Programmer Documentation

Project Overview:

Project Name: PacaThon.

Description: A cloud-based Interactive educational game that aims to teach Cloud Computing. The game incorporates a competitive element, in which users can promote their ranks based on their progress and participate in competitions with other players.

Table of Contents:

1. [Project architecture](#Project_Architecture)
2. [Essential libraries](#Libraries_used)
3. [Functions](#Functions)

Project architecture

The project is written in python, and uses HTML, made in “google colab”, a cloud-based Jupyter notebook environment. The project utilizes an external database “Firebase”, to store user data and necessary game resources such as questions etc.

Essential Libraries:

1. google.colab

To invoke functions within the embedded HTML code

1. numpy

Statistical information for the player

1. matplotlib

Plotting and displaying statistical information

1. IPython

To process and display embedded html code with python.

1. firebase and firebase\_admin

Read and write data to database

Functions:

1. Signup()

enables players to sign up for the PacaThon game by submitting their information through a signup form. The entered data is processed, validated, and stored in a Firebase database.

* The **html\_signup** variable stores a multi-line string that represents the HTML markup for the signup form. This form collects information such as the user's first name, last name, email, username, password, and confirm password.
* The **process\_data** function is defined within the **signup** function. This function is necessary since the HTML is written in a string format, therefore, in order to handle user interactions with the webpage, the function **google.colab.kernel.invokeFunction()** is utilized to invoke the corresponding python functions. This method of handling user interactions is used in most pages.
* Within the **process\_data** function, there are conditional statements that handle user actions based on the button clicked. If the user clicks the "Login" button, the **login** function is called. If not, the form input is checked for validity using the **checkValid** function.
* The **checkValid** function iterates over the existing user data stored in the Firebase database to ensure that the entered email or username is unique and does not already exist.
* If the form input is valid and the user clicked the "Create User" button, the code establishes a connection to the Firebase database and creates a new user entry with the provided information. The user's rank is initially set as "trainee," indicating their starting position.
* After processing the data, the code clears the output and calls the **login()** to move to the login page.
* **display(IPython.display.HTML(html\_signup))** is used to display the embedded html.
* The **output.register\_callback('notebook.runPython', process\_data)** line registers the **process\_data** function as a callback for handling user interactions in the form. It allows the JavaScript code in the HTML form to invoke the **process\_data** function with appropriate parameters.

1. login()

Displays a login form with input fields for username and password.

* The code establishes a connection to the Firebase database by creating an instance of **firebase.FirebaseApplication** and assigning it to the **FBconn** variable. The Firebase URL is provided as an argument to establish the connection.
* The **userData** variable is used to retrieve user data from the Firebase database. It retrieves all user records stored under the **/Users/** folder in the database.
* The **html\_content** variable stores a multi-line string that represents the HTML markup for the login page. This markup includes styling for the login form, such as background colors, font styles, input fields, and buttons.
* The **process\_data** function is responsible for processing the data entered in the login form and handling user actions based on the button clicked. Same method as explained in [signup()](#signup_func)
* Within the **process\_data** function, there are conditional statements that check the value of the **val** parameter to determine the user action. If the value is **'Login'**, the code iterates over the **userData** to find a matching username and password combination. If found, it checks whether the user is an admin or not and redirects them accordingly to the **landingPage** or **adminPage**.
* If the user action is **'Signup'**, the code clears the output and calls the **signup** function, which is likely responsible for handling user registration.
* The **display(HTML(html\_content))** line displays the login form in the notebook output cell using the HTML content)
* The **output.register\_callback('notebook.runPython', process\_data)** is explained in [signup().](#signup_func)

1. StartGame():

Displays the game page to the player:

* The code connects to the Firebase database using the **firebase.FirebaseApplication()** function, passing the database URL as a parameter.
* It retrieves questions from the Firebase database using the **get()** function from **FBconn**. The questions are stored as JSON strings.
* The code then parses the JSON strings and appends the question data (as lists) to a **list\_of\_lists**.
* It selects eight random questions from **list\_of\_lists** and stores them in **list\_of\_random**.
* The code defines an HTML template for the quiz interface using a multi-line string (**html\_code**). This template includes CSS styles for formatting and a JavaScript section for handling user interactions.
* Within a loop, the code dynamically generates HTML elements for each question using the data from **list\_of\_random**. It creates question containers, displays the question text and answer options, and attaches event handlers to the answer buttons.
* The JavaScript section of the HTML template handles the logic for displaying questions, checking answers, and updating the score. It keeps track of the current question number, handles the progress bar, and updates the score based on the correctness of the selected answer.
* The JavaScript function **checkAnswer()** is called when a user selects an answer. It compares the selected answer with the correct answer, updates the result display, increments the score, and keeps count of correct and wrong answers.
* The JavaScript function **showQuestion()** is responsible for showing the next question when the "Next" button is clicked. It updates the question container's visibility and progress bar.
* The JavaScript function **ChangeButtonText()** is called when the "End Game" button is clicked. It invokes a Python function **process\_data1()** to handle the end game event. **process\_data1()** implements same functionality as explained in [signup()](#signup_func) function.
* Finally, display the generated HTML content using the **display()** function.

1. resultsPage():

* Reading Data from the Database: The code retrieves data from a Firebase database. It first assigns the value of **usernameTemp** to the variable **username**. Then, it retrieves the **correctAnswer** and **wrongAnswer** values from the function arguments and assigns them to **correct** and **wrong** variables, respectively. It initializes some variables (**oldCorrect**, **oldWrong**, **oldScore**, and **ID**) with default values and then queries the Firebase database to find the corresponding user data based on the username. If a match is found, it updates the variables with the retrieved data.
* Updating Data in the Database: The code calculates the updated values for **oldCorrect**, **oldWrong**, and **oldScore** by adding the current values (**correctAnswer**, **wrongAnswer**, and **score**) to the retrieved values (**oldCorrect**, **oldWrong**, and **oldScore**). Then, it updates the corresponding fields in the Firebase database for the user with the updated values.
* HTML Code: The code defines an HTML string containing the structure and styling for the results page. It includes a background image, a container with a form, and a chart area for displaying the game results.
* Chart Rendering: Inside the HTML code, there is a JavaScript section that uses the Google Charts library to create a pie chart. The chart data is generated based on the **correct** and **wrong** variables. The chart is then drawn on the page using the **google.visualization.PieChart** class.
* Button and Callback: The HTML code includes a "Proceed" button with an **onclick** event handler that triggers the **mainPage()** JavaScript function. This function uses **google.colab.kernel.invokeFunction** to invoke the **notebook.runPython** function from the notebook's kernel.
* Callback Function: The code defines a Python function called **process\_data1()** that is used as a callback for the **notebook.runPython** event. This function clears the output and calls the **landingPage()** function.
* Displaying the HTML: The **display(HTML(html\_code))** line displays the HTML code as an output, rendering the results page in the notebook or the corresponding environment.

1. landingPage():

* Initializes a dictionary named **DB** to store user data from the Firebase database.
* Retrieves user data using the Firebase connection and stores it in **DB**.
* Fetches the rank and score of the current user from the **DB** dictionary.
* Defines a string variable named **styleLandingPage** containing CSS styles for the landing page HTML.
* Defines an HTML string named **htmlLandingPage** representing the structure and content of the landing page.
* Defines JavaScript functions for button interactions: **startGame()**, **logOut()**, **help()**, and **leaderboard()**.
* Functions are called when corresponding buttons are clicked.
* Uses **google.colab.kernel.invokeFunction** method to invoke Python functions defined elsewhere in the notebook.
* Defines the **process\_data1** function as a callback function.
* Depending on the value of the **logout** parameter, it clears the output and calls **login()**, **help()**, **startGame()**, or **leaderboard()** functions.
* Calls **HTML(htmlLandingPage)** to display the landing page content.
* Registers the **process\_data1** function as the callback for **output.register\_callback**.

1. leaderboard():

Generates a leaderboard page displaying user rankings and scores.

* A dictionary **DB** is initialized to store user data retrieved from the Firebase database.
* User data is fetched from Firebase using the **firebase.FirebaseApplication** class and stored in the **res** variable.
* The user data is processed to filter out admin users and store relevant information in the **DB** dictionary.
* The **DB** dictionary is sorted based on user scores in descending order and stores 5 users who have the highest scores.
* If the number of users in the **DB** dictionary is less than 5, additional "NONE" users with a score of 0 are added to make a total of 5 users.
* Two lists, **usersnames** and **usersscores**, are created to store the usernames and scores from the **DB** dictionary, respectively.
* CSS styles for the leaderboard page are defined in the **styleForLeaderboard** variable.
* The HTML structure of the leaderboard page is created using the **html\_leaderboard** string variable, which includes a table displaying the top 5 users' ranks, usernames, and scores.
* A JavaScript function, **Back()**, is defined to handle the "Back" button click event, which invokes the **notebook.runPython** function from Google Colab to return to the main landing page.
* The **process\_data()** function is defined as a callback function to clear the output and display the landing page when the "Back" button is clicked.
* The leaderboard HTML content is displayed using **display(HTML(html\_leaderboard))**.
* The **process\_data()** function is registered as a callback for the **notebook.runPython** event using **output.register\_callback('notebook.runPython', process\_data)**.

1. editQuestion():

* Initializes a list variable **DB** with an empty string.
* Establishes a connection to a Firebase application using **firebase.FirebaseApplication**.
* Retrieves the questions from the Firebase database using **FBconn.get()**.
* Converts the retrieved data from JSON format to a list of dictionaries and appends them to the **DB** list.
* Converts each dictionary in the **DB** list to a list and stores them in the **list\_of\_lists** list.
* Constructs an HTML string **html1** that represents the question editor form.
* Populates the dropdown list in the form with the available questions by iterating over the **list\_of\_lists** and dynamically creating **<option>** elements.
* Defines a JavaScript function **updateQuestionToDataBase(event)** that is called when the form is submitted. This function retrieves the values entered in the form fields and invokes a Python function using **google.colab.kernel.invokeFunction()** to update the question in the database.
* Adds event listeners to the dropdown list to update the form fields with the selected question's details whenever the selection changes.
* Defines a JavaScript function **Back()** that is called when the "Back" button is clicked. This function invokes a Python function using **google.colab.kernel.invokeFunction()** to go back to the admin page.
* Defines a Python function **process\_data()** that is called when the Python function is invoked by the JavaScript code. This function updates the question in the Firebase database based on the provided parameters.
* Displays the HTML form using **display(HTML(html1))**.
* Registers the **process\_data()** function as a callback for the Python function invocation using **output.register\_callback()**.

1. addQuestion():

* The HTML code for the form is stored in the **html** variable. It defines the structure and styling of the form for adding a question.
* The JavaScript code within the HTML sets up an event handler for the form submission. When the form is submitted, the **addQuestionToDataBase()** function is called. This function retrieves the values entered in the form fields and invokes the **notebook.runPython** function from the Colab environment, passing the form values as arguments.
* The **process\_data()** function is defined. It receives the form values as parameters, including **question**, **answer1**, **answer2**, **answer3**, **answer4**, **correct**, and **logout**. If **logout** is set to **"true"**, the function clears the output and calls the **adminPage()** function. If **logout** is **"false"**, the function prepares the data to upload to the Firebase database by creating a dictionary with the question details. The **FBconn.post()** method is then used to add the question data to the Firebase database.
* The HTML form is displayed using **display(IPython.display.HTML(html))**, showing the user interface for adding a question.
* The **output.register\_callback()** function is used to register the **process\_data()** function as a callback for the **notebook.runPython** event. This allows the JavaScript code to communicate with the Python code and pass the form values to it when the form is submitted.

1. adminPage():

* The HTML code for the administrator page is stored in the **html\_first** variable. It defines the structure and styling of the page, including a container with a title and three buttons for different tools: "Add Question," "Edit Questions," and "Logout."
* The JavaScript code within the HTML sets up event handlers for the button clicks. When a button is clicked, the corresponding **val** value is set, and the **notebook.runPython** function from the Colab environment is invoked with the **val** as an argument.
* The **process\_data()** function is defined. It receives the **val** and **logout** values as parameters. If **val** is "Logout," the function clears the output and calls the **login()** function to redirect to the login page. If **val** is "Add," the function clears the output and calls the **addQuestion()** function to redirect to the add question page. If **val** is "Edit," the function clears the output and calls the **editQuestion()** function to redirect to the edit question page.
* The HTML for the administrator page is displayed using **display(HTML(html\_first))**, showing the user interface with buttons and options.
* The **output.register\_callback()** function is used to register the **process\_data()** function as a callback for the **notebook.runPython** event. This allows the JavaScript code to communicate with the Python code and pass the **val** value to it when a button is clicked.

1. rankShop():

* The function initializes some variables and retrieves user data from a Firebase database.
* The function defines a style for the HTML page using CSS.
* The HTML page is constructed using a combination of HTML and Python variables. The page consists of a side navigation bar and a main container.
* The side navigation bar contains a logo and a list of navigation links, including "Dashboard," "Setting," "Rank Shop," "Help," and "Logout."
* The main container includes a header section and a content section.
* The header section contains a user profile image and a search bar.
* The content section displays a set of cards representing different ranks available in the shop. Each card contains the rank name, points required to purchase the rank, and an "Equip Rank" button.
* The function also includes JavaScript functions for handling button clicks. These functions invoke Python functions using **google.colab.kernel.invokeFunction()** with different parameters depending on the button clicked.
* The **process\_data1()** function is defined within the **rankShop()** function. It is registered as a callback function for the **notebook.runPython** event. This function handles the logic based on the button clicked or parameter passed.
* The function ends by displaying the HTML page using **display(HTML(htmlLandingPage))** and registering the **process\_data1()** function as a callback for the **notebook.runPython** event using **output.register\_callback('notebook.runPython', process\_data1)**.

1. help():

* The HTML code for the tutorial page is stored in the **html\_first** variable. It defines the structure and styling of the page, including a container with a title and two paragraphs explaining the game and its mechanics. There is also a "Back" button at the bottom of the page.
* The JavaScript code within the HTML sets up an event handler for the "Back" button. When clicked, it invokes the **notebook.runPython** function from the Colab environment.
* The **process\_data()** function is defined. It is called when the "Back" button is clicked. It clears the output and calls the **landingPage()** function to redirect to the landing page.
* The HTML for the tutorial page is displayed using **display(HTML(html\_first))**, showing the tutorial information and the "Back" button.
* The **output.register\_callback()** function is used to register the **process\_data()** function as a callback for the **notebook.runPython** event. This allows the JavaScript code to communicate with the Python code and trigger the **process\_data()** function when the "Back" button is clicked.

Database Architecture:

* We used Microsoft Firebase as our main database.
* Our database had 2 folders: 1) Users. 2) Questions.
* The Users folder houses the information about each user.
* The Questions folder houses the information about each question.

Here are some images that show the database structure:

1. Overall look:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence 2. Questions folder:

3. Users folder:

A picture containing text, receipt, screenshot

Description automatically generated

Another user:

A screenshot of a computer

Description automatically generated with low confidence