Standards Document

This document defines guidelines for designing documentation and writing code.

# Technology Choices

This is a desktop GUI for Windows – support for other platforms is out of scope. Persistent data is stored remotely in a SQL database which is stored on Plesk.

### Front-end

* **WinForms** – for creating the user interface
* **C#** - for writing application code
* **MySQL.Data** – for database querying – written in the main program (rather than it being proxied through another program such as a RESTful Web API).

### Backend

* **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

### C#

These are based on Microsoft’s C# Coding Conventions <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

### Naming

|  |  |  |
| --- | --- | --- |
|  | Naming | Example |
| Variables | Camel case | int someNumber = 42 |
| Constants | Pascal case | public const int Answer = 42 |
| Class, struct, and enum names | Pascal case | class SomeForm { } struct Vector2 { } |
| Interface names | Pascal case starting with an “I” | interface ISaveable { } |
| Method names | Pascal case | void DropTable() |
| Properties | Pascal case | string Name { get; private set; } |
| Event handlers | Pascal case starting with “On” | void OnClick(EventArgs e) { } |
| Windows Forms | Pascal case end with “Form” | class LoginForm { } |

### Indentation

* Tabs or spaces can be used but all C# files in source control should have each indentation level set to 4-character spaces (Visual Studio will convert tabs to this by default).
* Curly brackets should have their own line e.g.:

void Main()   
{  
 // …  
}

### Other

* Object oriented code should be the default however more procedural static classes and methods can be used where appropriate.
* Namespaces should be used to logically group classes together.
* C# source code files should be named after the class (or interface or class or enum) whose definition exists in the file. This should be PascalCase.

## 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 use 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

This 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.

## Design Documentation

We are using digrams.net (<https://www.diagrams.net/>) for creating diagrams and Microsoft Word for pseudo-code.

Design templates are stored in ‘./templates’.