Standards Document

This document defines guidelines for designing documentation and writing code.

# Technology Choices

This will be a web app, requiring a browser and an internet connection, hosted on Plesk. Persistent data is stored remotely in a SQL database which is also stored on Plesk.

### Front-end

* **React** – for creating the web user interface
* **Material –** for quick professional styling
* **JavaScript**  **& HTML** - for writing application code
* **Axios** – for sending requests to the RESTful Web API
* **Plesk** – for hosting our web app

### Backend

* **JavaScript** – for writing application code
* **Express** – for receiving HTTPS requests from the client
* **MariaDB (MySQL)** – for storing persistent data in a SQL database
* **Plesk** – for hosting our database

### Development Tooling

* **Github** will be used for source control
* A **prioritised requirements list** (PRL) will be used for high-level requirements
* **Github Projects** will be used as a Kanban board for task distribution
* **Github Issues** will be used to highlight issues

# Coding Conventions

### JavaScript

These are roughly based on the common JavaScript conventions described by W3 Schools: <https://www.w3schools.com/js/js_conventions.asp>

### Naming

|  |  |  |
| --- | --- | --- |
|  | **Naming** | **Example** |
| **Variables** | Camel case | let someNumber = 42;  const unchagedVar = ‘title’; |
| **Constants** | Capitalised | const PI = 3.14; |
| **Class names** | Pascal case | class SomeClass { } |
| **Function names** | Camell case | function dropTable() { } |

### Indentation

* Tabs or spaces can be used but all .js and .jsx files in source control should have each indentation level set to 4-character spaces (Visual Studio will convert tabs to this by default).
* Except for objects which fit on a single line, curly brackets should be on the same line as their identifier e.g.:

function myFunction() {  
 // …  
}

* Objects which fit on a single line should have a space inside the opening and closing bracket. e.g. { id: 1, data: { name: ‘object’, information: true } }

### Other

* Use semi colons to end lines.
* Strings are in ‘single quotes’ wherever possible.
* On the client side, the file structure inside the src folder should be as follows:
  + *pages*
    - *ComponentName*
      * *component-name.jsx*: for the React and HTML
      * *component-name.css*: if any CSS is needed
      * *component-name.controller.js*: if any more complex data processing is needed on the way to/from the API, which isn’t directly related to the function of the UI.
  + *services*
    - *functionality-area.service.js*: for each area of the app (e.g. users, questions, results, etc.), make API calls using Axois
* On the API side, the file structure inside the src folder should be as follows:
  + *routes*
    - *functionality-area.route.js*: for each functionality area of the app, receive API calls using Express and call the relevant function
  + *controllers*
    - *functionality-area.controller.js*: for the API-side data processing, database calls and error handling
* Folders with many files may be further broken down inside of component and controller folders.

## SQL

### Naming

|  |  |  |
| --- | --- | --- |
|  | **Naming** | **Example** |
| **Table names** | Camel case and plural | CREATE TABLE quizQuestions |
| **SQL keywords** | All upper case | SELECT \* FROM table |
| **Column names** | Camel case | SELECT firstName FROM users |

### Data Types

* VARCHAR should be used for text unless the text is long and freeform (then the TEXT type can be used)
* Dates should be stored in a UTC time format rather than location specific time. Dates should be ISO 8601 compliant <https://www.iso.org/iso-8601-date-and-time-format.html> like YYYY-MM-DD HH:MM:SS.SSSSS

### Other

* Prepared statements should always be used instead of string interpolation when writing SQL queries executed from a client.
* Inline SQL can be used instead of stored procedures with the requirement that the SQL uses prepared statements.

## Source Control

* Sensitive information should not be committed. Specifically, our database connection string should not be hard coded in our codebase but rather placed in an appsettings.json file which is not committed (Visual Studio has features to facilitate this)
* No binaries should be committed – we should ignore these with a .gitignore file (Visual Studio does this by default)
* No code should be directly committed to the main branch (except for the initial commit)
* All code going into main should be written in its own branch and only merged into main once a pull request has been approved by at least two authors

### Naming

* The production or “master” branch is called “main”
* There are no fixed rules for naming branches except that they should clearly describe the feature being implemented (branch name prefixes are allowed but not required)

# Design Documentation

Designs will be stored in our Github repository – the same repository where we have our code. This is to allow us to make use of source control and peer-reviews with our designs.

### Types of Designs

* **Screen/Wireframe diagrams** – for designing user interfaces
* **Class diagrams** – for showing some of our reusable code (at a high level)
* **Entity-Relationship diagrams** – for the SQL database schema
* **Pseudocode and flowcharts** – for describing program flow
* **Use-case diagrams** – for showing how a user interacts with a user interface

## Pseudocode Conventions

* Code should be designed to be understandable to non-programmers and not be associated with a particular programming language.
* Plain English should be used instead of symbols (except for common mathematical symbols)
* Keywords which mutate program state (SET), result in conditional logic (IF, ELSE-IF, ELSE), or result in program flow (WHILE, FOR EACH, GO TO) should be written in all caps.
* Functions should be written in all caps and be as descriptive as possible
* Each line should be numbered with a number and a number and letter for different indentation levels (see ./templates/psuedocdoe-template.docx)

### Examples

SET, IF, ELSE, ELSE IF, WHILE, FOR EACH, GO TO, OUTPUT, SAVE, LOAD

## When to use Pseudocode over a Flowchart

These decisions can be made at the designer’s discretion. Generally, pseudo code should be used to show linear logic, whereas a flowchart is good for showing logic with lots of branches.

## Documentation Design Tools

* Microsoft Word for Pseudocode
* Wireframe.cc ([Wireframe.cc – the go-to free, online wireframing tool.](https://wireframe.cc/)) for creating wireframes
* Digrams.net (<https://www.diagrams.net/>) for other diagrams