CODE INSPECTION

CODE INSPECTION FORMAT

The Code Inspection will be conducted on the entire source code. There are three separate programming languages used in the source code, and we will be examining the source code, organized in this manner. The three languages are: HTML + CSS, JavaScript, and PHP.

For each of these languages we will be answering these questions (wherever applicable):

Readability and Maintainability

- Can I understand the code reading it?
- Is the documentation good enough?
- How is the coding style?
- Do we follow appropriate naming conventions?
- Do we have bad copy and paste of code?
- Do we use constants whenever it is possible?

Design

- Does the code corresponds to our original design?
- Can we better structure the code using design patterns?

Functionality

- Can I understand what the code does?
- Does the code do what it is supposed to do?
- Is there some dead code or unnecessary debug statements?
- Do we handle the exceptions and errors correctly?

Testing

- Can I understand what the tests do?
- Do I understand the tests results?
- Do we have a good test coverage?

CODE INSPECTION

The following is a compiled Code Inspection report from all of our team members.

CODE INSPECTION – HTML + CSS

- Can I understand the code reading it?
  - Yes. HTML code are by nature already easy to read. We have used the conventions for HTML coding (Dividing the Head and Body sections in order), and using DIVS where appropriate. We have used best practices for naming conventions, and tagged each DIV (as well as other elements) with some class, or id name, so that any developer from our team can easily distinguish what each element corresponds to.

softserv/index.html

```
1  <!DOCTYPE html>
2  <html ng-app="navApp">
3
4      <head>
5          <script src="js/jquery.js"></script>
6
7          <link rel="stylesheet"
           href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.cs
           s">
8          <script
           src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
           </script>
9          <!-- load angular and angular route via CDN -->
10         <script
           src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"
           ></script>
11         <script
           src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular-
           route.js"></script>
12
13         <meta charset="UTF-8">
14         <!--Styles-->
15         <link rel="stylesheet" href="master.css" type="text/css">
16         <!--Fonts-->
17         <link href="https://fonts.googleapis.com/css?family=Raleway:400,700"
           rel="stylesheet">
18         <!--Stuff-->
19
20         <script src="js/script.js"></script>
21         <title>SoftServ</title>
22     </head>
23
24     <body ng-controller="mainController">
25         <div class="container">
26             <div ng-view>
27             </div>
28         </div>
29
30     </body>
31 </html>
```

  - We have used Angular, which reduced our use of JavaScript. Our use of the ng-directives are easily distinguishable.

CODE INSPECTION

prof-problemsets.html

```
24  </div>
25  <div class="container">
26      <span class="med dark-grey b">
27      Problem Sets
28      </span>
29      <div class="container">
30          <div ng-repeat="(unitid, unitdat) in unitproblemsets">
31              <table class="table table-bordered">
32                  <thead>
33                      <span class="med dark-grey b">
34                          {{unitdat.unitname}}
35                      </span>
```

- The CSS code is easy to read also by nature, we can easily see what properties are being applied to which classes. We have used basic conventions for naming, and have used inheritance wherever possible.

- Is the documentation good enough?
  - Somewhat. The HTML and CSS code is easy to read on it's own, so we have not included much documentation (HTML and CSS comments). We have structured the code in such an order that it will be easy to read.
  - However, we have included some documentation for our CSS code, to organize the types of classes (default: body, li, span, classes, id)

- How is the coding style?
  - We have used the conventions for HTML and CSS coding.
  - The HTML code is properly indented.
  - The CSS code is also properly indented.

- Do we follow appropriate naming conventions?
  - We have used Bootstrap for most of our HTML code, and only used our own custom CSS code to further customize the pre-sets by Bootstrap.

```
24  a {
25      text-decoration: none;
26      color: inherit;
27  }
28
29  input {
30      width: 100%;
31  }
32  .b {
33      font-weight: bold;
34  }
35  .big {
36      font-size: 30pt;
37  }
38  .med {
39      font-size: 20pt;
40  }
```

```
41  .sma {
42      font-size: 12pt;
43  }
44  .dark-grey {
45      color: #655b5b;
46  }
47  .light-grey {
48      color: #b9adad;
49  }
50  .button-large {
51      margin: 10px 10% 10px 10%;
52      width: 80%;
53      height: 60px;
54      border-radius: 5px;
55  }
56  .question-field {
57      height: 3
58  }
59
```

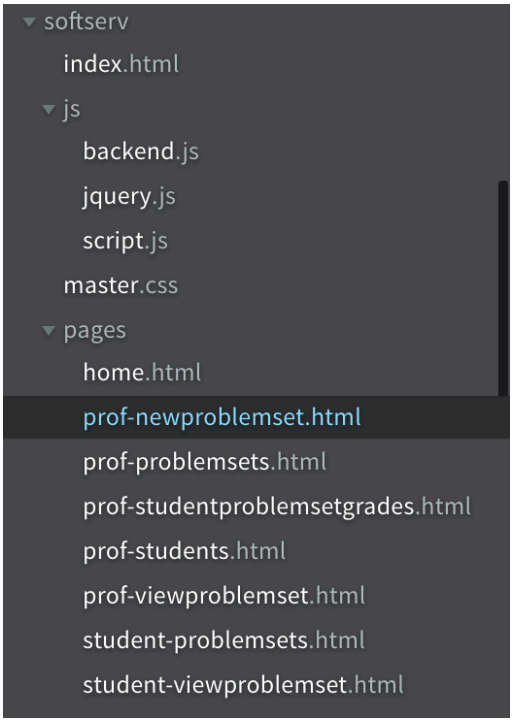
CODE INSPECTION

- For the fonts, we have used easy to read class names, which accurately describe the style that they apply. (words within names are separated by a dash, and abbreviated words are used whenever possible).

- Do we have bad copy and paste of code?
  - No, if there are instances of copy and pasting of code, it is because it is necessary.
  - We utilized a navigation bar that is the same for every professor on every page yet rewrote it every time we coded a new page. This meant we needed to update every page when we added a new page. This could be improved by using an html header file. We could simply abstract the navigation bar to a separate file that is linked in each html file that needs it. Thereby reducing the amount of code, we need to update upon creating and deleting pages.

```
6       </div>
7       <ul class="nav navbar-nav">
8         <li><a href="../softserv/#!prof-students">Students</a></li>
9         <li><a href="../softserv/#!prof-problemsets">Problem Sets</a></li>
10        <li class="active"><a href="../softserv/#!prof-newproblemset">New Problem Set</a></li>
11      </ul>
12    </div>
```

- Do we use constants whenever it is possible?
  - No, but there is no need to use constants for HTML and CSS code.



Design

- Does the code corresponds to our original design?
  - Yes, we have originally decided to have only one 'master' CSS file which will be used for all our HTML files.
  - For our HTML code, we decided that we would have one main index.html file which is placed in root folder of the source code. The different HTML files are loaded as views by Angular, into some frame that is defined in the main index.html file.
  - For most of the data stored in the database that we need to display, we have decided that we

are going display the information as they are in the database (with some modifications if we need to use SQL joins).
- For most of the functionality where we need to insert some data, we use simple forms with easy-to-read buttons, which was our original design.

Functionality

- Does the code do what it is supposed to do?
  - Yes, the CSS styles are applied properly and are working as expected.
  - The ng directives are also working. Where we have ng-repeat for cases where we want to repeat some row-data that was sent by the PHP web service, and all the data is being displayed as expected.
  - We also have ng-model statements where we want to bind the input value of a text field to a variable in the JavaScript files. The variables are bound properly, as per our testing.
  - We also have ng-click statements where we want to call a specific function when a button is clicked, and our tests have shown that those corresponding functions are being executed for the buttons that they are bound to.
  - For our navigation, the menu options are redirecting to the correct pages (as described by the menu option), as per our tests.

- Is there some dead code or unnecessary debug statements?
  - There are no unnecessary debug statements in our HTML and CSS code (as those are not really supported).

- Do we handle the exceptions and errors correctly?
  - We don't have much exceptions and error handling. When there should be errors displayed due to an input error (by the user), there is no information rendered in the HTML that tells the user of the errors.

CODE INSPECTION – JavaScript

- Can I understand the code reading it?
  - No, the code is difficult to understand if the developer that is examining the code doesn't know AngularJS.
  - We have structured the code properly and organized it in such a manner that each section is separated by the user interface that that section controls. Each section is called a 'controller.' In that sense, there is a default structure in our

CODE INSPECTION

JavaScript, so that when a developer looks at the code, they can easily identify which sections correspond to which user interface.

- The functions are defined to minimize code repetition, and the code within them is just basic loops which are easy to read.

- Is the documentation good enough?
  - There is marginal documentation for our JavaScript files. There are only headers for each controller, to state which user interface the controller corresponds to.
  - There is no internal documentation.
  - There are also no documentation that describes what each function is doing. Therefore, a developer may have a hard time distinguishing what exactly each function does, internally.

```
118  // ****************************************
119  // ****************************************
120  // PROFESSOR ADDING STUDENTS
121  // ****************************************
122  // ****************************************
123  navApp.controller('prof-studentsController', function($scope, $http,accountService) {
124      // create a message to display in our view
125      $scope.getstudents = function() {
126          $http.get("php/getstudents.php").then(function(data) {
127              console.log("getting students");
128              console.log("whats going on", data);
129              $scope.students = data.data;
130              console.log($scope.students);
131              //$scope.$apply();
132          });
133      }
134
```

- How is the coding style?
  - The code is poorly indented (which can be easily fixed by a JavaScript beautifier)
  - When we are calling the PHP web services to get the data that we need to render into the HTML, we wrap the parameters and header details in a configuration variable, which is then passed into the web request when it is called. We then receive the data from the web service and assign it to our own variable, as per best practices.
  - When we are rendering the HTML information to the user interface, we try to use ng-directives made available by Angular. Most of the rendering is thus done in the front-end. (ng-repeat, for rendering table data). This is to make the code more readable, because rendering the data via JQuery, by appending, is difficult to maintain and read for the developer.
  - There are instances where we do use JQuery to append some HTML data into the user interface, but this is done only for instances where it is necessary. For

CODE INSPECTION

example, we use this for adding question and answer fields into the user interface. We do this because there are no ng-directives that assist us with this problem

```
193     */
194     $scope.genQuestionField = function(num) {
195         var qLabel = "<label for='question" + num + "'>Question " + num + "</label>";
196         var qTextArea = "<textarea class='form-control' rows='5' ng-model='(questions[" + num +
            "]).question'></textarea>";
197         var qFormGroup = "<div class='form-group' id='question" + num + "'>" + qLabel + qTextArea + "</div>";
198
199         var aLabel = "<label for='answer" + num + "'>Answer " + num + "</label>";
200         var aTextArea = "<input type='text' class='form-control' ng-model='(questions[" + num + "]).answer'>
            </input>";
201         var aFormGroup = "<div class='form-group' id='answer" + num + "'>" + aLabel + aTextArea + "</div>";
202
203         var deleteButton = "<button type='button' onclick='deletequestion(" + num + ")'>Delete</button>";
204
205         var string = qFormGroup + aFormGroup;
206
207         return string;
208     }
```

- Every snippet of code is within some function, and there are no global variables in the code, and thus we don't have any closures (which is good).
- There are functions that are exceeding the acceptable standards, lengthwise. An example of such a function is "checkanswers()" under "student-viewproblemsetController".

```
428     // needs to record the results somewhere
429     $scope.checkanswers = function() {
430         var counter = 0;
431         $.each($scope.problemset, function(qid, data) {
432             counter += 1;
433             if (counter == 2) {
434                 //Counting the grades
435                 var countCorrectAnswer = 0;
436                 var str = "";
437                 for (var i = 0; i < $scope.stuAnswer.length; i++) {
438                     var result = "incorrect";
439                     console.log("Comparing answers");
440                     if ($scope.stuAnswer[i] == data.answer) {
441                         result = "correct";
442                         countCorrectAnswer+=1;
443                     }
444                     var answerStr = "<tr><td>" + $scope.stuAnswer[i] + "</td><td>" + data.answer + "</td>
                        <td>" + result + "</td></tr>";
445                     str += answerStr;
446                 }
447                 $scope.mark = countCorrectAnswer/$scope.stuAnswer.length;
448                 $scope.show_mark = true;
449
450                 $("#display-answers").empty();
451                 $("#display-answers").append(str);
452             }
453         });
454
455         //Setting up data to send to ssend
456         //$scope.user for utorid
457         //$scope.problemsetid for problemsetid
458         $scope.updatemark = function() {
459             var config = {
460                 params: {
461                     username: $scope.user,
462                     problemsetid: $scope.problemsetid,
463                     mark: $scope.mark
464                 },
465                 headers: {
466                     'Accept': 'application/json'
467                 }
468             }
469             console.log("config", config);
470             $http.get("php/storegrades.php", config).then(function(data) {
471                 console.log("storing the grades", data);
472                 //$scope.$apply();
473             });
474         }
475         $scope.updatemark();
476
477     }
```

CODE INSPECTION

- Do we follow appropriate naming conventions?
  - We use camel case notation for our naming of JavaScript variables, as well as function names.
  - We try to be as descriptive, but also concise as possible when naming, so that the variables are easy to read but also convenient to use.

- Do we have bad copy and paste of code?
  - There are some lines of code that are essentially duplicated data, which can be fixed by enclosing some code into a function and calling that function.
  - However, most of the code is non-repetitive. Albeit, it does appear to have that property, but it is necessary (example: HTTP get requests).

- Do we use constants whenever it is possible?
  - Yes, but there are no need for constants in our code.

Design

- Does the code corresponds to our original design?
  - Yes, we originally designed the JavaScript (front-end) component of our product to be divided into separate controllers for each user interface.
  - Most of the JavaScript code follows some structure. For functions where we have to render some data from the back-end into the HTML, we use a configuration variable to prepare some parameters and headers (if they are necessary) and then call the built-in Angular function that sends a request to PHP (usually a GET request), and then receives data back as a JSON object. This is as per our original design, which was to get some data from the web server where needed by using a GET request.
  - We have some JavaScript functions that render some HTML data into the UI. We use this format for rendering where we have originally decided to use this format (example: rendering additional question and answer fields into the HTML whenever the user clicks 'Add Question', for creating a problem set).
  - We also originally designed our navigation system to update the loaded view (an html page) whenever the URL is changed. This is all controlled in the router function, which was chosen by our team because it is easy to load the new view, and bind another controller to that view.

CODE INSPECTION

```
// configure our routes
navApp.config(function($routeProvider) {
    $routeProvider.when('/', {
            templateUrl: 'pages/home.html',
            controller: 'mainController'
        })
        .when('/prof-students', {
            templateUrl: 'pages/prof-students.html',
            controller: 'prof-studentsController'
        })
        .when('/prof-newproblemset', {
            templateUrl: 'pages/prof-newproblemset.html',
            controller: 'prof-newproblemsetController'
        })
        .when('/prof-problemsets', {
            templateUrl: 'pages/prof-problemsets.html',
            controller: 'prof-problemsetsController'
        })
        .when('/prof-grades-problemsets', {
            templateUrl: 'pages/prof-grades-problemsets.html',
            controller: 'prof-grades-problemsetsController'
        })
```

- There are also some additional services that we use for data transferring. We have two types: dataService to transfer problem set ids, and accountService to transfer account data for students. Our original design was to have the use of services that can be accessed by any of our controllers (that needed that service), to ensure consistency and proper data transfer across pages. This has been implemented.

```
//PASSING ACCOUNT SERVICE
navApp.service('accountService', function() {
    var data = {
        utorid: ''
    };
    return {
        setData: function(newObj) {
            data.utorid = newObj;
        },
        getData: function() {
            return data.utorid;
        }
    };

});
```

```
//PASSING DATA SERVICE
navApp.service('dataService', function() {
    var data = {
        problemsetid: '0'
    };
    return {
        setData: function(newObj) {
            data.problemsetid = newObj;
        },
        getData: function() {
            return data.problemsetid;
        }
    };

});
```

- Can we better structure the code using design patterns?
  - At this moment, there are no particular design patterns that can better structure our code.
  - Simply, the code should be separated into separate files, such that, for example, there is one JavaScript file for each controller.

CODE INSPECTION

Functionality

- Does the code do what it is supposed to do?
  - The navigation router function is working as expected. The proper html file is being loaded along with the controller as defined in the router function.
  - The HTTP get requests are working properly, the configuration variables are being prepared properly and the request is going through without errors. The data is also being received in the correct format in the front-end.
  - The functions that correspond to rendering the HTML are all working properly. We have checked that they are working by looking at the user interface. The tables are being loaded with all the correct data, and text-fields are being loaded dynamically as expected.
  - The buttons are also properly bound to the respective functions. Clicking the respective buttons call the functions that they were expected to call.
  - The scope variables are being bound properly. When input values in the user interface (text-fields) are changed, the changes are also being reflected in the variable in JavaScript.
  - The functions that perform calculations, for example, the function for checking answers, is working as expected. It is counting the correct number of answers and is calculating the percentage grade properly.

- Is there some dead code or unnecessary debug statements?
  - There are no plenty of debug statements used throughout the JavaScript code. We have console.log() statements everywhere, which should be deleted after testing is finished.
- Do we handle the exceptions and errors correctly?
  - We don't have much exceptions and error handling. When there should be errors displayed due to an input error (by the user), there is no information rendered in the HTML that tells the user of the errors. There is no catching statements in the JavaScript, that send data into the user interface when something goes wrong.

Testing

- Can I understand what the tests do?
  - We have only used acceptance tests to test our front-end JavaScript code. The tests are easy to read because we have procedures and expected results for each action that should be executed by the tester.

CODE INSPECTION

- Do I understand the tests results?
  - Yes, we have included what the results should be. We have also included images of the expected screen so that the tester can compare their resulting screen.

- Do we have a good test coverage?
  - We have tested per feature, because we believe that each individual feature should be tested before it is released.

CODE INSPECTION – PHP

- Can I understand the code reading it?
  - No, the code is difficult to understand if the developer that is examining the code doesn't know PHP.
  - We have structured the code properly and organized it in such a manner that the first section corresponds to setting up configuration with the database, then there we prepare our SQL query statements, then we execute the queries and retrieve the result, then we prepare and return the JSON object representation of our query data.
  - The PHP scripts are easy to follow as they follow a sequential order for execution.

```php
<?php
header('Content-Type: application/json');
include('./config.php');

$studentid = $_GET["username"];
// create a connection
$conn = mysqli_connect($servername, $username, $password, $db);
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql_getgrades= "SELECT PROBLEMSETS.ID AS ID, UNITID, UNITS.NAME AS UNITNAME,
PROBLEMSETS.NAME AS NAME, DATEDUE, HighestScore, RecentScore FROM (PROBLEMSETGRADES
RIGHT JOIN PROBLEMSETS ON ID = ProblemsetID AND StudentID = '$studentid') JOIN UNITS ON
UNITS.ID = PROBLEMSETS.UNITID;";
```

- Is the documentation good enough?
  - There is marginal documentation for our PHP files.
  - There is inadequate internal documentation. There are some comments but they seem to be irrelevant to the code.
  - There are also no documentation that describes what each function is doing. Therefore, a developer may have a hard time distinguishing what exactly each function does, internally.

CODE INSPECTION

- How is the coding style?
  - The code is poorly indented (which can be easily fixed by a PHP beautifier)
  - We are setting up the connection every time the product requires some data from the database. This is not efficient, and we should only set up the connection once per user session.
  - There functions are written to be as concise as possible.

- Do we follow appropriate naming conventions?
  - We use consistent notation across all our PHP web services. The names are changed according to the web service that they are providing. However, we have special naming, for example, for SQL statements, row data, and JSON objects that are meant to be returned into JavaScript.
  - We try to be as descriptive, but also concise as possible when naming, so that the variables are easy to read but also convenient to use.

- Do we have bad copy and paste of code?
  - There are some lines of code that are essentially duplicated data across all PHP web services, which can be fixed by enclosing some code into a function and calling that function. An example would be the statements that set up the connection to the database.
  - However, most of the code is non-repetitive. We use include statements, for example, for our configuration details, so that we do not have to repeat this information for each PHP file.

- Do we use constants whenever it is possible?
  - Yes, but there are no need for constants in our code.

Design

- Does the code corresponds to our original design?
  - Yes, we originally designed the PHP (front-end) component of our product to be divided into separate files that execute one type of service, as opposed to one PHP file that hosts every type of web service that we have.
  - Most of the PHP code follows some structure. The first section corresponds to setting up configuration with the database, then there we prepare our SQL query statements, then we execute the queries and retrieve the result, then we prepare and return the JSON object representation of our query data.

CODE INSPECTION

- We have originally decided that we would return the query data as a JSON object with key value pairs, which is what we have implemented.
- We have decided to convert any data that can be easily converted in the backend, so that the front-end code receives data that it doesn't need to further modify. This was designed with the justification that the user should not be able to see our algorithms for changing data.

- Can we better structure the code using design patterns?
  - At this moment, there are no particular design patterns that can better structure our code.

Functionality

- Does the code do what it is supposed to do?
  - The web services are setting up the connection to our external database as expected. The configuration file is therefore correct.
  - The web services are receiving the parameters from the front-end as expected.
  - The queries are also working as expected, and the parameters that filter the data is working. The query returns the correct table data, with appropriate names.
  - The conversion from table data into JSON object is also working correctly. The front-end receives data that it can work with.

- Is there some dead code or unnecessary debug statements?
  - There are no debug statements used throughout the PHP code.
- Do we handle the exceptions and errors correctly?
  - We don't have much exceptions and error handling. When a query is doesn't work, there isn't alternative or useful data that is sent back to the JavaScript front end. There are simply errors that are logged.
  - There is also no error handling when it comes to receiving faulty parameters via GET or POST.

Testing

- Can I understand what the tests do?
  - Yes, each test is named so that it tells the tester which feature or component it is testing.

- Do I understand the tests results?

CODE INSPECTION

- Yes

- Do we have a good test coverage?
  - No, the code is really simple, I don't think there should be anything to test, but we do not have a very good test coverage.