# Nursing Skills Tracking System Architecture Design and Process Report

**Document Category: DEV**

**Version: 2**

**Status: Initial Draft**

**Last Updated: 2019-11-12**

# Table of Contents

## Introduction

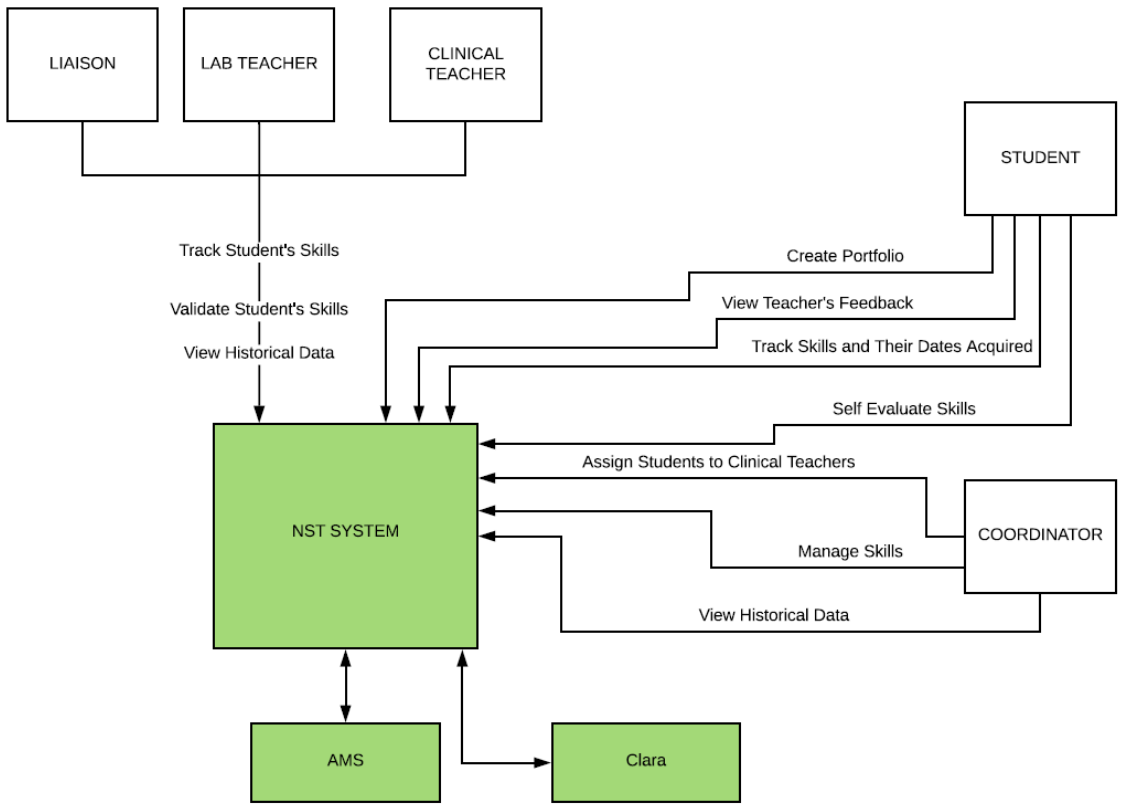
As part of the setup for the Nursing Skill Tracker System (NST), the team has decided on various standards and processes that will facilitate the design and implementation of NST. The team has also made decisions on the first iteration of the architecture, black box views, and white box views including the database model and class diagram.

The purpose of this document is to give clarification to the reader about the general layout and approach for the NST System. In addition, this document will serve as a source of information for future maintenance teams who will work on this system.

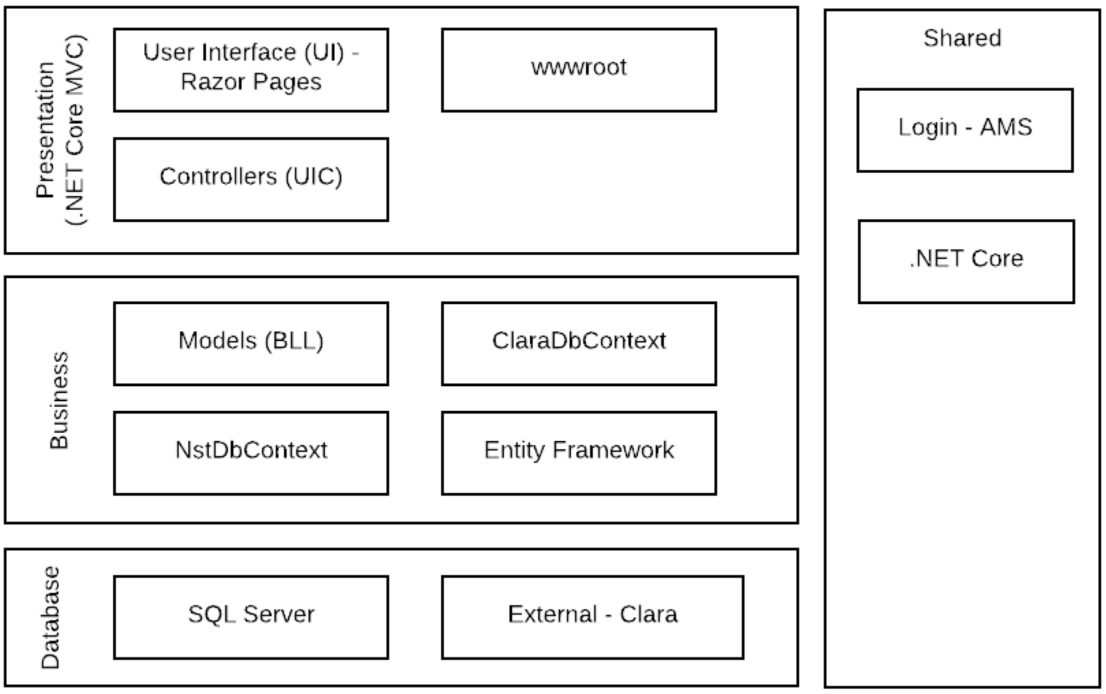
This report will go over the white and black box view, the architecture chosen, and the tools used within the system. Various standards will also be covered including coding standards, testing standards, and documentation organization standards. Finally, recommendations for updating the process in use at Heritage College will be made.

Please note that the team is using the Agile process, which is an iterative process that allows for adaptation of these processes as the project develops. By the final implementation of NST, some of the content may be different. Make sure to refer to the most up to date documentation in the NST folder of the Shared Drive.

## Black Box View



## High Level White Box View



## Software Architecture

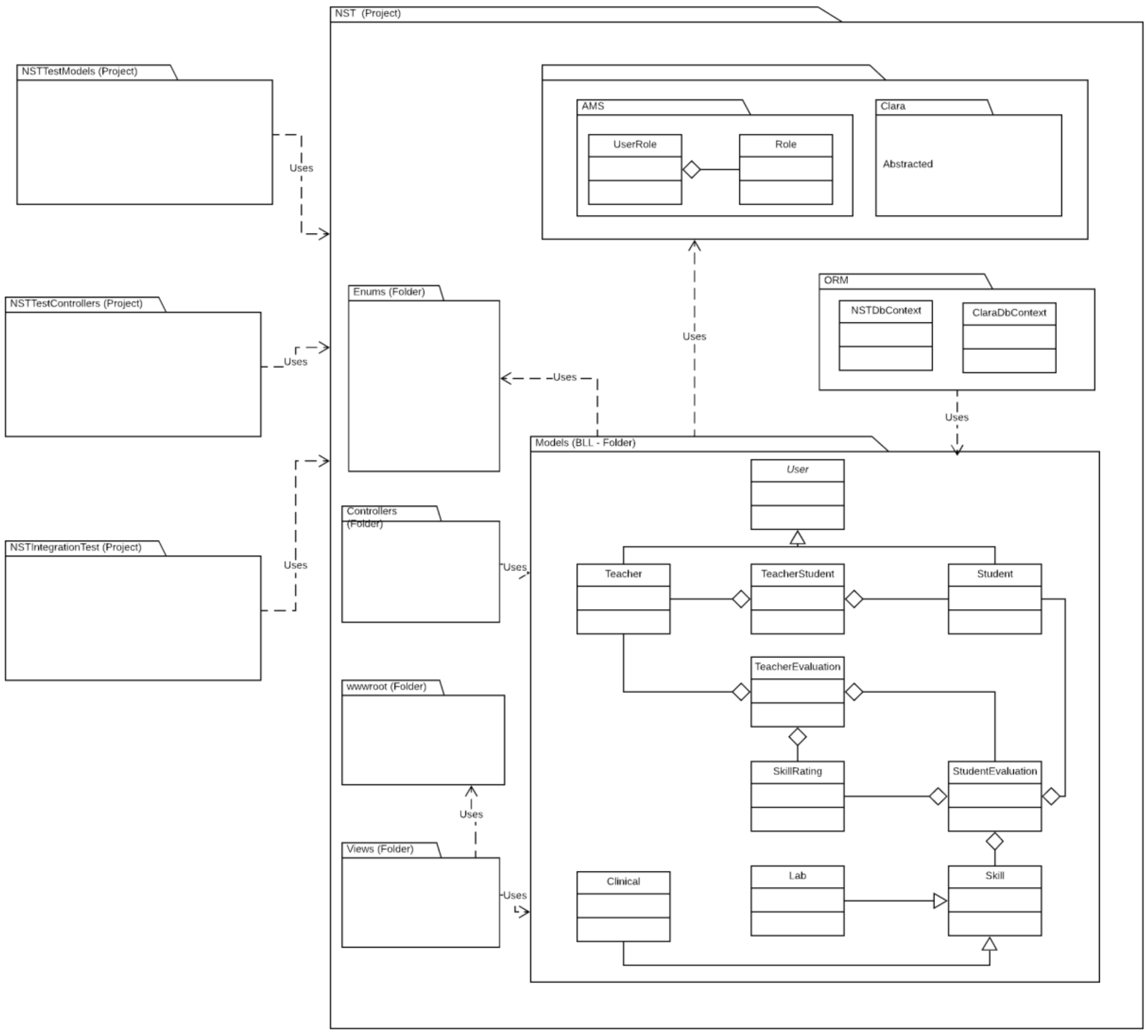
### Considerations and Alternatives

The initial plan was to use the EF Code-First approach for this application. After investigating and analyzing, the team thought about the fact that the Clara database already exists, meaning that the code-first approach will be a potential risk. To avoid this risk, the team decided that a hybrid of EF Code-First and EF Data-First will be used, where Code-First will be used for the NST database and Data-First will be used for the Clara database. This approach has been de-risked.

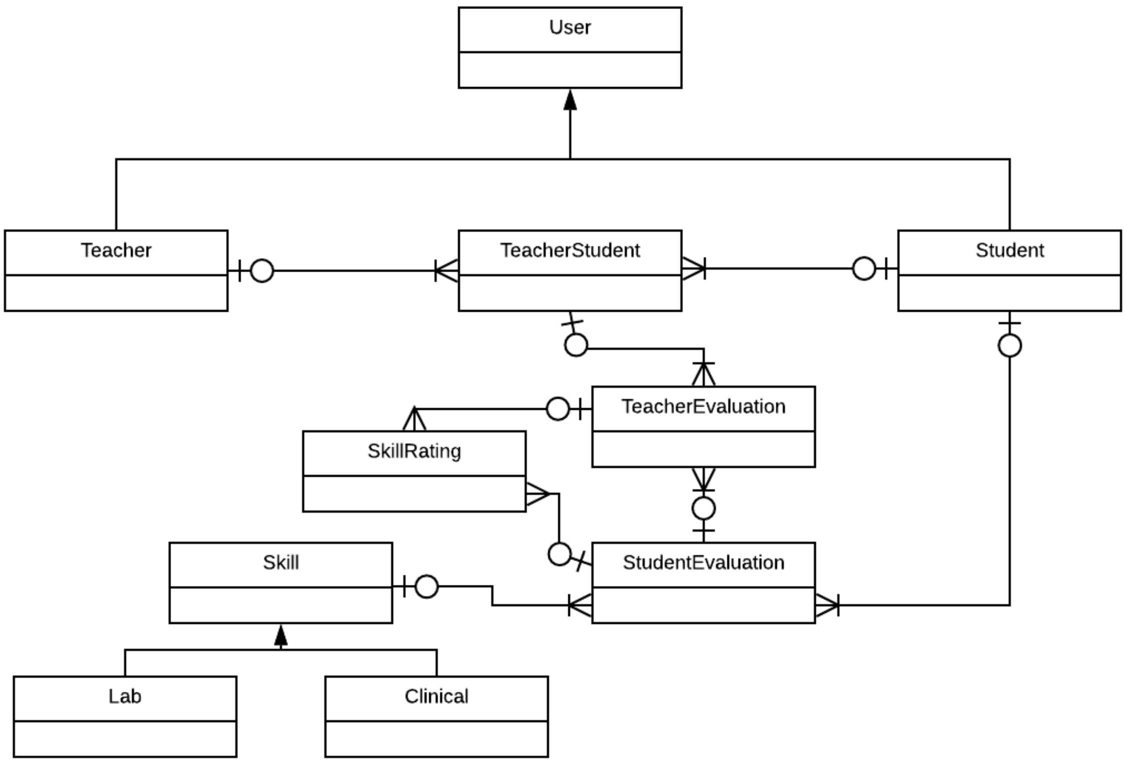
### Architecture Tiers

There will be code on server-side and client-side, but will mostly be on server-side. MVC will be used, which means that the logic will be separated in 3 different layers. The team has thought of keeping the controllers “thin” or “skinny”, meaning that there will not be much logic contained in the controllers, most of the logic will be contained in the models (meaning that there will be “fatter” models). AMS will be used to authenticate and authorize the users. Validation will be done on both ends for security purposes, which will be most of the client-side code. There will perhaps be some animation/view logic on the client-side.

## Object Model



## Data Model



## Framework and Tools

The frameworks used for the NST system consists of MVC on .NET Core with a code first and data first hybrid approach using Entity Framework. These frameworks were chosen after technical research was done, since they meet the initial requirements for the system.

## MVC

Having web forms in consideration, the team decided to use MVC. Web forms features a coupled code file for each page, which prevents proper separation of concerns and reusability of code. In addition, as agile development is generally composed of prototyping and refactoring, the separation of concerns that MVC uses should make the development process faster and clearer.

## .NET Core

Having .NET Framework in consideration, the team decided to use .NET Core in order to take advantage of the small and compact project size, and in order to make this system scalable and fast. Another thing the team took into consideration is how new .NET Core is, as it will give the team good work experience.

## Entity Framework - Code First and Data First

Considering the use of Entity Framework, the team chose to use it to simplify the creation of the application database, and to facilitate changes made during the development process. The team specifically used Entity Framework Code First, as it allows the models to shape the database, and it makes it a lot easier to access the database and make changes to it using migrations. However, since the Clara database already exists, it would be more complex to use code-first with that part of the system, hence the team chose to use a Data-First approach, as it will make the creation of objects easier.

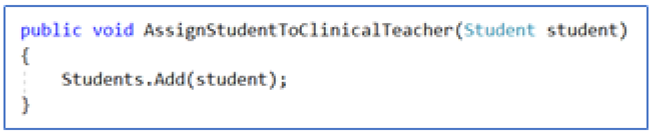
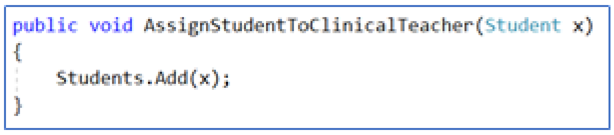
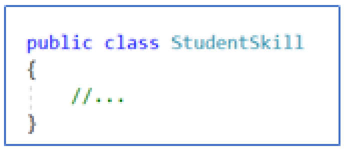
## Presentation Frameworks

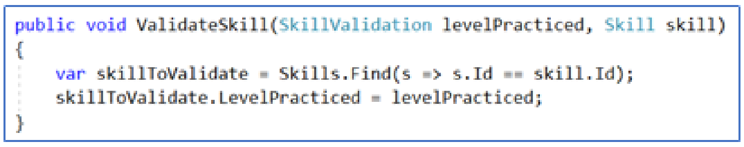
NST will be using the Heritage College Template for almost all styling, but it will also use Bootstrap in order to facilitate mobile responsiveness and to make styling easier. In addition, the team decided to use Razor pages, as they allow to make changes to the HTML depending on the model, allowing for dynamic forms and tables.

# Coding Standards

## C#.NET Core

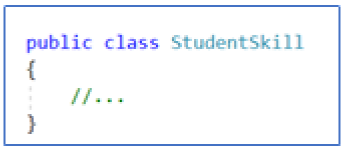
### Naming Conventions

| **Category** | **Rules** | **Example** |
|---|---|---|
| **Identifiers** | - All identifiers will have meaningful names; | **Do** (all identifiers are meaningful)  `public void AssignStudentToClinicalTeacher(Student student) { Students.Add(student); }` **Do Not** (parameter name is not meaningful)  `public void AssignStudentToClinicalTeacher(Student x) { Students.Add(x); }` |
| **Class** | - PascalCase; - Noun or noun phrases; |  `public class StudentSkill { //... }` |
| **Class Properties, Enums, Enum Values** | - PascalCase; | Please refer to class example for PascalCase example |

| | | |
|---|---|---|
| **Local Variables** | - CamelCase;<br>- Implicit typing when the type is clear; | `public void ValidateSkill(SkillValidation levelPracticed, Skill skill)`<br>`{`<br>`    var skillToValidate = Skills.Find(s => s.Id == skill.Id);`<br>`    skillToValidate.LevelPracticed = levelPracticed;`<br>`}`<br><br>For clarification on implicit typing please refer to: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions#implicitly-typed-local-variables |
| **Parameters** | - CamelCase | Please refer to Local Variables example |

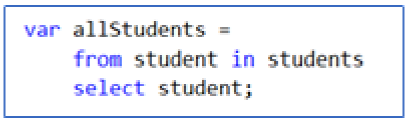
For further clarification and constructs not specified please refer to https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines.

## File Conventions

| | **Rule** | **Example** |
|---|---|---|
| Class | - There can only be one class per class file. | `public class StudentSkill`<br>`{`<br>`    //...`<br>`}` |

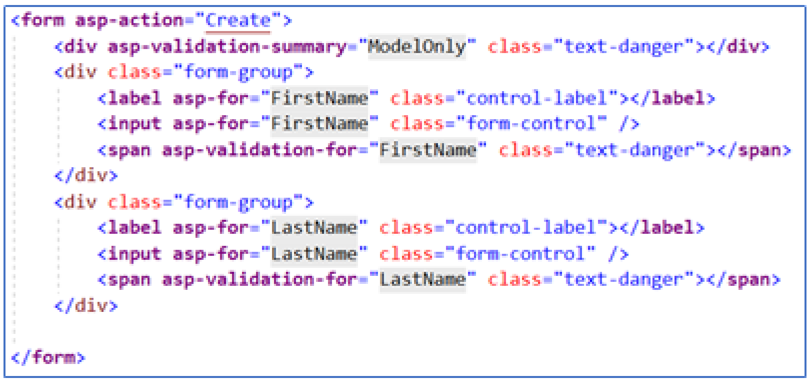
## LINQ Conventions

| | **Rule** | **Example** |
|---|---|---|

| | | |
|---|---|---|
| LINQ Query | - All LINQ queries will use the query syntax; |  `var allStudents =`<br>`    from student in students`<br>`    select student;` |

For further clarification please refer to https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions#linq-queries

## View Conventions

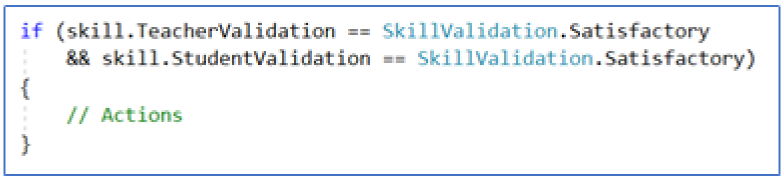
| | Rule | Example |
|---|---|---|
| Razor Views | - ASP HTML style tags will be used, NOT Html Helpers; |  `<form asp-action="Create">` `<div asp-validation-summary="ModelOnly" class="text-danger"></div>` `<div class="form-group">` `<label asp-for="FirstName" class="control-label"></label>` `<input asp-for="FirstName" class="form-control" />` `<span asp-validation-for="FirstName" class="text-danger"></span>` `</div>` `<div class="form-group">` `<label asp-for="LastName" class="control-label"></label>` `<input asp-for="LastName" class="form-control" />` `<span asp-validation-for="LastName" class="text-danger"></span>` `</div>` `</form>` |

```
<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="FirstName" class="control-label"></label>
        <input asp-for="FirstName" class="form-control" />
        <span asp-validation-for="FirstName" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="LastName" class="control-label"></label>
        <input asp-for="LastName" class="form-control" />
        <span asp-validation-for="LastName" class="text-danger"></span>
    </div>

</form>
```

## Formatting

For all formatting the default Visual Studio 2015 IDE settings will be used.

| | Rule | Example |
|---|---|---|
| Indentation | - Four spaces will be used in indentation; | Refer to above examples. |
| Parenthesis | - Always use parenthesis for clauses in expressions<br><br>- All braces will be on their own lines at the beginning and end of a code block |  ```if (skill.TeacherValidation == SkillValidation.Satisfactory && skill.StudentValidation == SkillValidation.Satisfactory) { // Actions }``` |

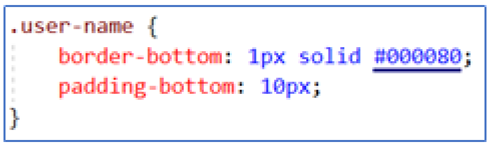
```
if (skill.TeacherValidation == SkillValidation.Satisfactory
    && skill.StudentValidation == SkillValidation.Satisfactory)
{
    // Actions
}
```

For further clarification please refer to: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions#layout-conventions.

For all clarification concerning .NET conventions please refer to: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions and referenced documentation therein.

## CSS

### Naming Conventions

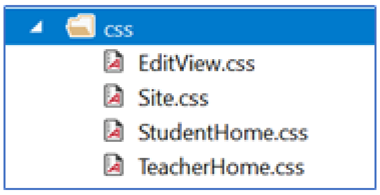
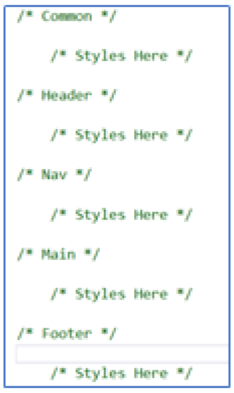
| Category | Rules | Example |
|---|---|---|
| **Identifiers, Classes, Ids** | - All identifiers will have meaningful names;<br>- Kebab-case | <br>`.user-name {`<br>`    border-bottom: 1px solid #000080;`<br>`    padding-bottom: 10px;`<br>`}` |

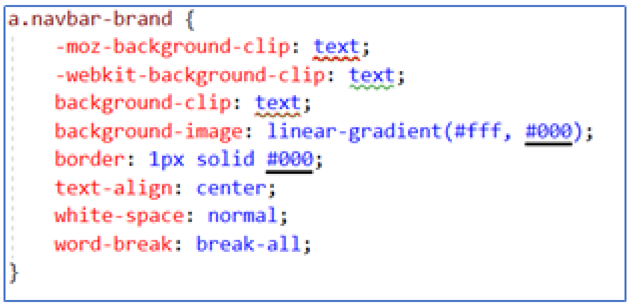
### Formatting

For all formatting the default Visual Studio 2015 IDE settings will be used.

| | Rule | Example |
|---|---|---|
| Indentation | - Four spaces will be used in indentation; | Refer to above examples. |
| Parenthesis | - Opening braces will be CSS selector<br><br>- Closing braces on own line, no indentation | Refer to above examples. |

## Folder Structure and File Layout

| | **Rule** | **Example** |
|---|---|---|
| Grouping CSS in Folder | - View specific CSS should be in its own file<br>- All common CSS will be in Site.css | <br>css<br>EditView.css<br>Site.css<br>StudentHome.css<br>TeacherHome.css |
| Grouping CSS in File | - Group CSS by the flow of the page, wherein the common CSS will be at the top of the CSS file;<br>- Label each group with a comment stating the outermost container for easy navigation; | <br>`/* Common */`<br>`    /* Styles Here */`<br>`/* Header */`<br>`    /* Styles Here */`<br>`/* Nav */`<br>`    /* Styles Here */`<br>`/* Main */`<br>`    /* Styles Here */`<br>`/* Footer */`<br>`    /* Styles Here */` |

| | | |
|---|---|---|
| Property Order | - All properties of a style should be in alphabetical order;<br>- Ignore browser specific prefixes for alphabetical order. |  |

```
a.navbar-brand {
    -moz-background-clip: text;
    -webkit-background-clip: text;
    background-clip: text;
    background-image: linear-gradient(#fff, #000);
    border: 1px solid #000;
    text-align: center;
    white-space: normal;
    word-break: break-all;
}
```

For all further clarification please refer to: https://google.github.io/styleguide/htmlcssguide.html#CSS. Please note Google uses indentation of two spaces, NST will use four spaces as noted previously.

## Document Organization Standards

For the purposes of maintaining proper document organization, all main copies of communication, diagrams, notes, code documentation, schedules, standards and others will be stored within the Heritage College Student Shared Drive. This will ensure that the team and project manager have access to all up-to-date documents required for the project. The Student Shared Drive will be subdivided into multiple folders. Each type of file and folder will have their own naming standard following the structure below:

- S:\Computer Sciences\Projects\Development\2019\NST
    - Emails – contains all documents received or sent to client of importance.
        - yyyy-mm – subfolder used to separate emails by year and month
            - yyyymmdd_personLastName_Title - Format
    - Diagrams –Contains all diagrams produced for the project
        - Current – Contains all up-to-date diagrams of the project
            - NST_DiagramType_VersionNumber - Format
        - Previous – Contains all diagrams that are no longer being used.
            - NST_DiagramType_VersionNumber - Format
    - Charts – Contains all charts produced for the project
        - NST_Title_ChartType - Format
    - Reports – Contains all reports produced for the project, including unit test reports
        - NST_ yyyymmdd _Title - Format
    - Meeting Notes – Contains all documents gathered during a meeting.
        - NST_ yyyymmdd _MeetingTitle_NoteType – Format

- Client Documents – Contains all documents provided from the client
  - NST_DocumentTitle_ClientFirstLetterFirstNameLastName_VersionNumber
- Prototypes – Contains all prototypes created or used during the project
  - NST_DocumentTitle_PROTOTYPE

## Test Standards

### Black Box

The development team will be using acceptance criteria with the help of TFS. Based off the user requirements provided by the product owner, tasks will be made and stored on TFS. The source control system allows for easy access to the tasks by the development team and project manager. Acceptance Criteria will be written for every task using the user requirements, which clearly define the performance of a task.

Test plans will be created using the TFS feature based on the Acceptance Criteria and expected flow.

Note: Please refer to the acceptance criteria of the tasks in sprint 1 in TFS where these tables are present in the associated tasks.

| Acceptance Criteria | ATDD Test | Date Run | Tester | Result | Link to TFS Task |
|---|---|---|---|---|---|
| User can login | User Id and Password is authenticated | | | | https://cstfs.cegep-heritage.qc.ca:8080/tfs/NST/NST/_workitems?id=1&_a=edit |
| | Incorrect User Id or Password will not be authenticated | | | | |

| | | | | | |
|---|---|---|---|---|---|
| | Students are correctly authorized as students, not teacher | | | | |
| **Acceptance Criteria** | **ATDD Test** | **Date Run** | **Tester** | **Result** | **Link to TFS Task** |
| Student can self-evaluate their skills | Students view a list of their skills | | | | [https://cstfs.cegep-heritage.qc.ca:8080/tfs/NST/NST/_workitems?id=11&_a=edit](https://cstfs.cegep-heritage.qc.ca:8080/tfs/NST/NST/_workitems?id=11&_a=edit) |
| | Student skill is evaluated | | | | |
| | Student comments on a skill | | | | |
| | Evaluation is saved and viewed after it is completed | | | | |

| **Acceptance Criteria** | **ATDD Test** | **Date Run** | **Tester** | **Result** | **Link to TFS Task** |
|---|---|---|---|---|---|
| Teacher can view all their students | Teacher views all their students when logged in | | | | https://cstfs.cegep-heritage.qc.ca:8080/tfs/NST/NST/_workitems?id=2&_a=edit |
| | Teacher searches for a student by name when logged in | | | | |
| | Teacher searches for a student by studentId when logged in | | | | |
| **Acceptance Criteria** | **ATDD Test** | **Date Run** | **Tester** | **Result** | **Link to TFS Task** |
| Teacher can view all skills for a student | Teacher sees all skills for a student when logged in | | | | https://cstfs.cegep-heritage.qc.ca:8080/tfs/NST/NST/_workitems?id=3&_a=edit |

| | | | | | |
|---|---|---|---|---|---|
| | Teacher views student profiles when logged | | | | |
| | Teachers sees evaluations completed by students for their skill | | | | |

## White Box

The system will be using unit testing in order to improve the quality and performance of the code. The team chose to implement unit tests using XUnit, as the development team has prior knowledge on how to use it. Unit tests will stored in their own project to allow logical grouping of testing.

All code that is going to be tested by unit tests should be kept on the model layer, allowing for easy testing of the logic. Dependency injection should be used in order to facilitate class testing. Code in the data access layer (model) should be tested using Moq, if necessary, in order to test database access. In addition, the flow of the system should be tested every sprint using user interface testing, and regression testing. This allows to test code located on the controller and the view layer, and it might show other needed requirements for the system.

Testing classes should use PascalCase and have a name that represents what is being tested. For example, a testing class testing the class “Dog” should be named “DogTests”. Testing methods should be named using the “MethodName_StateUnderTest_ExpectedBehavior” format. For example: “ChangeName_InvalidFirstNameField_ThrowsArgumentException”.

All tests should be kept on the Team Foundation Server for NST. In the Test module of TFS, there should be one test plan for each sprint. Inside each test plan, a static suite named after a user story holds all tests related to the tasks in that users story. Each test or suite should be linked directly to the user story that is being tested. The way to link them is by opening the user story and adding a direct link by imputing the suite or test ID.

## Process

This project will be using the agile methodology, as such, the TFS process will be Agile.
The Agile process provides the team with the ability to associate tasks to user stories. The user stories are completed when all tasks and bugs are finished for them. This process allows the team to:

- Create new epics, features and user stories.
- Create tasks and bugs.
- To set new, active, resolved, closed and removed states to work items.

Unfortunately, tasks do not have the resolved state. It would be useful to have a resolved stage before closing, wherein review of the completed task would occur. Furthermore, the team should be able to create tasks without associating them to a user story and link smaller tasks to that task. These are the recommendations to improve the process.