement necessary permissions and access controls to ensure safety.
5. Create queries to make it easier for employees to add, edit, or remove articles.
6. Implement robust error handling to deal with issues that may arise during data modifications.

Test Database Queries

1. Take care of any problems found during testing and adjust queries as necessary.
2. Make sure the developed queries are accurate and reliable by thoroughly testing them.

**4.4 ITERATION 4**

UI / UX Development

1. Create the user interface design for the web application, making sure that it is easy to use and intuitive.
2. Include tools to add, edit, and remove articles as well as to browse articles by category and keyword.

User Authentication and Authorization

1. Put in place user authentication procedures so that users can safely log in.
2. Enforce user role-based access controls to guarantee appropriate authorization.

**4.6 NON-FUNCTIONAL REQUIREMENTS**

* 99.9% system uptime
* 24/7 availability
* High load capacity
* Fast response time (within 2 seconds)
* User friendly interface

1. https://www.mongodb.com/why-use-mongodb [↑](#footnote-ref-1)