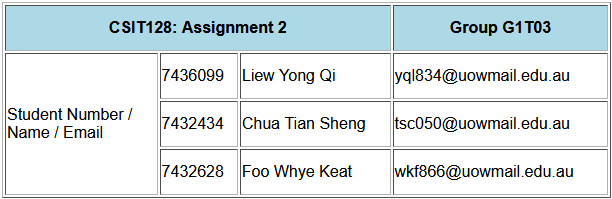
**Part 1:**

****

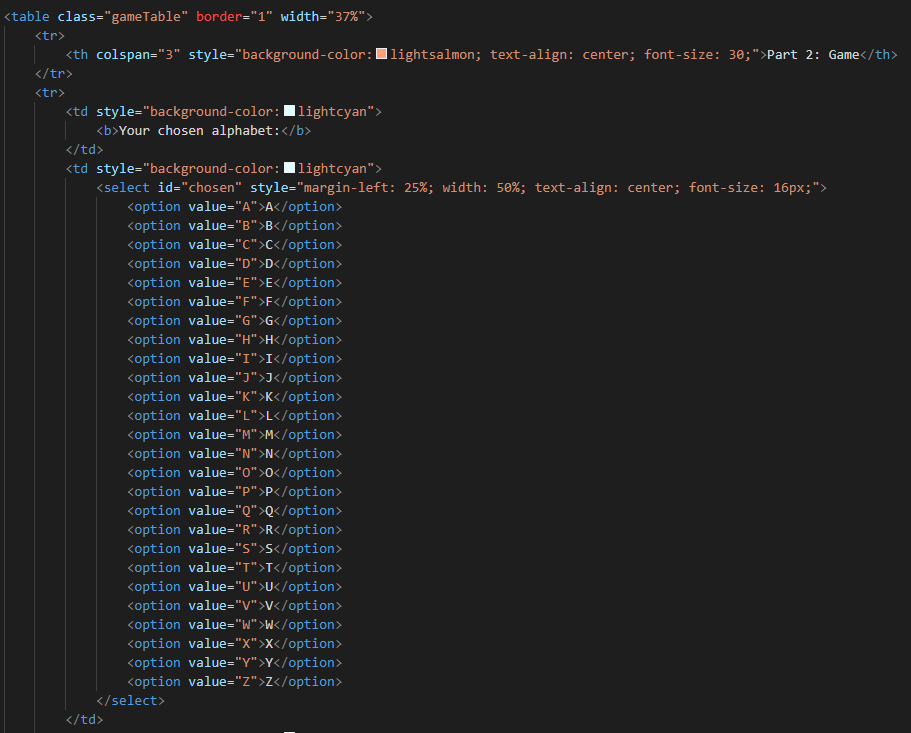
We begin part 1 by building a table with the <table> tag. Border = “1” sets a border around the table cells, we then adjust the size of the cells through the height and width percentages. This allows us to have the table look neater and get rid of any unnecessary white spaces. Next, the font family allows us to give the browser font styles to fall back on. The first choice will be Arial and if Arial is not available to the browser, it will use Helvetica and so on.

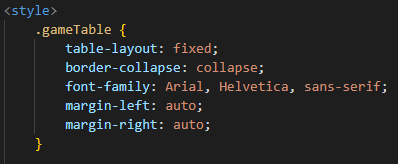
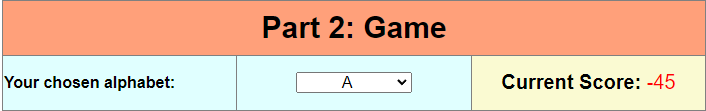
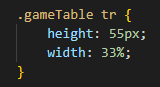
We create our first table row by using the <tr> tag, and in this tag, we change the background colour using inline CSS, so the background colour is light blue. In our first table row, we have 2 sets of information, this is defined by our table headers <th>. For “CSIT128: Assignment 2” in its <th> tag, we assign it a colspan of 3. This means that the words “CSIT128: Assignment 2” would span over 3 columns. The <th> tag automatically bolds the words for us.

After we are done with our first row, we start our second table row with another <tr> tag. We enter data into this row through the table data tag <td>. For the first <td> tag, we assign this data to span 3 rows by using rowspan. This allows the words “Student Number / Name / Email” to occupy 3 rows. We then use more <td> tags to enter the necessary information in the first row.

For the next 2 table rows, since the words “Student Number / Name / Email” are occupying 3 of the rows in column 1, the words are shifted into the second column of the same row thus aligning itself nicely. We then continue adding the information of the remaining 2 members using the same <tr> tag and <td> tags. And with that we have completed our table of personal information.

**Part 2:**

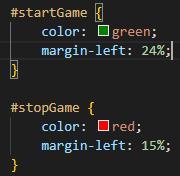
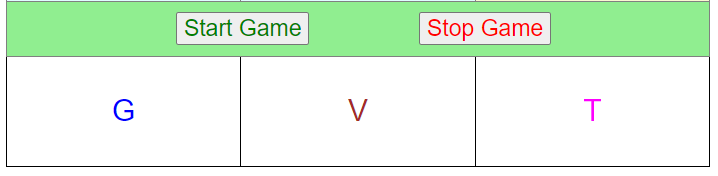
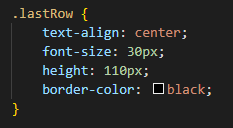
****

** **

We begin part 2 by creating a table, setting a border around the cells, and adjusting the width of the table. We then style the table using CSS in the header making it look as close to the given table as possible. Our table begins with the table head which consists of “Part 2: Game” that occupies 3 column spaces. For the next row of data, we have the cell background colour set to lightcyan and inside the cell are the words “your chosen alphabet”.

The next column in the same row will be a drop-down menu containing all the words in the alphabet, we do this by using a <select> tag and within the <select> tag we have <option> tags. In the <select> tag we have CSS to manipulate the size, shape, and format of the drop-down menu. Since we did not include a “selected” attribute in the <select> tag, the first option ‘A’ will be the default option in the drop-down menu.

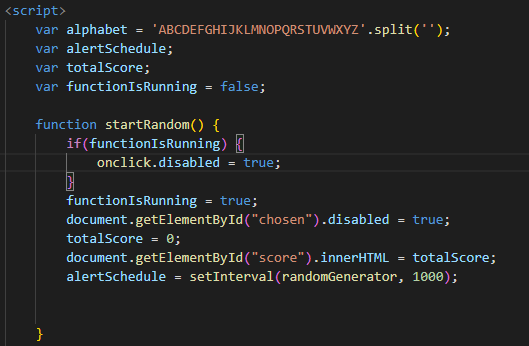


For the third column of the row, we display the current score that the user has. The background colour is set to lightgoldenrodyellow, and the score is defaulted to 0.

The next row includes the buttons, the table data is set to occupy 3 columns with a background colour of light-green. The first button is to Start the game and the second button is to Stop the game, the colour of the words in the button are styled using CSS. Onclick, the start and stop button will call their respective functions startRandom and stopRandom respectively.

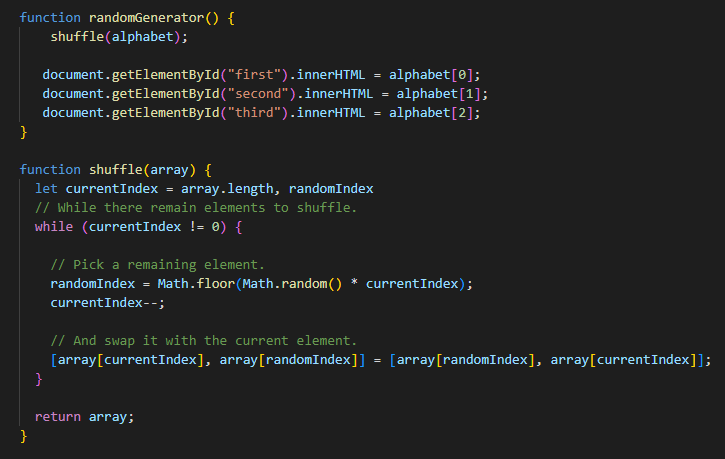
In the last table row, we display the 3 constantly randomising letters, we give each letter a different ID attribute and have each onclick call a function while passing its own ID into the function parameters. This allows us to call the same function for the validation of the random letters against the chosen letter. Additional CSS is added to format the letters within the cells of the last rows.



For the JavaScript, we start by declaring variables outside the functions, these are known as global variables, and they can be accessed by any functions within the script. The first variable alphabet is an array of all letters of the alphabet, this is done by using the String split() method, the String of letters are split letter by letter and added into an array called alphabet. Next, we have an alert schedule for setting the change interval of the random letters, a variable to keep track of the total score and lastly a variable to return a boolean value on whether our game has started or not.

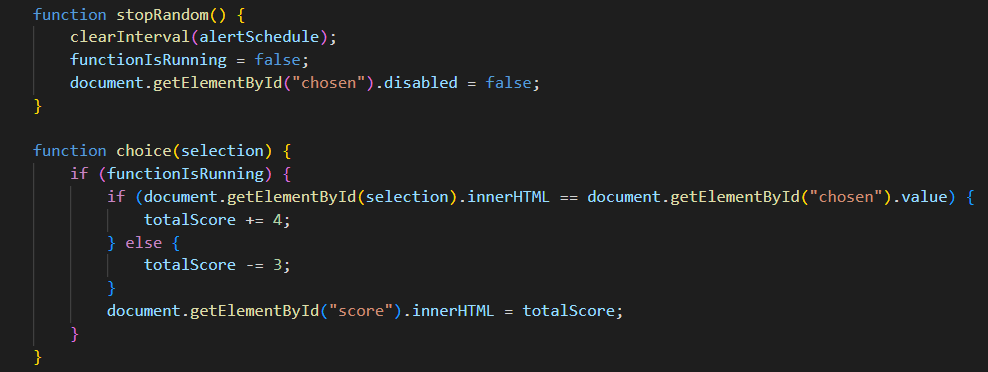
The first function, startRandom() is called when the start button is pressed by the user. The function first checks whether the game is already running, if the game is running, it disables the onclick for the start button. This prevents the user from starting the game more than once. If it is the user’s first time pressing the button, the boolean variable is set to true and the drop-down menu is disabled, so that the user does not change the chosen alphabet midway through the game. Next the total score in the <span> tag is set to 0 by default, so that when the user restarts the game, the total score resets too.

The alertSchedule calls the setInterval method which calls the randomGenerator method every 1000ms or 1 second.



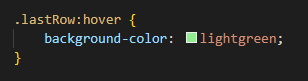
The randomGenerator function works in correlation to the shuffle function. We start by explaining the shuffle function. One problem we had to solve was the displaying of randomised letters without any of the letters repeating. After some research, we found a simple and elegant solution. By implementing an algorithm called the Fisher-Yates shuffle, we are able to shuffle the elements of the alphabet array within its own array. The shuffle works by going through every index of the array and throwing them into a new random position. This means that after every shuffle, the 26 letters of the alphabet will be at a different index in the alphabet array. Since we only need to show 3 letters, we display the first, second and third element of the array. This solves our problem of repeating alphabets as the array only consists of the 26 alphabets.

The flow of our program is as follows. After the start button is pressed, the randomGenerator function is called every 1000ms. When the randomGenerator function is called, it passes the alphabet array into the shuffle function to be shuffled. The alphabet array is then returned, fully shuffled. The first 3 elements of the array are then displayed with the innerHTML.

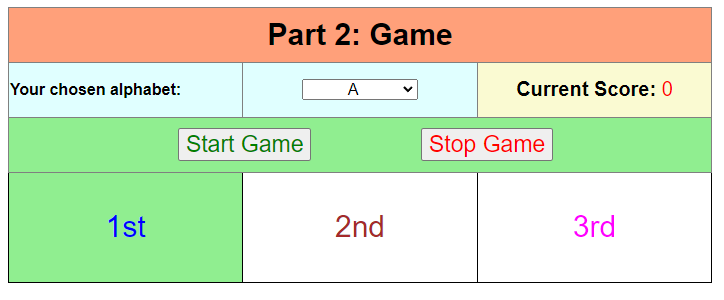


Next, we have the stopRandom function, this function stops the Interval and changes the boolean value of the functionIsRunning to false, indicating that the game is stopped. The drop-down menu is also enabled so that the user can reselect an alphabet.

The choice function validates the user's inputs against the chosen letter in the drop-down menu. The function requires a selection parameter to be passed which is the ID of the first, second and third letter. A simple if else statement compares the ID of the clicked letter against the chosen letter. If it is the same, the score is incremented by 4, otherwise the score is reduced by 3. The total score value is constantly updated every time one of the letters is clicked.



Last but not least, we have additional CSS for the last row of the table. This makes it so that when the user hovers over the cell he wants to click, the cell will light up green.



**Part 3:**

**XML**



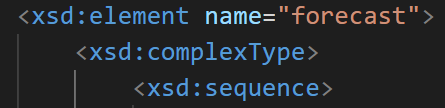
In the given xml document, A2.xml, we have added the element, xml-stylesheet.

This element is to link the xml document to our xsl document, enabling the transformation and formatting of the xml document. The attribute (href = “S2.xsl”) is used to link the external xsl document, which is named “S2.xsl”. This allows the xsl document to retrieve elements and their values, format them and display the way that we want them to be. Attribute (type = “text/xsl”) is used to define the type of document it’s linking to, which in this case is text/xsl type.

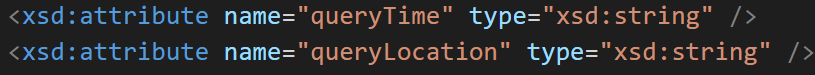
**XSD**

****

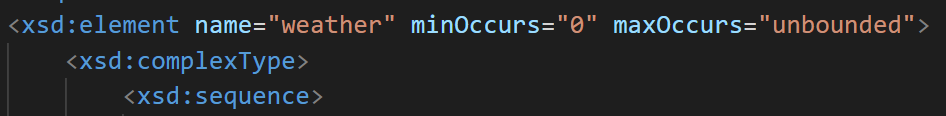
We start off the xsd document with the xsd:schema element to define that this is a xsd schema. In this element, we use the xmlns:xsd to define the standard namespace http://www.w3.org/2001/XMLSchema. The element and data types that we will be using in this schema will be from the namespace.



After that the root element of the xml document will be placed first, which is “forecast”. “forecast” contains a sequence of weather elements so it is a complexType. The element <xsd: complexType > is placed after. The sequence element placed after, specifies the child elements that must appear in the sequence.



In the xml, the root element, “forecast”, contains two attributes of “queryTime” and “queryLocation”. These attributes are placed at the ending tag in-between the closing tags of complexType and sequence.

****

Next, we have the element for “weather”, a complex type, which contains a sequence of elements. The “forecast” element can contain zero or unlimited number of weather elements, hence the “weather” element has the attribute of minOccurs= “0” and maxOccurs= “unbounded” which stands for zero and unlimited respectively. “weather” has a sequence of elements hence we used <xsd:sequence>.

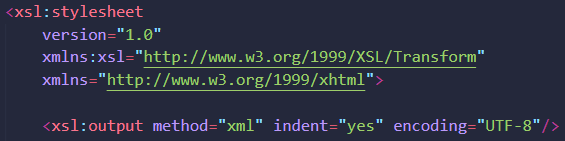
****

In the child elements for “weather”, we have “year”, “month”, “date”, “dayOfWeek”, “overall”, “overallCode”, “highest” and “lowest”. for “year”, “month”, “date”, “highest” and “lowest”, the attribute will be type = “xsd:byte” as they are number types. For “dayOfWeek”, “overall” and “overallCode”, the attribute will be type = “xsd:string” as they are words and sentences. All of these elements are simple types as they do not contain other elements.

**XSL**

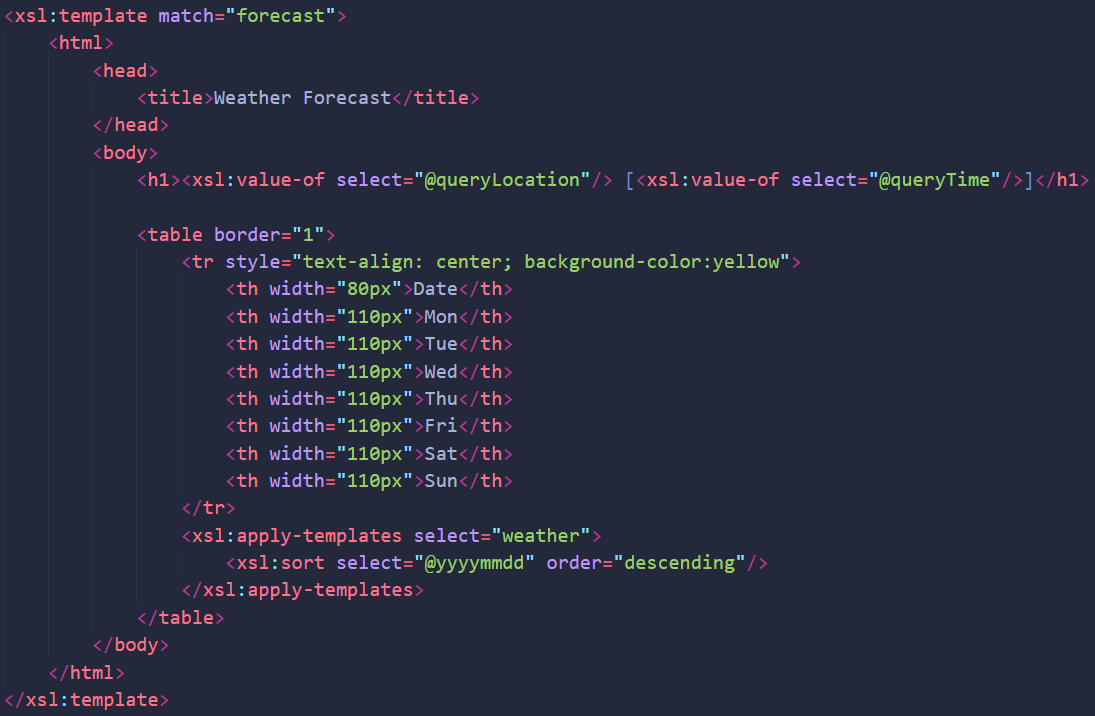
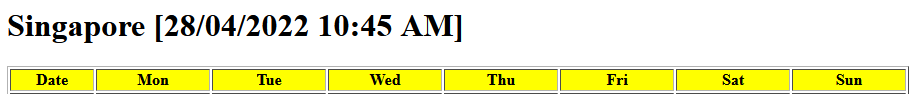


Using the provided A2.xml, we will create a stylesheet S2.xsl which is used to transform the XML file into HTML. To make use of this .xsl file, we have to add the sentence above at the top of the .xml file after the xml version and encoding:



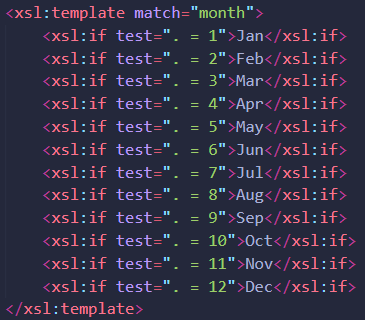
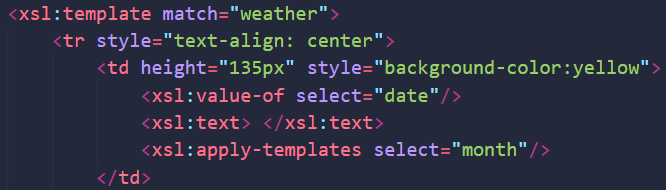
Inside the .xsl file, the version, encoding, namespaces and output method are declared at the top of the file.

A template associated with the element "forecast" is created first. Inside the template, contains a HTML tag to represent the root of the HTML document. Inside the HTML tag, is the head section consisting of title tag, and then the body section.

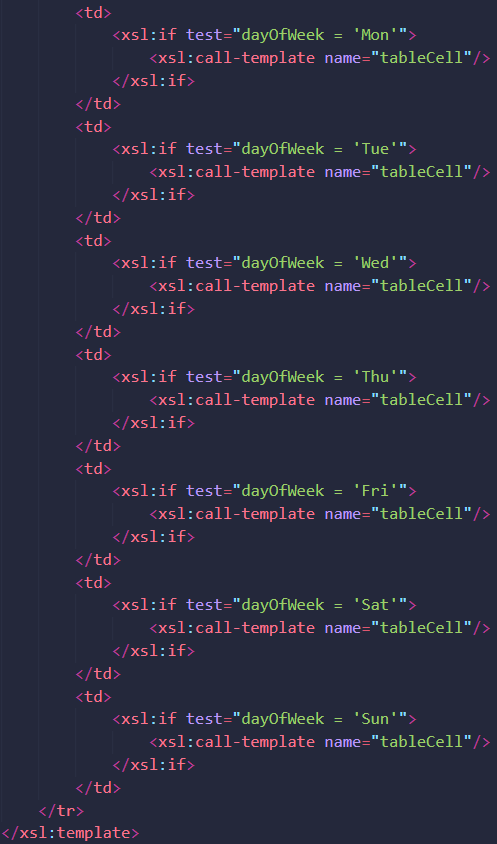


The first thing in the body tag, is the topmost heading defined by <h1></h1>. Inside the heading, is the value of xml attribute "queryLocation", extracted using <xsl:value-of/> and adding "@" since this is an attribute, plus the value of attribute "queryTime" enclosed in a square bracket.

Next, is a table with borders set around the table cells. The first row <tr></tr> of the table consists of 8 columns of header cells, defined by <th></th> tag. Inline CSS is used to style the row to centralise the text and make the table cell yellow, each header cell is also styled to set the width of the cell. After this row, a template is applied, selecting the attribute of "weather". Each weather is sorted by the attribute "yyyymmdd" and in descending order from latest date to earliest date.



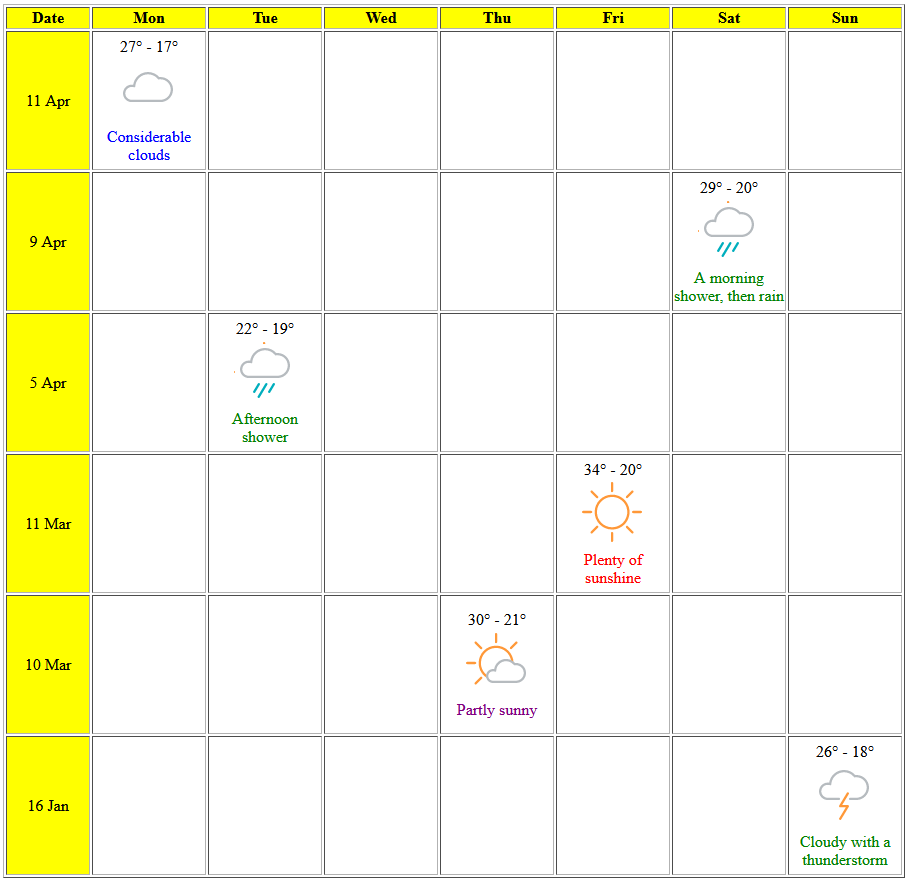
The next template associated with weather is used to apply rules to each weather element in the xml. It starts with a table row with inline CSS to centralise the text within. Inside the row, there are 8 table data <td></td> for each column of the table. The first table data is for the date, with styled height and yellow background. The date is formatted using the value of the "date" element and a space using <xsl:text> </xsl:text>. The name of the month is displayed using another template associated with the "month" element. Inside this template, use of <xsl:if></xsl:if> checks the value of each weather's month against the 12 months of the year. For example, if the month of the current weather is "1", it will display "Jan" in that table data. The "." or dot character refers to the context item.



The remaining 7 table data are for each day of the week, according to the table header cells. In the first table data, if the "dayOfWeek" element of the current weather is 'Mon', a template with name "tableCell" is called to fill the table data with respective weather data, if not the table cell will remain present but empty. The next 6 days of the week are done in the same way in subsequent table data.



When the named template is called, it will display the temperature, weather image and overall description of the current weather being processed. The temperature is presented using the value of "highest" element to "lowest" element, and the degree Celsius symbol is displayed using UTF-8 encoding "&#176". The image is presented using <img></img> tag and xsl:attribute, inside the tags is the name of the images to be displayed, which is based on the value of "overallCode" element. Lastly, for the weather description, a xsl:choose is used in conjunction with xsl:when and xsl:otherwise to express multiple conditional tests. When checking the "overallCode" element, if it is 'cloudy', the value of the "overall" element is displayed in a <span> tag, styled with blue colour. When the "overallCode" element is equal to 'thunderstorm' or 'rain', a <span> tag with value of "overall" element is displayed in green colour. When "overallCode" is equal to 'sunny', <span> tag with value of "overall" element is displayed in red colour. Otherwise, in other words if "overallCode" is not equal to any of the conditions stated above, and is for "partlySunny", <span> tag with value of "overall" element is displayed in purple colour.



Once all "weather" elements in the "forecast" element are processed, a table with all the weather data is completed.