





# 

[**1.0 Introduction**](#_va6maet9f3yw) **3**

[1.1 Purpose](#_zdw95vz4uusr) 3

[1.2 Target Audience](#_tyjcwt) 3

[**2.0 Black Box**](#_fwpgy2k3tf08) **4**

[2.1 Context Diagram](#_t7f0h6al1rer) 4

[2.2 Use Case Diagram](#_eb04krai26ik) 5

[**3.0 White box**](#_iykyvveu8eng) **7**

[3.1 Context Diagram (Detailed White Box)](#_2e05ubtnwx4d) 7

[3.2 Layered MVC Diagram](#_7xji3f7mrioe) 8

[**4.0 Software Architecture**](#_p8cwb1lii6pc) **9**

[4.1 Considerations and Alternatives](#_mqfnz9355l0r) 9

[4.2 Architecture Tiers](#_bunhade3cwgb) 9

[4.3 Object Model](#_u3jakrqb1gg1) 9

[4.4 DB Model](#_3rptc8bcg5d7) 9

[**5.0 Framework**](#_3vbo7j251zi8) **10**

[5.1 Considerations, Alternatives and Decisions](#_opy9l74y90p9) 10

[.NET Variants (Framework vs Core)](#_43nl9ejqrcnu) 10

[MVC vs Web Forms](#_cqinhb728prk) 10

[Entity Framework Strategy (Code-First vs Database-First)](#_r4iinw1rijx5) 10

[Other Presentation Framework](#_hxq7lz20dcbw) 10

[**6.0 Coding Standards**](#_t6x7711w70k9) **12**

[6.1 Languages](#_sg02zqjh6npl) 12

[C#](#_i6gulddbxw2q) 12

[Syntax](#_p82824gyyemk) 12

[Classes](#_olpsgn8gq5wf) 13

[Test Naming Standards](#_os2b031wyjvv) 13

[Test Class](#_28z6a7mlmxjr) 13

[Test Methods](#_2041qqwp9fa) 13

[Linq](#_5jmakfpuxl7q) 13

[JavaScript](#_f6e4yi3yhmx0) 14

[Syntax](#_36jzlz2eii9g) 14

[HTML](#_hzt50driw8a1) 14

[CSS](#_c9b4uo3buyhk) 14

[6.2 Best Practices](#_tj6otknyh968) 15

[6.3 Source Code Project Folders and Files](#_do2vd8tkt69n) 15

[Folder Names](#_euilwdnfv9l0) 16

[wwwroot Files](#_priscmxlxzwy) 16

[Class Files](#_3dl41uwdqxm) 16

[**7.0 Document Organization Standards**](#_uktk7gjouflc) **17**

[7.1 Storage and Folder Structure](#_4f0hhw3lphwf) 17

[TLC](#_ejjds532zwcj) 17

[Sprints Folder](#_doogyt3fu4jq) 17

[Retrospective, Actions, and Lessons Learned Register](#_9m2khsu8o07r) 17

[Sprint](#_sq9ttcdyoj06) 18

[Meeting Minutes](#_gykuzbbosl3o) 18

[Sprint Commitment Email](#_85m6qyjgvug2) 18

[Unit Test Report](#_re5y31m8z14a) 18

[**8.0 Testing Standards**](#_ibvuuym27qxa) **19**

[8.1 Black Box Unit Testing](#_r713j8bev0yv) 19

[XUnit Test Cases](#_9bw528prfu1k) 19

[8.2 White Box Unit Testing](#_h9nbfsskwte9) 22

[Tools/Frameworks](#_lrhk49xpy67q) 23

[Test Naming Standards](#_4qmjjgn1km6) 23

[Test Class](#_mfovpclm5x7p) 23

[Test Methods](#_t4ol683ig4dp) 23

[**9.0 Process**](#_8guign6f28di) **24**

# 

# 1.0 Introduction

## 1.1 Purpose

The purpose of this document is to provide relevant information about the coding and organizational standards of the TLC Digital Log Book (Pending Name). This document will cover the following topics of the project:

* Black Box View
* White Box View
* Software Architecture
* Framework And Tools
* Coding Standards
* Document Organization Standards
* Test Standards
* Process

## 1.2 Target Audience

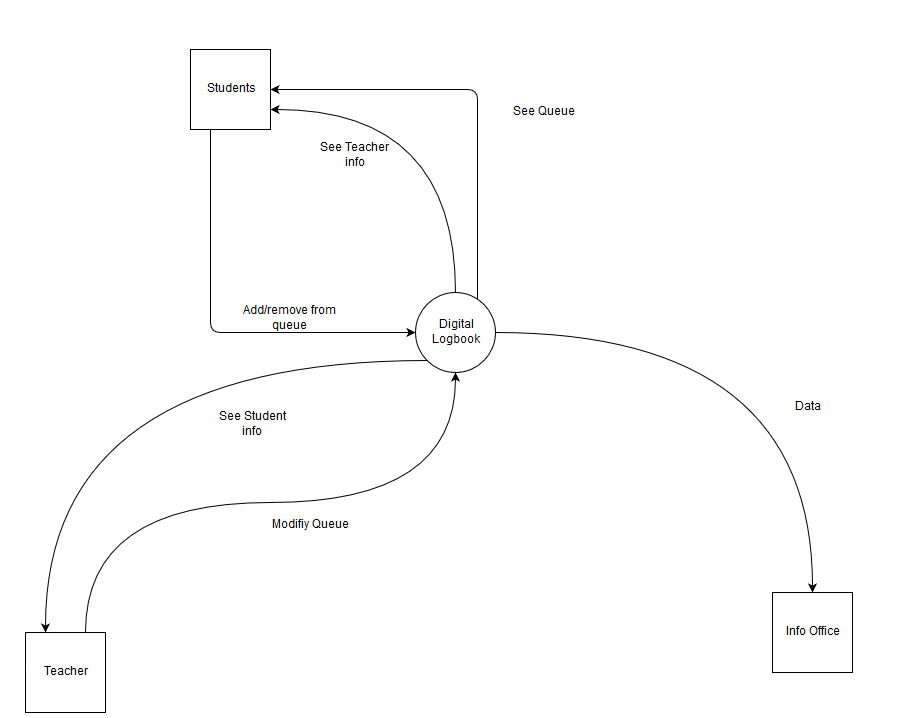
This document is targeted for use as a reference by third year Computer Science students or any other user that requires information about the TLC Digital Log Book System. The third year students developing or maintaining the system are responsible for updating this document in case of a change to the TLC architecture or code organization.

# 

# 2.0 Black Box

## 2.1 Context Diagram

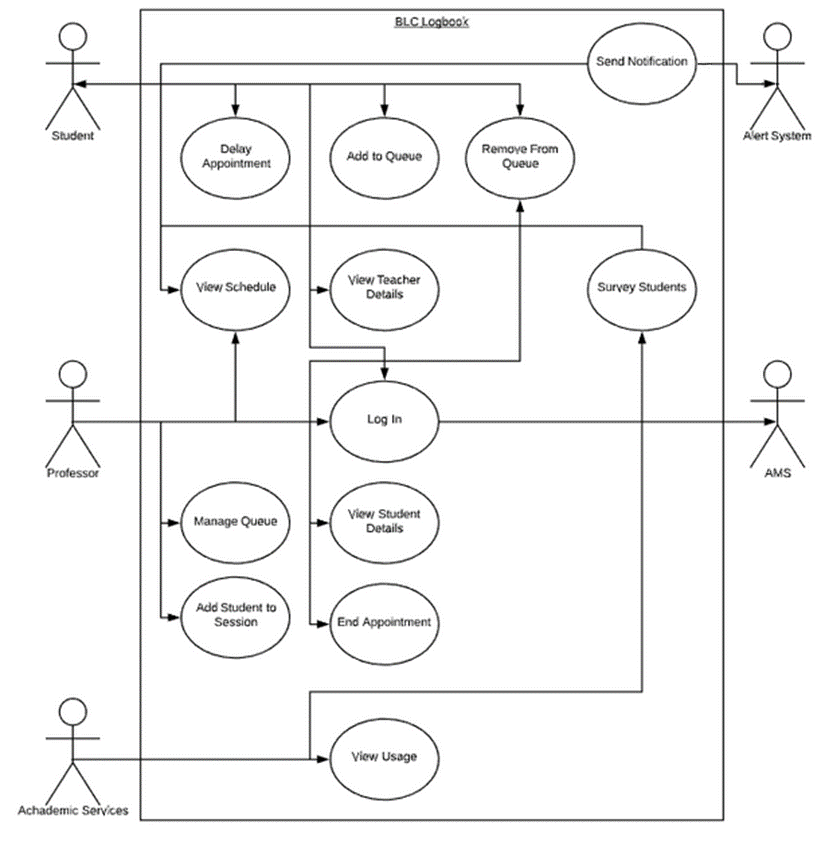
The following context diagram highlights the various interactions with the users, applications and repositories directly tied to TLC and how these applications interact with each other.



## 

## 2.2 Use Case Diagram

The use case diagram is a visual representation of the actors (roles) and their functionalities in our system. The flow of the data is also included as double-lined dashed arrows.

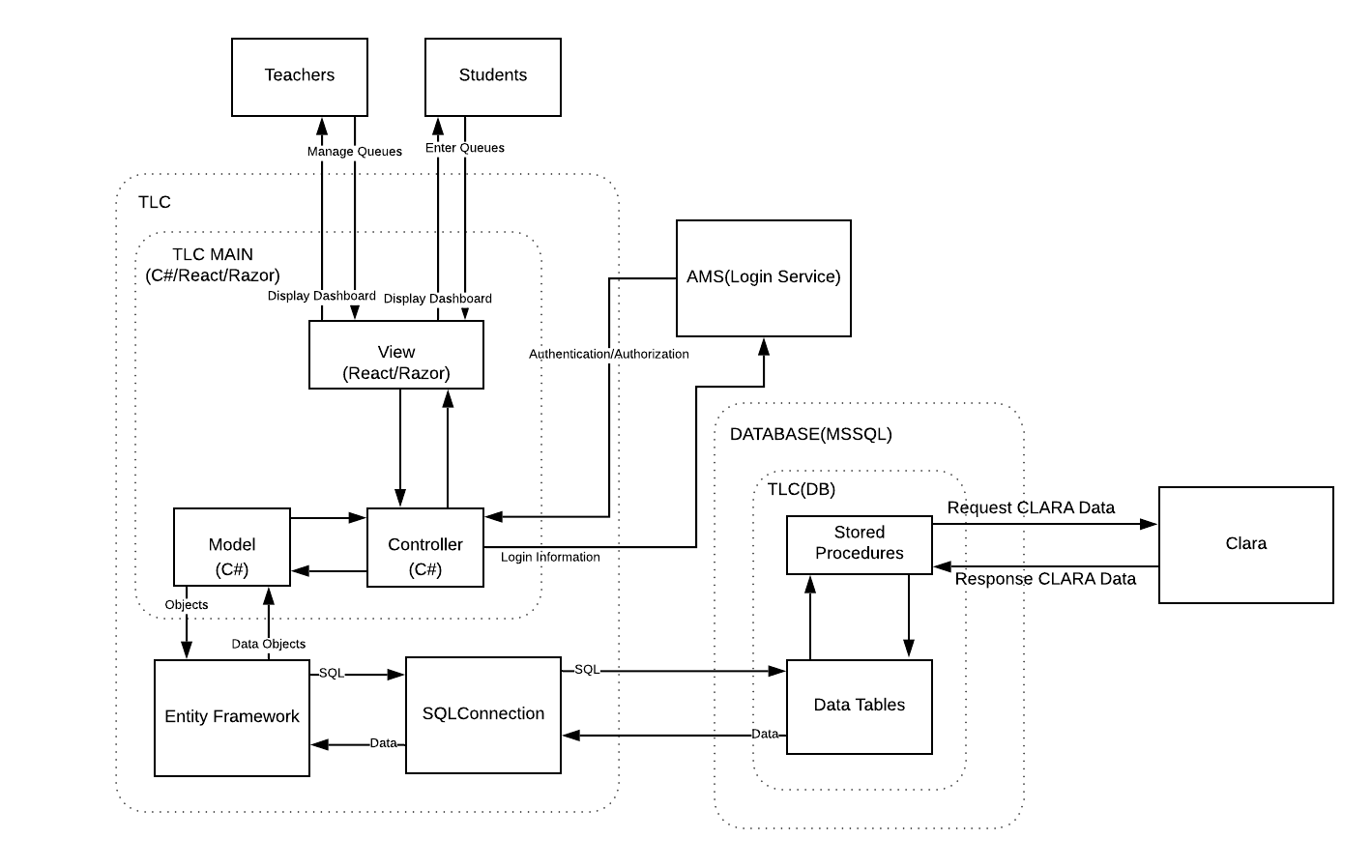


# 

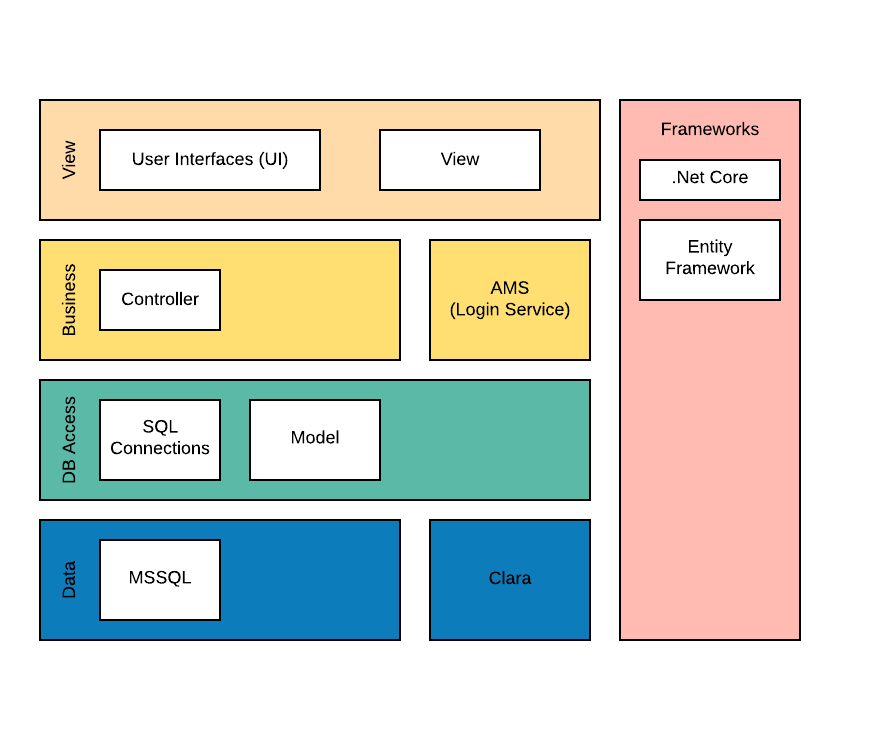
# 3.0 White box

## 3.1 Context Diagram (Detailed White Box)

The following context diagram highlights the detailed white box view of various interactions with the users, applications and repositories directly tied to TLC and how these applications interact with each other.



## 3.2 Layered MVC Diagram

The following layered MVC diagram highlights the various layers of architecture such as Presentation/View, Business, DB Access, and Data. The purpose of the View layer is to present the user interface to be shown and interacted with and passed the user inputs to the business layer. The purpose of the Business layer is to handle the business logic with what is passed from the View layer. The purpose DB Access layer is to provide a connection between the data layer and the Business logic layer that is used to access the data. The purpose of the Data layer is to store the data relevant to the business layer of the system. 

# 4.0 Software Architecture

## 4.1 Considerations and Alternatives

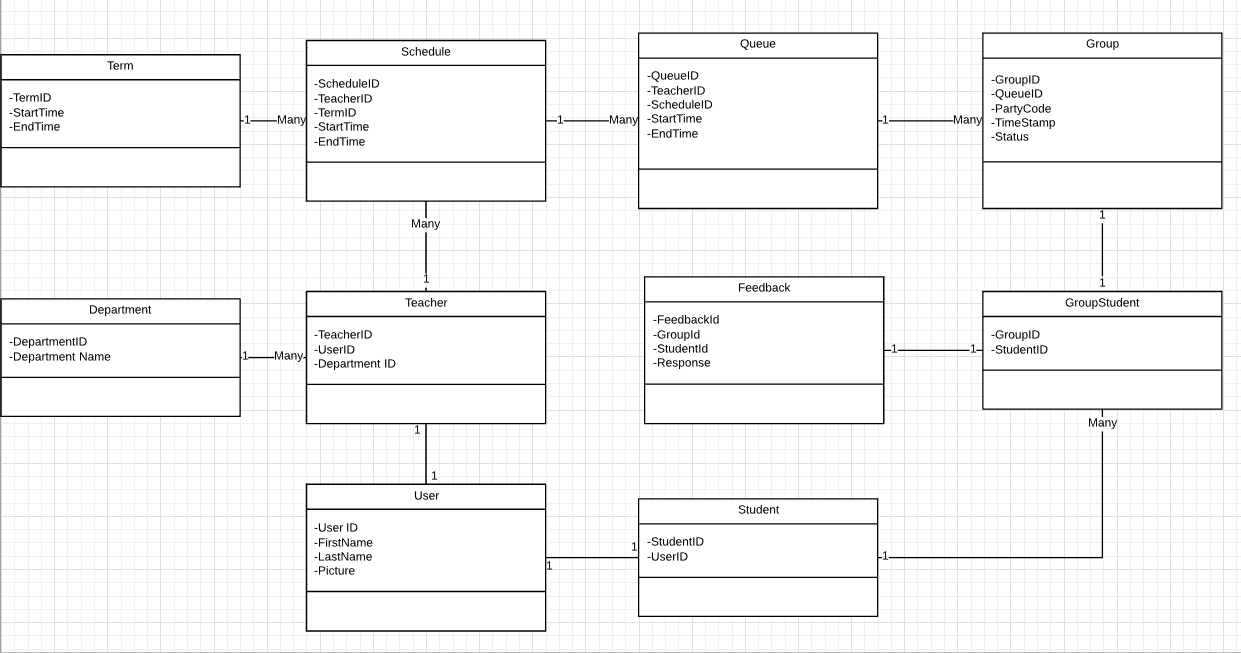
1. Having the ranking of the groups per queue held in memory by the server, not implemented as the queue would be wiped if the server crashed

## 4.2 Architecture Tiers

The majority of processing that is done on the server. No processing is done on the Database, it is used for storage only. The only thing that is done on the client side is refreshing the queue

## 4.3 Object Model

## 4.4 DB Model



# 

# 5.0 Framework

## 5.1 Considerations, Alternatives and Decisions

### .NET Variants (Framework vs Core)

Microsoft supports two variants of their framework: .NET Framework and more recently, .NET Core. The company has declared [.NET Core to be the future of .NET](https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/)[[1]](#footnote-0) and that alone should be sufficient as a justification to prefer it. The technological benefits are that Core features superior performance in all scenarios, is cross-platform which lends itself to being used by developers on non-windows platforms and is generally a superior development experience.

### MVC vs Web Forms

.NET Core does not support Web Forms. As such, we are using MVC.

### Entity Framework Strategy (Code-First vs Database-First)

The simplicity of the code-first strategy for new projects greatly improves productivity, as modifications to the database are done in a single code-base, managed with migrations and there is no auto-generated code to work with. Database-First would be preferred if the database already existed, which is not the case.

Clara will be integrated into the system using the database-first strategy as the database already exists.

### Other Presentation Framework

The primary design framework used for the presentation layer will be Bootstrap, with the official Heritage theme.

The TLC project will be dealing with real-time data, as users join queues, leave queues and otherwise create events which need to be reflected immediately in order to prevent users from trying to manipulate stale data. The pages will need to be able to continually update themselves, which may warrant the use of a front-end framework to make it easier to process incoming changes. The team is already familiar with modern JavaScript, which should make it straight-forward to adopt React and TypeScript, which are both simple libraries with major advantages in developer experience and maintainability through React’s declarative paradigm and TypeScript’s deep integration into IDEs.

Lastly, in order to handle communication with the server in a real-time manner, the system will require websockets. The library which is pushed by Microsoft for this purpose within the ASP.NET ecosystem is SignalR.

# 6.0 Coding Standards

## 6.1 Languages

### C#

Naming conventions in C# for common identifiers are as follows:

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Convention** | **Example** |
| Namespace | PascalCase | namespace System.Security { ... } |
| Type | PascalCase | public class Teacher { ... } |
| Property | PascalCase | public int Length { get; } |
| Method | PascalCase | public double CalculateSum(){ ... }; |
| Parameter | camelCase | public string FormatDate(DateTime date); |
| Field\* | \_underscoreCamelCase | private int \_number; |
| Constant\* | UPPER\_CASE | Private final static int FIXED\_NUMBER; |
| Local variable | camelCase | int number; |

\*exceptions to the Microsoft naming guidelines are denoted with an asterix

For details, the full Microsoft design guidelines can be found at: <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines>.

#### Syntax

C# syntax should follow common syntax standards such as:

* Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces).
* Curly braces follow the default Code Editor settings (each brace is on its own line).
* Write only one statement per line.
* Write only one declaration per line.
* If continuation lines are not indented automatically, indent them one tab stop (four spaces).
* Add at least one blank line between method definitions and property definitions.
* Use parentheses to make clauses in an expression apparent, as shown in the following code.

Further details on coding conventions can be found on Microsoft’s website: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>

#### Classes

C# Model class names map to the database. Entities and classes are named as singular (e.g. Student). Associative entities either have a name that represents the association between the entities (e.g. Registration), or a concatenation of the entities’ names (e.g. StudentCourse).

#### Test Naming Standards

##### Test Class

The name should be descriptive of what the project will be testing.

Bad Example: Test\_A

VS

Good Example: Test\_Add\_Calculation

##### Test Methods

The name of your test should consist of:

* The name of the method being tested.
* The scenario under which it's being tested.
* The expected behavior when the scenario is invoked.

Bad Example : Test\_Single

VS

Good Example: Add\_SingleNumber\_ReturnsSameNumber

#### Linq

For Linq, the system will use method syntax, as opposed to query syntax.

### 

### JavaScript

Naming conventions for common identifiers:

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Convention** | **Example** |
| Class | PascalCase | class Teacher { ... } |
| Property | camelCase | this.name |
| Method | camelCase | calculateSum(){ ... }; |
| Parameter | camelCase | function formatDate(date){ ... }; |
| Constant | UPPER\_CASE | const FIXED\_NUMBER = 5; |
| Local variable | camelCase | let number; |

For any specifics, the Google JavaScript is available for consultation: <https://google.github.io/styleguide/jsguide.html>.

#### Syntax

Noteworthy JavaScript syntax:

* Opening curly braces are at the end of a line, and closing braces are on their own line.

### HTML

HTML syntax is based on common W3C standards found here: <https://www.w3.org/standards/>. The Google HTML/CSS guide will be used for any specifics in syntax for HTML.

### CSS

CSS classes are to be named following the BEM naming conventions: <http://getbem.com/naming/>.

BEM stands for Block-Element-Modifier, where:

* CSS class names follow the “block\_\_element--modifier” form.
* Block is the description of the component using the class or containing the targeted element (such as a div intended to represent a queue, “queue” would then be the block part of the class name)
* Element is a part inside the component that is targeted by the CSS (e.g. a button inside the “queue” block)
* Modifier is the style to apply to the element or block (e.g. color-red).

## 6.2 Best Practices

Consult the following questions concerning best practices when evaluating code:

* Are the classes that require a dependency acquiring that dependency outside the class?
* Is the code the only example of code with its functionality?
* Is there no other code that matches the code being reviewed?
* Is the main goal of this class singular and modular?
* Does the code do one thing and one thing only?
* Is the low complexity/modular code being unit testing?
* Is Integration Testing being done from the controller downwards?
* Is an AntiForgeryToken being embedded into the HTML of the code?
* Is every POST back to the code validating the HTML AntiForgeryToken?
* Are possible database exceptions caught and handled?
* Are database returns checked that they will return what they are expected to?

A copy of the best practices can be found here at the following address in the S Drive:

S:/Computer Sciences/Projects/Development/2019/TLC/Code Review

## 6.3 Source Code Project Folders and Files

The source core project uses the default .NET Core MVC structure, with a few specifications in regards to further folder organisation:

* TLC (Web Project)
  + wwwroot
    - images
    - js
    - styles
      * theme
      * tlc.css
    - …
  + Migrations
  + Models
  + Controllers
  + Views
* TLC\_Test (XUnit Project)

The folders under the wwwroot folder exist to organise any CSS, images, and JavaScript that might be added to the project. Any theme styles (i.e. the Heritage web theme) are put into the theme folder, and any custom CSS is to be added to the tlc.css file, following the project CSS naming standards.

### Folder Names

Folders outside the wwwroot folder follow PascalCase. Folders inside the wwwroot folder are in camelCase.

### wwwroot Files

Files inside the wwwroot folder are in camelCase.

### Class Files

Class files have one class per file. The name of the file is the name of the class it contains, case-sensitive.

# 7.0 Document Organization Standards

## 7.1 Storage and Folder Structure

The project documentation is kept on a shared drive, in the dedicated TLC directory for all team members to access.

S:\Computer Sciences\Projects\Development\2019\TLC

The TLC directory:

* TLC
  + Sprints
    - TLC Team Action and Lessons Register.xlsx
    - 01\_STARTDATE\_ENDDATE
      * TLC\_SprintCommitment\_DATE
      * TLC\_SprintReview\_DATE.docx
      * TLC\_TestReport\_DATE
    - 02\_STARTDATE\_ENDDATE
    - …
  + ...

### TLC

Any files that are updated throughout the project are kept outside, at the same level as the Sprints folder. These are:

* Standards
* Action and Lessons Register
* Product backlog
* Operational documentation
* Architecture and design documentation

### Sprints Folder

All sprints are contained in the sprints folder.

### Retrospective, Actions, and Lessons Learned Register

The actions and lessons learned are tracked in the dedicated register (“TLC Team Action and Lessons Register”), that is updated throughout the project, and therefore kept at the same level as the sprints folder.

### Sprint

Each sprint has its folder in the Sprints folder. The naming standard for the folder is:

SPRINTNUMBER\_STARTDATE\_ENDDATE.

Example: 01\_2019-09-03\_2019-09-28

#### Meeting Minutes

Meeting minutes for sprint reviews are kept in their respective sprint folder.

The naming standard for sprint review meeting minutes is:

TLC\_SprintReview\_DATE\*

Example: TLC\_SprintReview\_2019-11-07

#### Sprint Commitment Email

The sprint commitment email is kept track of in its respective file.

The naming standard is:

TLC\_SprintCommitment\_DATE\*

Example: TLC\_SprintCommitment\_2019-10-29

#### Unit Test Report

Unit Test Reports are generated using Visual Studio’s code coverage reports.

The naming standard is:

TLC\_TestReport\_DATE\*

Example: TLC\_TestReport\_2019-11-07

\*dates in the file names are date sent, and for test reports the date generated

# 

# 8.0 Testing Standards

Testing will be done when all tasks under a User Story are past the active status. Whoever is assigned to the User Story is responsible for the tests. If a User Story is not assigned to anyone, then there is no code to be tested.

## 8.1 Black Box Unit Testing

Acceptance Test Driven Development (ATDD) involves team members with different perspectives, such as client, development and testing, in order to write acceptance tests in advance of implementing the corresponding functionality.

The three questions used in order to create these acceptance tests are :

* What problem are we trying to solve? (from a client perspective)
* How might we solve this problem? (from a development perspective)
* What about any unexpected cases? (from a testing perspective)

### XUnit Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| User is able to login | Can not log in twice |  |  |  |
|  | User Id and Password is authenticated and valid |  |  |  |
|  | User Id and Password is valid but not authenticated |  |  |  |
|  | User has access to respective content |  |  |  |
| Adding to queue works properly | Correctly add students to queue |  |  |  |
|  | Correctly add group of students to queue |  |  |  |
| Queue displayed properly for teachers | Show current session separately |  |  |  |
|  | Next in queue is emphasized |  |  |  |
|  | Display queue for the day |  |  |  |
| Queue displays properly for students | Display queue(s) for topic(s) for the day |  |  |  |
|  | Students can’t identify other students name in queues |  |  |  |
|  | Time estimation shown |  |  |  |
| Teacher can manage queue | Delete student from queue |  |  |  |
|  | Add student when empty queue |  |  |  |
|  | Put Students into group |  |  |  |
|  | Move student |  |  |  |
| Student can manage their session in queue | Student can delay themselves in queue |  |  |  |
|  | Student can cancel session |  |  |  |
| Reason for leaving session logged | Student asked why they’ve left queue |  |  |  |
| Notification is sent out properly | Student notified when their session is coming up |  |  |  |
| Teacher can manage schedule | Hours can be set by teachers |  |  |  |
| Student and teacher can view teacher schedule | Schedule is displayed properly |  |  |  |

### 

## 8.2 White Box Unit Testing

Unit testing is a development practice centered around testing software components in isolation from their surroundings and dependencies. The key focus of Unit Testing is improving software quality by identifying and resolving defects before they are leaked into production.

* **Without Mocking**: Write all code in the same class and pass the tests. Once the tests pass, refactor the code by extracting methods and classes. Make sure the tests pass.
* **With Mocking**: Instead of requiring to write all the code at once in the same class, just implement the flow by making use of dependent classes and make sure the tests pass. By doing this, one would ensure that method flow pass without having the need to write the entire code.

**Dependency Injection**

To be completed at a later date.

**Inversion of Control**

To be completed at a later date.

### Tools/Frameworks

We will be using XUnit for testing.

### Test Naming Standards

#### Test Class

The name should be descriptive of what the project will be testing.

Bad Example: TestA

VS

Good Example: Test\_Add\_Calculation

#### Test Methods

The name of your test should consist of:

* The name of the method being tested.
* The scenario under which it's being tested.
* The expected behavior when the scenario is invoked.

Bad Example : Test\_Single

VS

Good Example: Add\_SingleNumber\_ReturnsSameNumber

# 9.0 Process

The default TFS workflow fits our needs, and no amends to the existing process are necessary.

1. https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/ [↑](#footnote-ref-0)