Architecture Design and Process Report

THE CODING CHALLENGE SYSTEM

Heritage College | Development Project

Last Updated: December 10, 2021

Contents

[Introduction 2](#_Toc90043028)

[Black Box View 2](#_Toc90043029)

[White Box View 3](#_Toc90043030)

[Software Architecture 3](#_Toc90043031)

[Architecture Tiers 3](#_Toc90043032)

[Object Model 4](#_Toc90043033)

[Data Model 5](#_Toc90043034)

[Presentation Frameworks 5](#_Toc90043035)

[MVC vs Webforms 5](#_Toc90043036)

[.NET vs .NET CORE 6](#_Toc90043037)

[Code-first vs Data-first 6](#_Toc90043038)

[Coding Standards and naming conventions 6](#_Toc90043039)

[C# .NET Core 6](#_Toc90043040)

[LINQ 7](#_Toc90043041)

[Testing 7](#_Toc90043042)

[Security 8](#_Toc90043043)

[Views 8](#_Toc90043044)

[Formatting 10](#_Toc90043045)

[Folder Structure and File Layout 10](#_Toc90043046)

[Document Organization Standards 12](#_Toc90043047)

[Storage and Folder Structure 13](#_Toc90043048)

[Directory 13](#_Toc90043049)

[CCS Files and solution 13](#_Toc90043050)

[Sprint 13](#_Toc90043051)

[Test Standards 14](#_Toc90043052)

[Black Box 14](#_Toc90043053)

[White Box 17](#_Toc90043054)

[Process 17](#_Toc90043055)

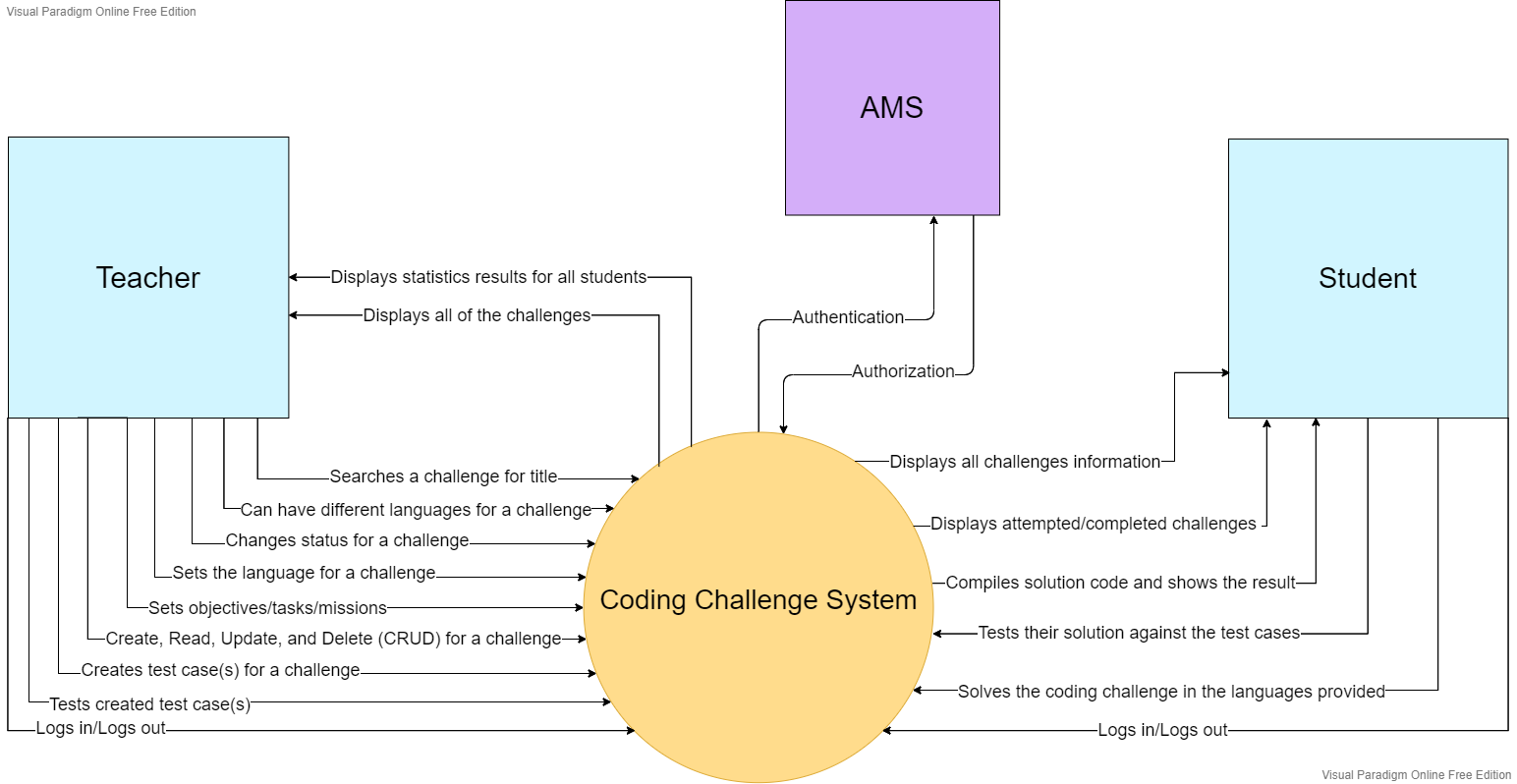
[Appendix 19](#_Toc90043056)

[Hosting Judge0 with Docker 19](#_Toc90043057)

# Introduction

The purpose of this document is to outline all the requirements set into place for the CCS project and to provide resources used and the setup process our team is carrying out before implementation phase. This document consists of black/white box views, S/W architecture, framework and tool decisions, coding, and document organization, and testing standards, as well as the process on we’ll be using each status for a task.

# Black Box View



# White Box View

A picture containing timeline

Description automatically generated

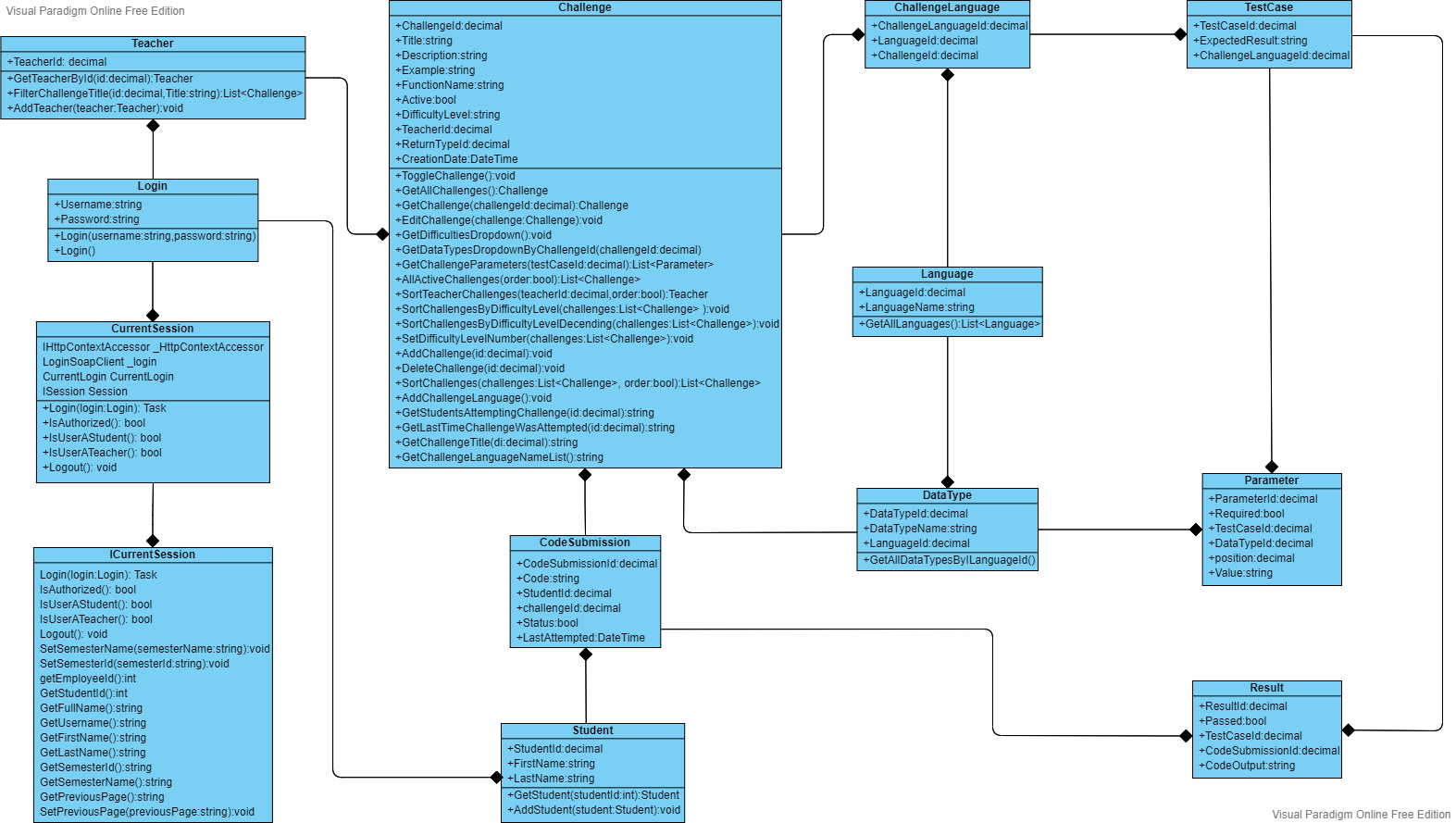
# Software Architecture

## Architecture Tiers

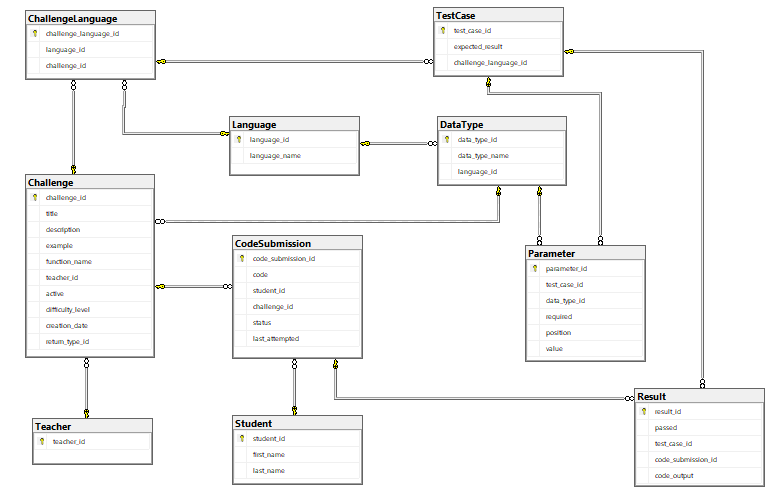
For this project, we will be using the Model View Controller (MVC) design pattern. As a best practice, we will be keeping any database access and logic in the Models, keeping the Controllers skinny and the Views with minimal logic. As such, access to AMS will be done through the model. Validation will be done in the Models and Views, to ensure maximum robustness. Most of the code will be server side, except for client-side validation in the Views for any forms.

This system will interact with two outside systems. AMS will be used for authentication and authorization, and a self-hosted Application Programming Interface (API) to compile code. No extensive configuration is needed for AMS, as it is already hosted on CSDEV, a windows server run by Heritage College. For the self-hosted API compiler, we have successfully deployed it with Docker on a Windows 10 Home machine (B218-16) (Steps to reproduce in appendix). However, to use this in Production, the API will need to be hosted on CSPROD, a windows server. We have not yet found a way to host this API on a windows server, but it’s a low priority as for testing purposes we can use the API hosted on [B218-16](http://b218-16.cegep-heritage.qc.ca:2358/dummy-client.html).

## Object Model



## Data Model



# Presentation Frameworks

The Coding Challenge System will use the Heritage College stylesheets and Bootstrap for its CSS. We will use the most recent version of bootstrap which is Bootstrap v4.3.1.

## MVC vs Webforms

The first consideration our team had was whether to use Model–view–controller (MVC) or Webforms. MVC is a new standard design Pattern in Software development. Compared with Webforms, MVC provides more control over HTML, jQuery, JavaScript, and CSS. MVC also provides better testability of the Web Application and good support for Test Driven Development (TDD). These features are what our team want. Our team determined that MVC is better than webforms for our project. We decided that an MVC approach using .NET Core will be our best decision. The team made this decision due to the facts that MVC is more logically organized, and the separation of layers will be more consistent, and it is currently the Microsoft supported framework going forward. This level of support will allow for a longer lifespan of the system before needing to be reworked for newer technologies.

## .NET vs .NET CORE

The second consideration was whether to use .NET Core or .NET Framework. We decided to use .NET Core. The team reached this decision because we hope our application will be used for a long time after we are gone, and the .NET Framework will no longer be supported after April 26, 2022. We decided to use .NET Core v3.1 since it is the newest version. This version will no longer be supported in December 2022.

## Code-first vs Data-first

Our third consideration was whether to use code-first or data-first. We chose to do data-first because seeing what data and what type of data we are going to use before having to code it, will make the process easier. Making the database first will also reduce the possible risks because it gives a strong sense of the system overall structure.

# Coding Standards and naming conventions

## C# .NET Core

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Convention** | **Example** |
| Namespace | PascalCase | namespace **MlC50Store.Models** |
| Class | PascalCase,  Noun or noun phrases; | public class **User** {} |
| Property | PascalCase | public int **UserId** {get; set;} |
| Methods | PascalCase | public void **JoinQueue**() {} |
| Parameter | camelCase | public void JoinQueue(int **userId**) {} |
| Local Variables | camelCase | int **userId**; |
| Brackets | Egyptian Style | public void JoinQueue() **{**  **// code**  **}**  if (condition) {  **// code**  } else {  **// code**  } |
| Readonly property | camelCase with underscore | private readonly **\_db** |

## LINQ

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Syntax** | **Example** |
| LINQ Queries | Both Query and Lambda are acceptable | **Query:**  var teachers = from teacher in teachers select teacher;    **Lambda:**  var teachers = await \_db.Teachers.OrderBy(g => g.Name).ToListAsync(); |

## Testing

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Convention** | **Example** |
| Test Project Name | Should be name of the project that is being tested + “Test” all in PascalCase | **MlC50StoreTest** |
| Namespace | PascalCase | namespace **MlC50StoreTest** |
| Class | PascalCase | public class **UpdateBuyPriceTests**{} |
| Methods | Pascal\_Snake\_Case | public void **Test\_Join\_Queue**() {} |

## Security

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Convention** | **Example** |
| Model Validation | - Data Annotation,  Custom Validations | [Required(ErrorMessage = "Title is required")]  public string Title { get; set; } |
| SQL Injection | - Antiforgery Token, on class or method | [ValidateAntiForgeryToken]  [HttpPost, Route("Semester/Add")]  public IActionResult Add(Semester semester) {  ...  } |
| Cross Site Scripting | - Encode all unvalidated input, using built in Razor, JS Encoder, or Html Encoder.  - More info: [Prevent Cross-Site Scripting (XSS) in ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-5.0#:~:text=HTML%20Encoding%20using%20Razor%20The%20Razor%20engine%20used,encoding%20rules%20whenever%20you%20use%20the%20%40%20directive.) | HtmlEncoder \_htmlEncoder;  JavaScriptEncoder \_javaScriptEncoder;  UrlEncoder \_urlEncoder; |

## Views

|  |  |  |
| --- | --- | --- |
| **Category** | **Rule** | **Example** |
| Razor Views | - ASP HTML tags will be used as opposed to HTML helpers |  |
| HTML/CSS | - Google’s HTML and CSS standards will be used. | Link: <https://google.github.io/styleguide/htmlcssguide.html> |
| - All identifiers will have meaningful names  - Kebab-case  - Should have space between each CSS code block |  |
| - CSS headers that are too long should be listed line-by-line or completely changed |  |

## Formatting

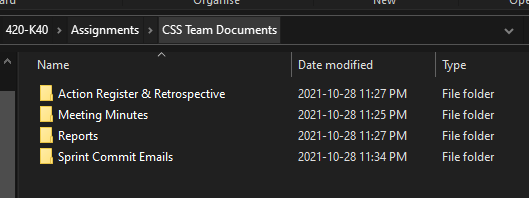
|  |  |  |
| --- | --- | --- |
| **Category** | **Rule** | **Example** |
| Indentation | - Four spaces will be used in indentation: | Refer to the above examples. |
| Parenthesis | - Always use parenthesis for clauses in expressions  - All braces will be on their own lines | if **(\_**cs.IsAuthorized()**)** {  return View();  } |
|  |  |  |

## Folder Structure and File Layout

|  |  |  |
| --- | --- | --- |
| **Category** | **Rule** | **Example** |
| Grouping CSS in Folder | - View-specific CSS should be in its own file  - All common CSS will be in main.css |  |
| Grouping CSS in File | - Group CSS by the flow of the page  - Label each group with a comment |  |
| Libraries | - Libraries should be put in their own folders |  |
| Images | - Images should be put in their own folder |  |

# Document Organization Standards

The document organization standard which the CCS team is following is to have all essential documentation stored on school servers under the directory “CCS Team Documentation”, where sub-folders are organized to have specific files assigned to each of them. Our naming standard for the files are for each file to start with “CCS\_” followed by the file topic, which gets placed in the respective folder.



## Storage and Folder Structure

All project documentation will be kept in the appropriate CCS project on CS AZURE. All members of the CS AZURE project will be able to access these files.

## Directory

All documents will be stored on CS AZURE to ensure that the team and project manager have access to all up‑to‑date documents required for the project:

## CCS Files and solution

The files and solution for the system will be kept in a separate folder called CCS inside the first CCS folder. (CCS-> CCS)

## Sprint

All sprint documents will be put in the appropriate sprint’s folder such as meeting minutes, sprint review, etc. These documents will follow the same naming conventions as the directory example above.

# Test Standards

## Black Box

Acceptance Test-Driven Development (ATDD) is a synergetic process where a development team, testing team, and business representatives all come together to task out requirements, identify potential drawbacks, and to reduce the likelihood of errors before coding is started.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| Entire system should work appropriately and not fail for different browsers. | Chrome functionality is successful. |  |  |  |
|  | Firefox functionality is successful. |  |  |  |
|  | Microsoft Edge functionality is successful. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| Valid account should be accepted. | Should be a successful login |  |  |  |
| Invalid account should be denied. | Error message should notify the user that the entered login details has been denied. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| The compiler must interpret code in under 5 seconds | Code is interpreted in under 5 seconds |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| User can login | Chrome |  |  |  |
|  | Firefox |  |  |  |
|  | Microsoft Edge |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| The teacher is successfully able to edit a challenge | Successful update of a challenge with proper input, all fields valid. |  |  |  |
|  | Unsuccessful update of a challenge due to incorrect input, at least one field is invalid. |  |  |  |
| Teachers should not be able to edit...{something} |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| Teacher is successfully able to delete a challenge | Cascading works successfully in the database after the deletion of a challenge. |  |  |  |
| Student who submits a deleted challenge will be prompted with an appropriate error. | Submit a challenge as a student after it is deleted, an appropriate error should be displayed. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| Teacher can create a test case and add it to a challenge | A test case can successfully be created |  |  |  |
|  | An invalid test case will prompt an error |  |  |  |
|  | A test case can be successfully attached to a challenge |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acceptance Criteria | ATDD Test | Date Run | Tester | Result |
| Available challenges should be categorized | Challenges are shown to be categorized by language |  |  |  |
| The list of challenges should be all there | All challenges in database that are toggled should be visible |  |  |  |

<http://csazure.cegep-heritage.qc.ca:8080/F2021-DevProject/Code%20Challenge/_boards/board/t/Code%20Challenge%20Team/Stories>

## White Box

Compared to NUnit (just for C#) or other frameworks, XUnit testing has a more extensible testing framework, although it might require more effort for migrating existing implementations to XUnit framework. Our team will use XUnit for unit testing.

TDD mocking is to create an object that simulates another object (normally by constructing simple dependent classes) and sets expectations for its behavior. Mocking is a testing isolation method that simulates the interaction between components and systems. The goal is to test the interface functionality and the correct behavior. No mocking means it puts all the code into the same class and achieve a pass on all tests. Then, refactor the code by decomposing the large class into constituent classes and methods. Our team will use no mocking to test the code at model layer.

We will use class name add “Test” for class for naming standards for tests. If we will test “Challenge” class, we name it for “ChallengeTest”. And we will use “MethodName\_WhatsBeingTested\_ExpectedResult\_OtherInfo” format as the naming standard for our test cases. If we test “GetTitleLength” method, we will name test method for “GetTitleLength\_Input\_NoMoreThan30\_Test”.

# Process

|  |  |
| --- | --- |
| **New** | Column is used when an item has been created but not started. |
| **Active** | Column is used when an item has been started and is being worked on. |
| **Pending Review** | This column would be used when an item needs to be reviewed by another person. The item would stay assigned to the person that was initially working on it. The reviewer will write “Reviewer: their name” in the task discussion so that everyone knows the task will be reviewed by said person. The reviewer will then edit their comment in the task discussion with notes based off their review and either set the task back to active (needs more work) or resolved (it looks perfect). |
| **Resolved** | Column is used when the review is done, and the reviewer does not believe it needs more work. |
| **Closed** | This column is used once the branch has been merged back unto the master branch. |

# Appendix

## Hosting Judge0 with Docker

1. Install WSL 2 with Ubuntu if not already installed

* Notes
  + Windows server:
    - Follow these steps (**Not tested**): <https://docs.microsoft.com/en-us/windows/wsl/install-on-server>
  + Windows home
    - Follow these steps (**tested**) <https://docs.microsoft.com/en-us/windows/wsl/install>
    - If using windows 10 19041 or higher / windows 11
    - You only need to enter “wsl --install” in PowerShell
    - Else, follow manual installation <https://docs.microsoft.com/en-us/windows/wsl/install-manual>

1. Install Docker Desktop

* Windows server / Windows home:
  + Follow these steps: <https://docs.docker.com/desktop/windows/install/>

1. Host Judge0 with Docker

* Follow these steps: [https://github.com/judge0/judge0/blob/master/CHANGELOG.md#deployment-procedure](https://github.com/judge0/judge0/blob/master/CHANGELOG.md%23deployment-procedure)

1. After this, test with http://<IP ADDRESS OF YOUR SERVER>: 2358/dummy-client.html