**FA1 Hands-on Integrated QP**

**Instruction to Candidates:**

* It is a controlled book test. Please follow the guidelines provided by Invigilator
* The code will be evaluated only after submitting it to the submission folder (server).
* **If the timestamp of the file is greater than the submission time the test will not be evaluated.**
* **You are not supposed to use any network resources**

**Instructions to use the project file provided**

* Download the project **CulinaryInstitute - ToTrainee** in to your system and unzip it
* Open a new workspace in Visual Studio Code
* Open only the **CulinaryInstitute - ToTrainee** folder in your Visual Studio Code workspace
* Do not change the partial code given to you.
* Do not change the IDs.
* Make sure your output is exactly the same as the screenshots.
* Always run the live server(port:5500) in chrome browser.
* Use Bootstrap classes wherever possible
* **Please note to check your scripts, testcases has been provided**. To run the test cases follow the given steps:
  + Open the terminal and run the command “npm install”, Do it only once
  + Run the javascript file **reportGenerator**.**js** through node
  + Open reporter.html with live server , this will show the test cases passed or failed
  + To re-check your code, follow the above two steps

**Problem description**

**Culinary Institute** is a well-known cooking institute. They operate in various parts of India and offer classes for various cuisines. You can pay online to register for a particular class.

The files present in the given project are described as below:

# 

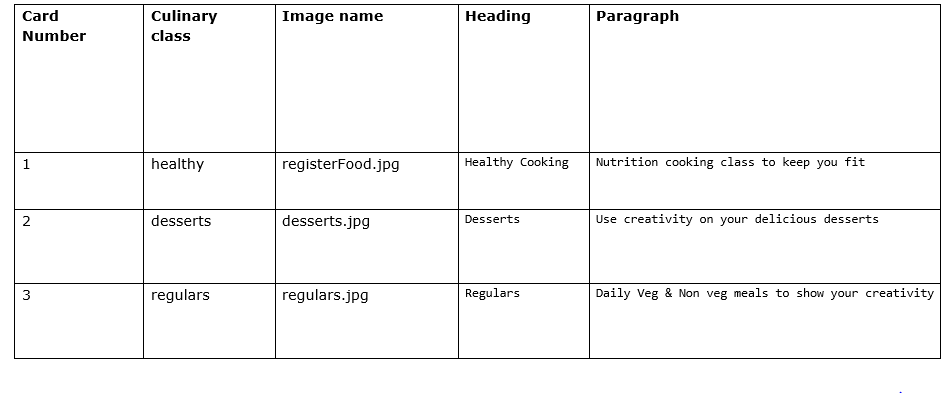
# D:\FA1 Related Artifacts\FA1_Integrated\New folder\Integrated\Latest@Updated\capture6.PNG

# index.html

The view of **index.html** should be as shown below:



* The **Navbar** is **rgba(71, 57, 82)** color with a **transparency of 0.863**
* The page heading should have the following features:
  + It should occupy **8 columns** with an **offset of 2** columns in small, medium and large screens. It should occupy the full width in xs screens.
  + It should use proper **bootstrap class** to set font-size to 3.5rem, font-weight to 300 and line-height to 1.2
  + Use the **toPad** class to add a **padding of 4rem** and a **font color of whitesmoke** to the heading. Style the .toPad class in the **stylingIndex.css** file.
* Create **3 Bootstrap cards** to display the different **types of culinary classes** as shown in the screenshot. The cards should have the following properties:
  + The cards should be placed in a **column grid of 3 in medium and large screens**. They should occupy the entire screen width in smaller screens.
  + On **clicking any of the cards** the user should **navigate to register.html** page. Put the card in **anchor tag** to achieve this.
  + Use appropriate **bootstrap class** to place the **image at the top** of the card. **Make sure all the images have appropriate attributes to increase accessibility.**
  + The images **transform to a scale of 1.01 on hover**. Add **transition** to the **img tag** in **stylingIndex.css** to add a transition to transform all the properties with a **duration of 0.2s** and **delay of 0s** and make sure the transition **eases in and out**.
  + Inside the card there should be a **heading (<h5>)** to display the type of culinary class as shown in the screenshot
  + There should be a paragraph after the heading to display the details of the respective culinary class as shown in the screenshot
* Use the below table for reference while calling the store function and populating data and images in the card:

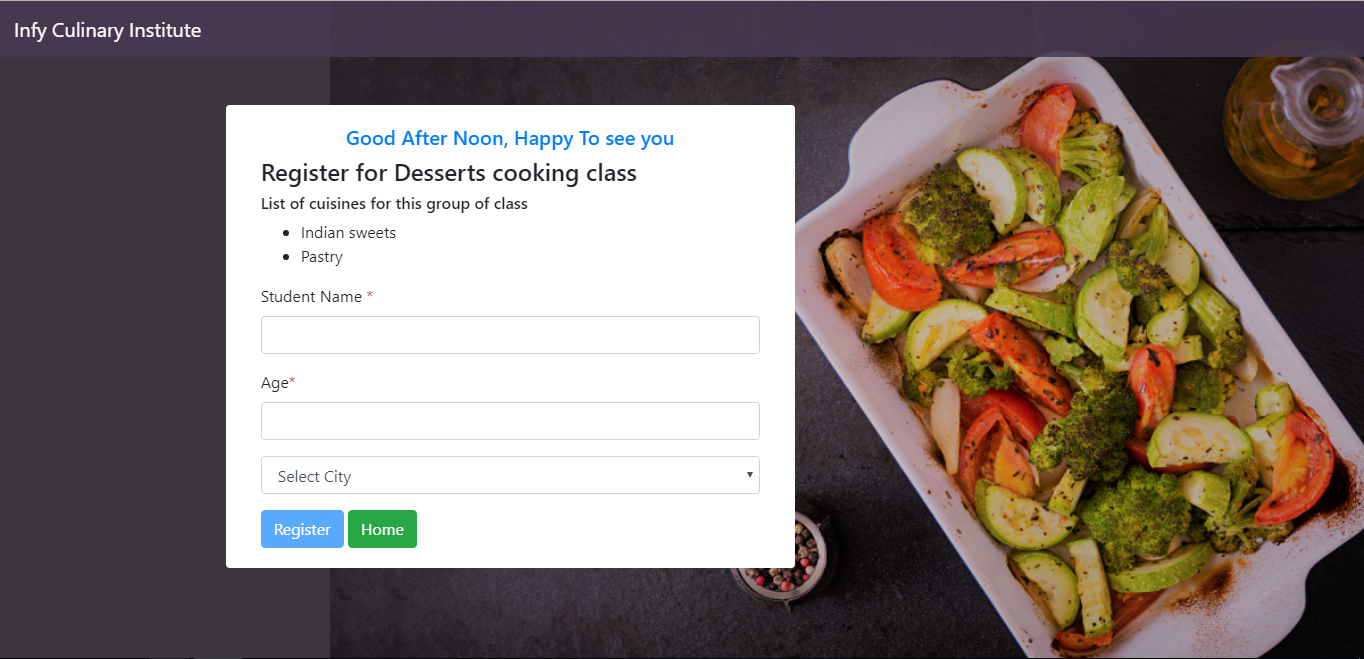


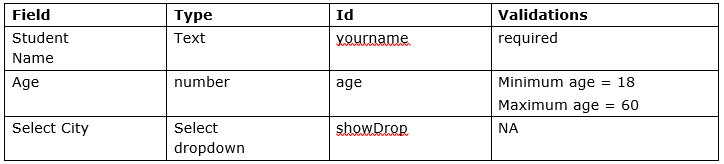
**index.js**

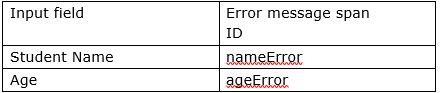
* **store(): onclick** of the **card or card-image**, this method is invoked
  + Get the **type of class** selected by the user and store it’s value in the session storage as **interest**

# register.html

* On clicking on any of the **Culinary class cards** in index.html, the view of **register.html** is as shown below: (Ex. desserts class is selected)



* The **card** (whose structure is already given) should be in a grid system of **5 columns** with **2 offset** for a **medium screen**. It should occupy the entire screen in small devices.
* The card should contain the following:
  + A greeting message in a **heading (<h5>).** Add appropriate **bootstrap classes** to **position the heading at the center** of the card and **add color** as shown in the screenshot. This message should be **populated dynamically on the showDropdown** function in the register.js file. Use the **showGreeting id** for the tag.
  + A **heading (<h4>)** to display the message *“Register for <selected class> cooking class”*. This message should be **populated dynamically in the getFromSession** function in the register.js file. Use the **type id** for the tag.
  + A **heading (<h6>)** to display – “*List of cuisines for this group of class*”. This heading should be **statically populated**.
  + An **unordered list** to display the **list of cuisines**. The list should be **dynamically populated in the getFromSession** function in register.js file. Use the **list id** for the tag.
  + A form with following input fields, input types, ids and validations:
  + 
  + The **enableButton** function should be called when the **user types something in the Student Name field**.
  + The **checkAge** method should be called when the **user types something in the Age field**.
  + The **Select dropdown** should be **populated dynamically in the showDropdown** method. The **enableButton** function should be called if the **user selects a city in the Select City dropdown**.
  + Refer the below table to populate **error messages** (if any) for respective input field:



* + Ensure all the **error messages** are **red in color** using appropriate bootstrap classes.
  + The **error messages** should be **populated dynamically** and displayed just after the respective input fields.
* There should be two buttons:
  + A **register button**. This button should be disabled by default and should get enabled only when the user has entered valid inputs in all the fields.
  + The **home button** should take the user to the index.html page. Style the buttons using appropriate bootstrap classes as shown in the screenshot.
* **All labels must have proper labels with “for” attribute.**

**Note: Ensure that you use the ID’s mentioned below. Else you will lose marks.**

****

**Note: Refer GIF for further understanding**

**Note: A utility file is already linked with this both index and register pages. If you open the console, it will tell you if any ID’s are missing.**

**Register.js**

**getFromSession() :** This function must be invoked **onload** of register.html

* This function should make an AJAX request to the given URL and fetch an object containing different types of classes.
* It should get the type of class selected by user from session storage, the heading should be populated as:

If “Healthy cooking class” selected, heading should be, **Register for Nutrition cooking class**”

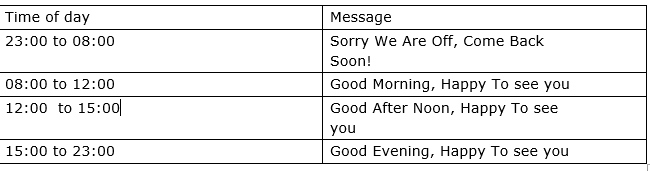
If “Desserts class” selected, heading should be, “**Register for Desserts cooking class**”

If “Regulars class” selected, heading should be, “**Register for Regular cooking class**”

* Populate the varieties, corresponding to type of class selected, from types.json file and insert them as **list items** into the **ul** tag provided, as shown in the screenshot.
* If there is any error in the above steps use exception handling to catch the error and display the following error message in a span tag in place of the list:
  + *“Couldn't fetch data”*
* Also populate the city array, corresponding to type of class selected.
* Finally error or no error the showDropdown method should be called and the city array should be passed as a parameter to this function.

**showDropdown():** This function must be invoked **in the finally block** of **getFromSession method**

Greet the user using appropriate messages based on the time of day as per the below table:



Populate the above message in the h5 tag with showGreeting id.

Populate the Select city dropdown with the showDrop id using the city array.

**enableButton():** This function must be invoked **onkeyup** of the name input field and **onchange** of select city dropdown

* Get the name entered by the user and the selected city.
* If the name, age and selected city are valid then enable the register button else disable it.sss
* Use agevalid global variable defined in Register.js

**checkAge():** This function must be invoked **onkeyup** of age field

* Get the age entered by user.
* Check if age is greater than 18 and less than 60.
* Enable the Register button and make ageValid to be true.
* If not, populate an error message in a div tag with id “ageError”
* Message: “Sorry, you should be between 18-60 years old!”.
* Register button should be disabled and make ageValid to be false.

**register():** This function must be invoked **onsubmit** of the form.

registrationAmount = {

"healthy":4500,

"desserts":5000,

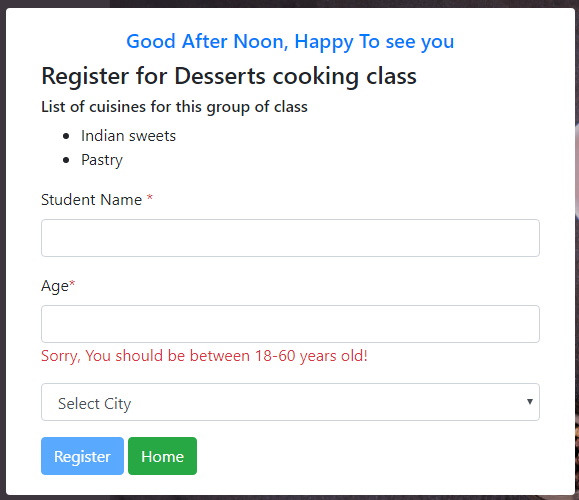
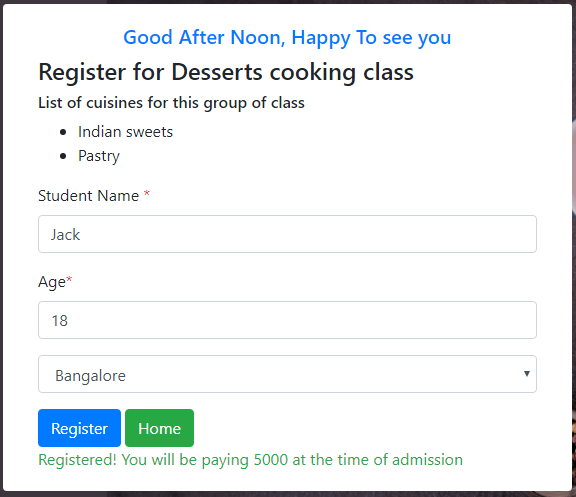
"regulars":3500

}

* Calculate the amount to be paid based on the object given above.
* Populate a success message in a div tag with “successMessage” as id
* Message: “Registered! You will be paying <<amount>> at the time of admission.”

Note: When error messages are appearing on the page, success messages/information messages should disappear and vice versa.

**Sample Input:**

* Use appropriate bootstrap classes for success and error messages.

**~~~~~~~~~~ALL THE BEST~~~~~~~~~~**