NAME \_Filipa Ivanković\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Project 1 – Rubric

ISTE340 Client Programming

|  |  |  |  |
| --- | --- | --- | --- |
| **✔** | **Tasks** | **Points** | **Score** |
| **Web Design and Programming Principles** | |  |  |
| ☐ | Professional looking web application, following modern web design principles and practices, that provides a meaningful content to the user.  The website is styled to suit the appropriate style for the old Pokémon games with some of the darker color choices. The index.html, the main page leads to the two other pages: pokedex.html and form.html. The content provided on this page is minimal, as the primary source of information will be provided from the pokedex.html page, which contains all the main data for every Pokémon. The content on the main page is short about the origins and history of the Pokémon franchise, and generally about Pokemon as a whole. Hovering over the elements makes them appear darker and makes them differentiate from the background, so it is easier to read.  Near the bottom of the page is the Pokemon Selector which is responsible for the dynamic image and background change generation.  The most important content is on the Pokedex.html page which displays all the data for all the Pokemon. Pokemon are generated from the API data and displayed in appropriately sized cards. Each Pokemon has their id, name, image, typing, abilities and link to the Pokemon Wiki displayed on their card. The background of each card is determined by the Pokemon’s typing.  The sticky navbar is present on all the page, as well as the footer.  Several version of the .ico have been made to support different types of browsers. | 10 |  |
| ☐ | Professionally structured & implemented website, following good programming principles and techniques (OO approach with the MVC architecture based on ES6 features)  Due to some of my misunderstanding, the MVC structure is a bit flaky. The Controller and the View portions are melded together, as I was no aware of certain rulesets when it comes to the division of the Controller and View in JavaScript. But I figured these things out a bit too late, which resulted in me grossly underestimating the complexity and amount of time needed to implement the full MVC structure. In place of the model is the database with the Pokémons that are to be displayed in the Pokémon Selector parts of the webpage. I believe that the Pokedex API portion of the webpage upholds to the MVC standards, but the SelectionController and Form - not so much.  The webpage does however use the “<script type=”module>” of exporting and importing the classes to start them, instead of the usual “window.onload”.  I tried to uphold the correct programming principles and avoid the DRY; however, due to the MVC being lackluster, some of these things may not be apparent as much.  The MVC part of this webpage is the weakest link by far. | 10 |  |
| ☐ | The code is properly aligned and indented AND documented using comments in the expected format (JSDoc) that clearly explain the code functionality.  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  The code is documented, and properly aligned. Each class, function, and important object has comments written for them. Each comment for a class or function contains the @return, @param (if available) and other important stuff written out.  The comments should provide enough information on what each class and method does. Classes and objects have also been commented.  All pages and .js should be properly aligned. | 10 |  |
| **Conditional/Cascading Selects** | |  |  |
| ☐ | Each select must have at least 2 choices and there must be a depth of 3 different selects  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  The Pokemon Selector is present on the index.html page.  The JavaScript code that executes this code is contained in the SelectionController.js class, and the data that is being displayed is taken from the pokemonData.js.  There are 3 different select types. The first one is the choice of the Pokemon image. There are 5 Pokemon choices. The second select is about the color of the Pokemon image. Each Pokemon has at minimum at least 2 different color variations. The last select is related to the background which displays behind the Pokemon Image. Each Pokemon has at minimum 2 different backgrounds to choose from. Each Pokemon uses a unique color and background.  More Pokemon, colors and background can be easily added by simply changing the pokemonData.js. All that is necessary is to provide the images into the “media”, the correct pathing and following the naming convention of the other properties. Other classes do not have to be changed to add additional Pokemon, colors or backgrounds.  All the colors and backgrounds are contained in the pokemonData.js database.  When a user selects an option, the change is reflected, and the image is changed appropriately to the user’s choice. The same thing happens when the user changes the background image. | 5 |  |
| ☐ | Selects are dynamically created depending upon the user’s previous select option.  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  Yes, they are. When the user selects the Pokemon from the Pokemon Dropdown, the color options will be created through JS, and then those can be accessed. After a color is chosen, the backgrounds become available for selection. Each Pokemon has their own color variants and background available to them. These images and backgrounds are unique to them, which means that other Pokemon can’t share in-between themselves a background or color image.  The options for the Pokemon Dropdown select are generated as the index.html is loaded. The number of available Pokemon should reflect the number of keys from the pokemonData.js  The code for this is found in the SelectionController.js, and the data for the Pokemon Images is in the pokemonData.js. | 5 |  |
| ☐ | The user can start over and the selects are re-drawn from the point of the user’s new selection  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  Yes, the user can go back and re-pick their choices. After the Pokemon image is selected, the Pokemon colors are generated via JS. The Background options are generated when the user picks the color for the Pokemon Image.  The user can freely go back to the previous options which will reset the color and background selects. When re-choosing a Pokemon image, the pokemon will return to the color1 option which the default state. After choosing a new Pokemon, the color options are generated once again. Same thing with the backgrounds.  There is also a reset button, which will remove the image and the selected background.  The user must choose a Pokemon, color and background in order to proceed to the form.html page, where the form is contained. The state of each select is tracked via “Boolean” values.  The code for this is in the SelectionController.js. | 5 |  |
| ☐ | The selection process includes a dynamically generated UI element that changes in response to the user's selection  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  The image and background change as the user selects their option. When the user selects something and option the is available, the div that holds the image will reflect the change. When choosing a background, the same will be reflected in the appropriate div.  Once the user submits their selection, their selection is stored in the local Storage. This data is then displayed and written out onto the form located on form.html. The user selected pokemon image, color and background will also be displayed on the form.html in a similar form structure as the one that is present on index.html.  Found in the SelectionController.js. | 10 |  |
| **Form Element** | |  |  |
| ☐ | Once the user is done with the selection, a form is used to send the user selected options as well as additional data that are application specific (e.g. email, age, ...).  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  The form asks the user for their name, email, password and repeat of their password. Each entry musn’t be left blank. Email must be in a correct format and can’t contain illegal characters. Password cannot be shorter than 7 characters and the repeat password must match the main password.  Found in the FormValidator.js and FormView.js for displaying the changes to the form. | 10 |  |
| ☐ | The form inputs are validated, and an informative feedback with restyled form elements is provided in case of invalid data. At least 2 inputs with an expected format to validate are provided. Validation needs to be performed by JavaScript not by HTML5.  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  When the user tries to submit their solution, the FormValidator.js will go over each input and their value to see if the inputs are valid or invalid. If an input is valid, the input will be highlighted green and a green checkmark icon will appear within the input. On the opposite end, on a invalid input, the input will be highlighted red and a red exclamation icon will appear. On the invalid input, a error message above the input field will appear. The error message will tell the user exactly what needs to fixed.  Located in FormValidator.js and FormView.js | 10 |  |
| **Web Storage** | |  |  |
| ☐ | Once the user is done with the selection, the Web Storage is used to store the selected options.  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  On submission, when all selections have been made, and no selection is left not selected, the choices of the user ill be stored in the local web storage.  In the same vain, when submitting the user inputs for the form, after all the values have been determined valid, the input values will be stored into the local web storage.  Storing the Pokemon Selector values is in the SelectionController.js, while storing the values for the form are in the FormValidator.js class. | 5 |  |
| ☐ | The data regarding the user choices are loaded from the Web Storage into the form. On form submit, all the data are stored in the Web Storage  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  When the form is submitted, the FormValidator.js checks if all the inputs are valid or invalid. Depending on the state of the data entered the inputs will reflect it through change (green lit on valid, red on invalid). Also, a error message will be made above the input field if the input was invalid.  Once all the inputs are valid, the user entered data is stored into the local web storage.  If there exists any valid data while on the form.html page, that data will be displayed on the page. If the user has previously selected a Pokemon from the Pokemon Selector, and has entered correct data into the input fields, those data will be loaded into the appropriate places.  Code is in FormValidator.js and FormView.js | 5 |  |
| **Browser Support** | |  |  |
| ☐ | The application checks how well the user’s browser supports the application’s features and redirects the user to a different web page in case of no support. The check is performed in the beginning of the program, before allowing the user to interact with the web page.  TODO: Explain what you did here, and if applicable, where to find this feature in your code (e.g. init.js, line12)  Certain .css features are made in mind to support multiple types of browser. Bootstrap is used to make the elements of the html pages more responsive and better suited for various kinds of devices and browsers. Certain elements and their attributes are changed with media queries. | 5 |  |
| **SUBTOTAL:** | | **90** |  |
| EXTRAS: An excellent project (grade A) will have all of the requirements above PLUS something extra as listed below. The extra features (whether they are listed below or not) need to be discussed and approved by your instructor in order for you to proceed with the implementation. You cannot get more than 10 points for this rubric. | | | |
| ☐ | The extra content is the usage of the Pokemon API. The pokedex.html displays the data that is taken from the API in the form of cards that hold the image, data and a link to the Pokemon wiki.  The code that fetches the data from the API is within the PokedexFetcher.js class. That class sends the extracted data to the PokedexView.js class that styles the data appropriately with .css and properly formats it for displaying. The card holding the data is then appended to the main pokedex.html id “pokedex” where all the cards will be displayed. The range for how many pokemon want to be displayed can be found in the PokedexFetcher.  The variables which determine the range of the Pokemon showns are appropriately called “startingPokemon” and “finalPokemon”. The minimum is 1 to a max of 898 Pokemon can be displayed from the API. Also, the PokedexView.js class uses the typesColors.js object which holds hex color values that are added to the Pokemon cards backgrounds. Each hex color is given to a card if the name of the hex color matches the “type” of the Pokemon which is taken from the API. | 10 |  |
| EXTRAS TOTAL: | | 10 |  |
| TOTAL: | | 100 |  |
| **COMMENTS:** | | | |