**Maintenance Guide**

This guide provides instructions for maintaining the codebase and ensuring its proper functioning. It covers the necessary steps to follow and the various components involved in the code.

**This code uses several libraries and modules, including:**

* ipywidgets
* firebase
* IPython
* google.colab
* matplotlib
* requests

Make sure to periodically check for updates to these libraries and install the latest versions as needed.

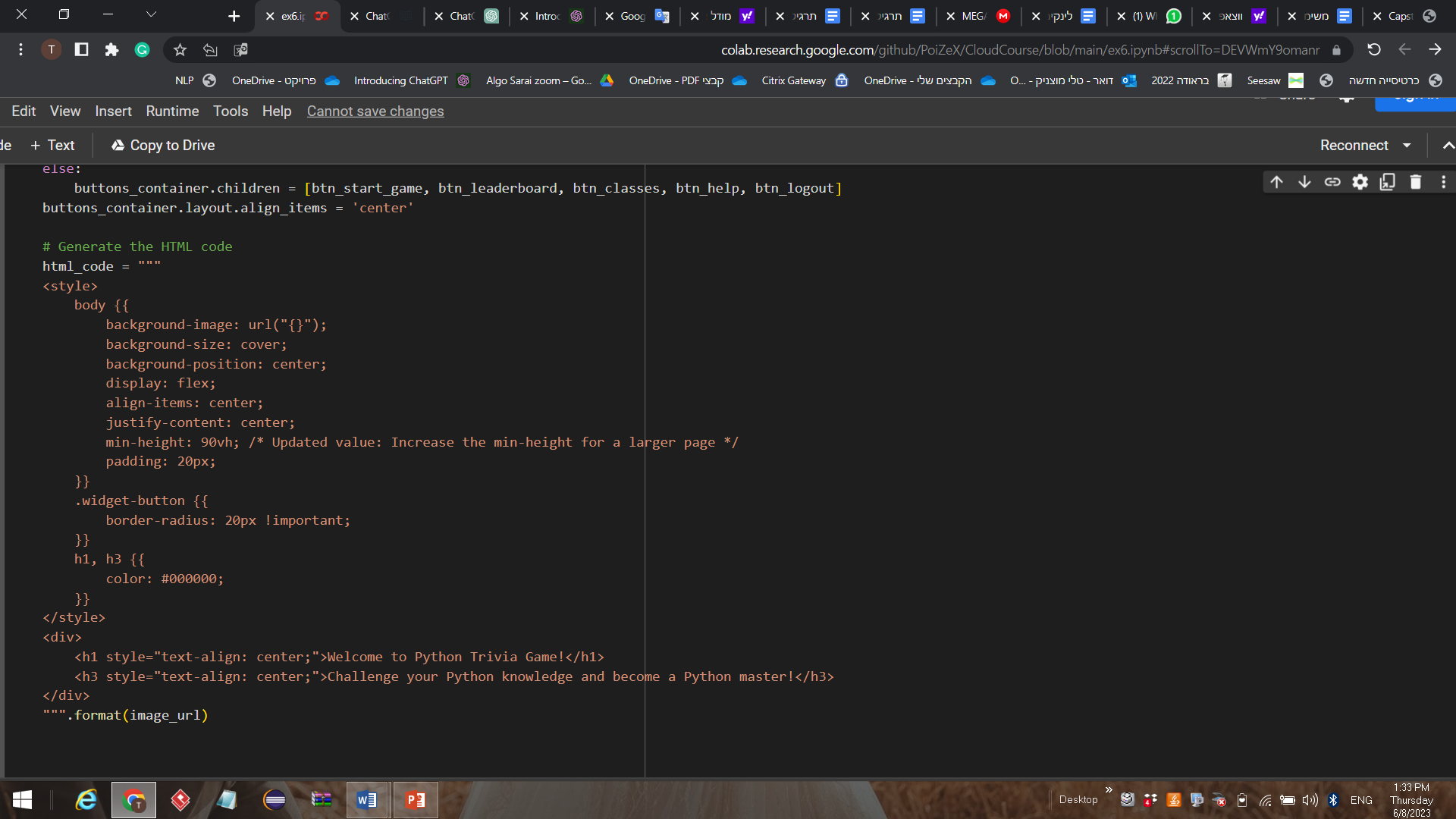
**Database environment**

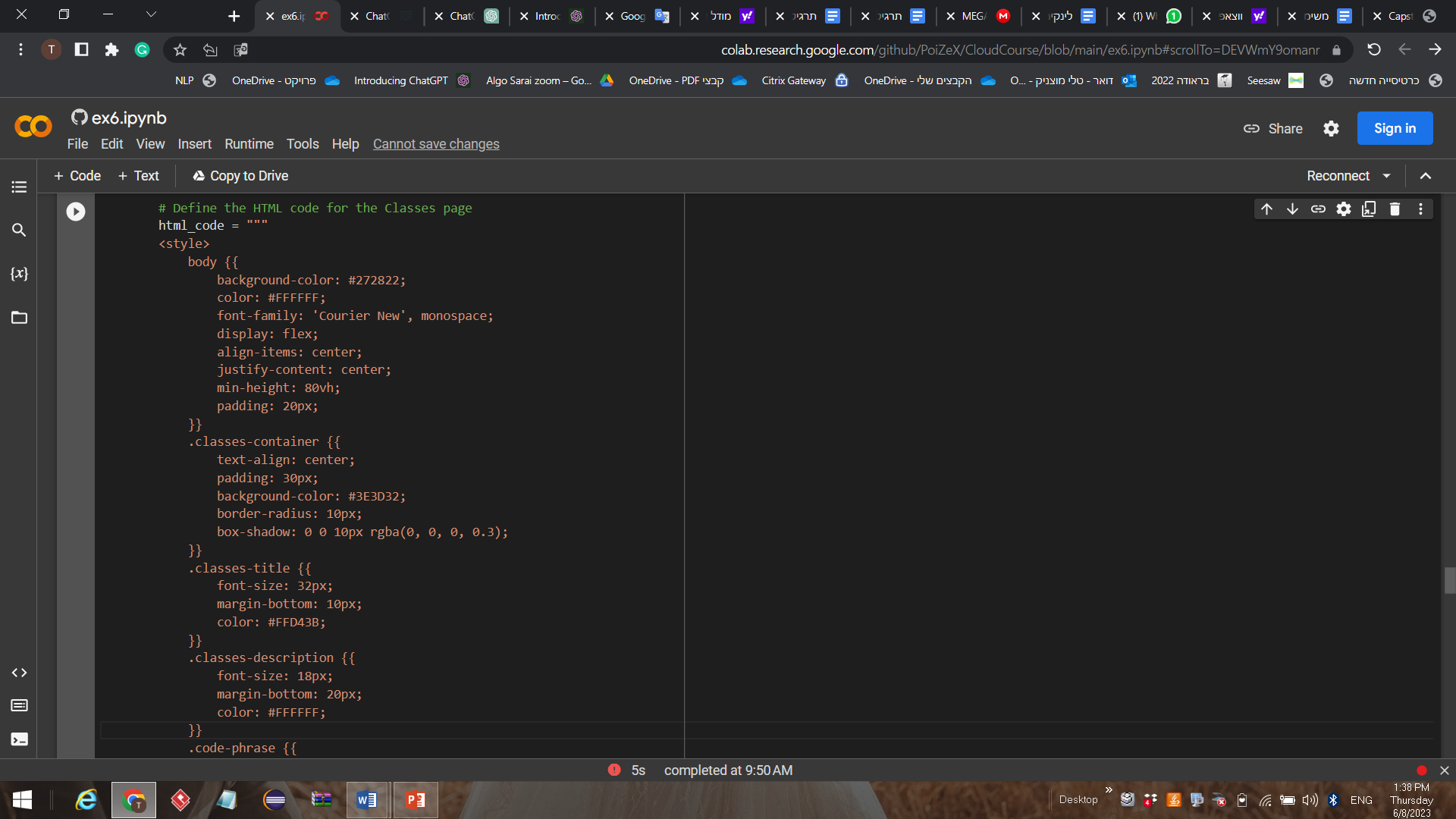
The code integrates with Firebase 4.0.1 to store and retrieve data. The firebase library is used for this purpose. Ensure that the Firebase project is properly set up and the Firebase database rules allow the required operations (read, write, etc.) as per your application's needs.

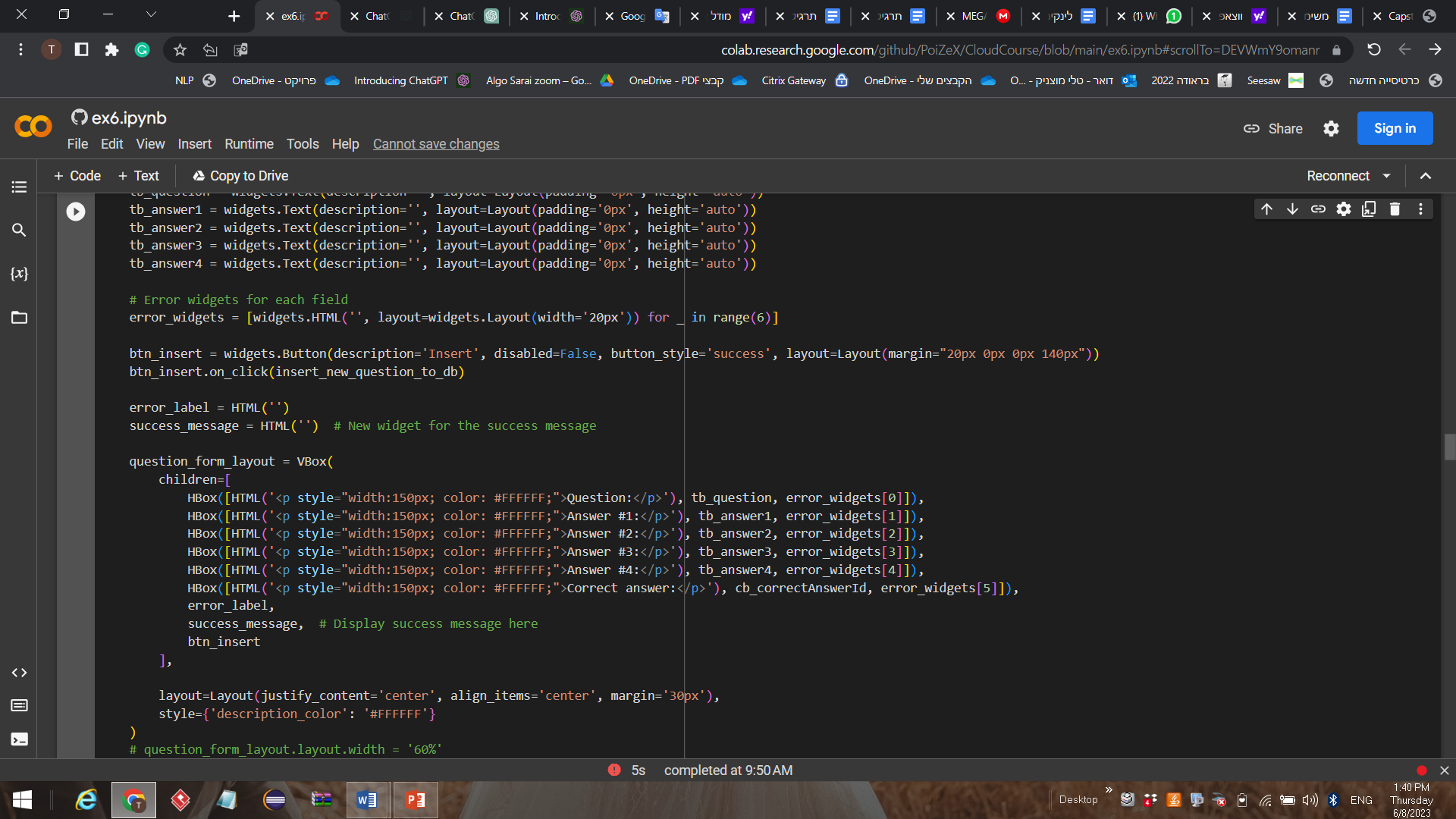
To connect for database, run this line: !pip install firebase

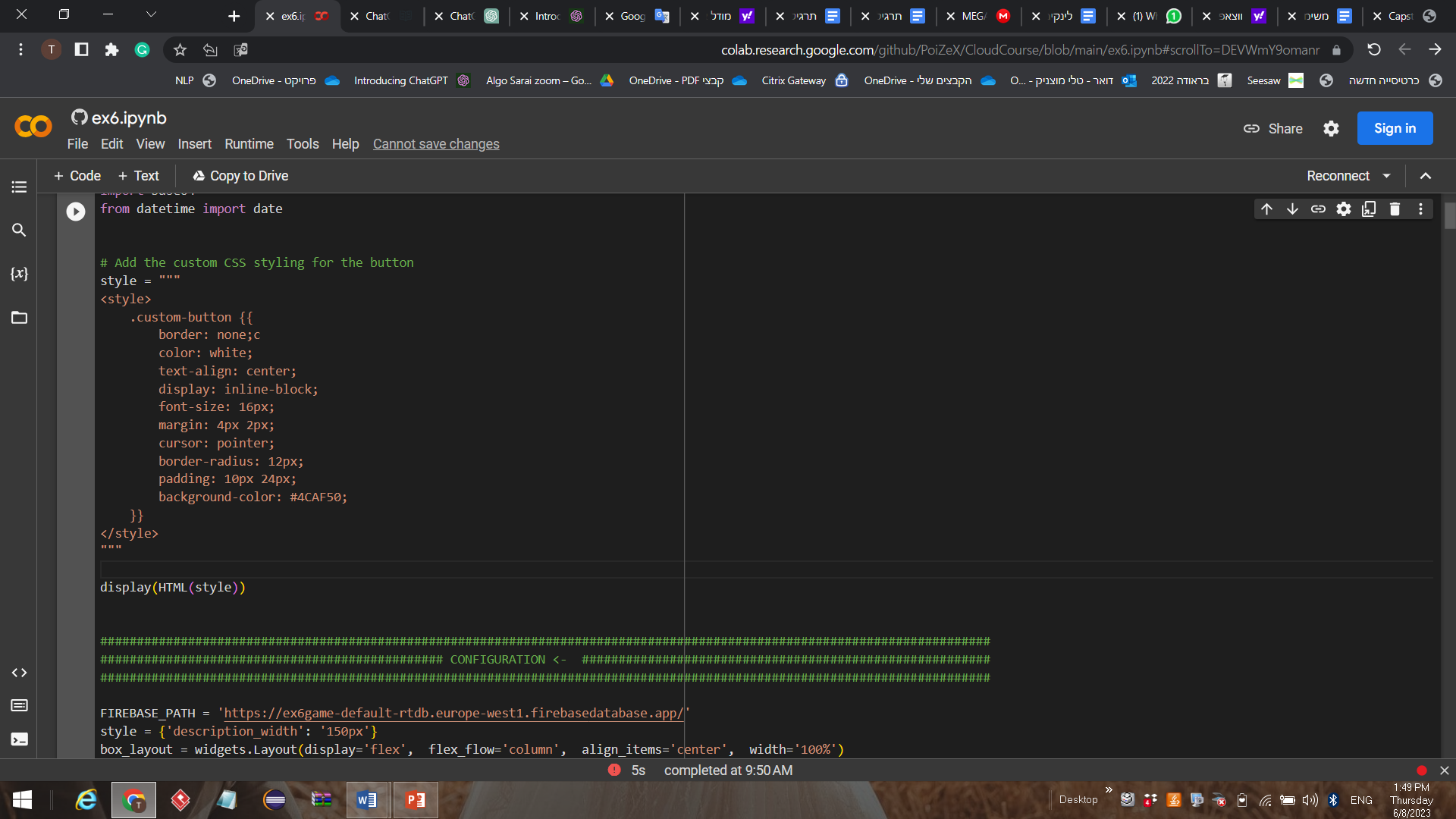
* The code uses Firebase as the database provider. Ensure that the Firebase SDK is properly installed and configured.
* Check the `FIREBASE\_PATH` variable to ensure it contains the correct path to your Firebase database.
* Verify that the `firebase` library is imported and compatible with the code.

**`firebase.FirebaseApplication`:** This class is used to establish a connection with the Firebase database and perform operations such as getting data (`get`), deleting data (`delete`), and updating data.

**User Interface:** For the UI we use HTML code style for the classes page, Widgets and CSS for button style.



 Present a form for adding new questions into DB



**Main functions:**

**user login**

1. **is\_user\_Exist(username)** function used to check if a user with a specific `username` already exists in the database. This can be useful when someone is trying to register a new account to ensure that the chosen username is unique.
2. **validate\_login(username, password)** function used to validate the login information provided by a user.If a match is found, the function sets the `current\_user` variable, indicating that the user is now logged in .If no match is found, an error message will be displayed to the user.
3. **register\_page(b=None)** function used to display a registration page to the user. This page typically includes text input fields for the desired `username`, `password`, and `confirm password`. The user can enter their chosen credentials, and when they click the registration button, the `sign\_up` function is called to create a new user account with the provided information.

**Design Patterns:**

**Singleton Pattern**: The User class implementing the Singleton pattern. It ensures that only one instance of the class can be created, which is relevant in the context of representing the currently logged-in user.

**For the menu page:**

**show\_menu\_page()** function creates the main menu of the Game, defines button callbacks for each menu option, and utilizes HTML and widgets to present a visually appealing and interactive user interface.

**Design Patterns:MVC**: The HTML code represents the view, the button callbacks handle the controller logic, and the underlying data structures and functions represent the model.

**generate\_leaderboard()** function retrieves the scores from a Firebase database and generates an HTML table to display the leaderboard. The function starts by establishing a connection to the Firebase database using the FirebaseApplication method. To display the results of all users who played the game

**Design Patterns:**

**Singleton Pattern**: The connection to the Firebase database is established using the firebase.**FirebaseApplication** method. This method ensures that only one instance of the Firebase application is created and reused throughout the code.

**Function for admin edit page**

**AdminEditPage()** function is responsible for displaying the admin edit page. it iterates over each

question in the list of questions retrieved from the database and creates a button for each question.

**Design Patterns:MVC-** AdminEditPage acts as the controller, handling user interactions and updating the view. Event handlers (handle\_button\_click, handle\_edit\_click, handle\_remove\_click ו-handle\_save\_click) modify the data model (questions in the database) and update the view components accordingly.

**insert\_new\_question\_to\_db:** Insert a new question into the database. It retrieves the values entered in the form fields

**show\_add\_questions\_page:** Display a form for adding new questions to the database, Create the form layout using `VBox` and `HBox` containers **.**

**Function to show the results page to the user**

**show\_user\_statistics(combined\_list)** function: Generates and displays a pie chart that represents the distribution of the total number of attempts for each game. This function uses the matplotlib.pyplot library to create the pie chart

Parameters: combined\_list: A list containing the number of attempts for each game.

**show\_results\_page()** function: Displays the results page after the game is over.This function generates HTML content and uses the display() function to show the content.

**Saving points:**

**incrementPointsToUser** **(username, points\_to\_increment)**a function that increments the total points of a user in a Firebase database based on their username, and updates the total points in the database using FBconn.put. It compares the usernames one by one until it finds a match. Once a match is found, it retrieves the user's information

**get\_user\_stats(username)** used to retrieve the statistics of a user from a Firebase database looks for the user's scores in the database and extracts the statistics associated with those scores.

**update\_points(new\_points, display\_points=True):** This function updates the points variable of user based on his answer. It takes new\_points as a parameter, representing the points to be added.

**The question logic:**

**on\_answer\_button\_clicked** function handles the logic for processing user interactions with answer buttons in a game. It updates statistics, evaluates correctness, updates points by checking if it is his first try (receives the max points), and manages the display.

**show\_success\_dialog** the func changes the button style of the continue\_button global variable to 'success', and presents a success dialog by clearing any previous errors, updating the style of the continue button, and enabling it for further interaction.

**display\_questions** function sets up and displays the elements of the questions page in a GUI. It handles the presentation of the question, points label, progress bar, answer buttons, and error output.

**show\_questions\_page** function is responsible for generating and displaying a new question on the questions page. The function calls the get\_random\_question method of the questionsFromDB object to retrieve a random question from the database.

**Main classes:**

1.**WidgetEnum** enumeration defines different types of widgets. 2.**User class** represents a user in the system, with attributes for username, password, role, and total points. It provides methods to check if the user is an admin, retrieve the username, and get the total points. 3.**Score** class represents a game score, with attributes for the date, username, statistics list, and points earned. It provides methods to retrieve the statistics list and game points. 4.**Question** class represents a single question in the quiz game and provides methods for accessing and manipulating question data. 5.**QuestionsDB** class manages the collection of questions, including retrieving random questions, adding questions, and maintaining a local list of questions. 6. **StatsDB** represents a statistics database and provides methods to retrieve statistics based on user ID and date range.

**Conclusions** Throughout the development phase we faced many challenges. Mainly in:

* **Performance Optimization:** As the game grows and handles a larger number of questions, users, and interactions, optimizing performance becomes crucial. Ensuring smooth gameplay, minimizing loading times, and managing memory usage was challenging and may require optimizing database queries, and implementing caching mechanisms.
* **Game Logic**: Implementing the game's logic, including tracking scores, handling user input, and managing game flow, was little complex. Ensuring the game functions correctly by thinking about interesting graphs for user and combine attractive design interface was challenging **In conclusion, we think we have created an interesting and attractive game. Despite the challenges, we are happy to achieved our goal**