External Documentation

Contents

Database Classes

The database services contain the classes that specifically have to do with the database.

It contains the auth.dart, database_serivce.dart, database.dart and mock_database.dart classes.

Auth class:

The AuthSerrvice class handles the authentication using Firebase in a Flutter application. It creates an instance of the FirebaseAuth which is used to interact with the Firebase authentication service.

It contains the loginUsingEmailPassword method. This takes in two parameters – String email and String password. It returns a Future<User?>

It also has a signOut method. This method returns await_auth.signOut() if the sign out is successful and null if it is not successful.

Database_service class:

This class contains several abstract method declarations for interacting with the database.

These are the methods that the class contains:

1. **getUser()**: Returns a **Future<String?>** representing the user.
2. **addSignupToFirestore(String email, password, username, name, DateTime givendate)**: Adds signup data to the Firestore database. It takes the user's String email, password, username, name, and a **DateTime** object representing the given date.
3. **addUpdatedScore(String quizSelected, int _currentIndex, questionlength)**: Updates the user's score for a particular quiz. It takes a String quiz selected, an integer current index, and an integer length of the questions.
4. **getMCQQuestionsAnswers(String x, List<String> _questions)**: Retrieves multiple-choice questions and their answers from the database. It takes a String **x** and a list of questions **_questions**. It returns a **Future<List<Map<String, dynamic>>>** containing the questions and answers.
5. **getShortQuestionsAnswers(String x, List<String> _questions, bool isShuffled)**: Retrieves short answer questions and their answers from the database. It takes a String **x**, a list of questions **_questions**, and a Boolean **isShuffled**. It returns a **Future<List<Map<String, dynamic>>>** containing the questions and answers.
6. **getMAQQuestionsAnswers(String x, List<String> _questions)**: Retrieves multiple-answer questions and their answers from the database. It takes a String **x** and a list of questions **_questions**. It returns a **Future<List<Map<String, dynamic>>>** containing the questions and answers.
7. **addDataToCreateaQuizFirestore(String getQuizName, getQuizType, getQuizDescription, getQuizCategory, String imageURL)**: Adds quiz-related data to the Firestore database. It takes parameters for the quiz name, quiz type, quiz description, quiz category, and an image URL.

8. **addNumberOfQuestions(String quizID, int numQuestions, bool isTimed, int time)**: Adds the number of questions for a quiz to the database. It takes the quiz ID, the number of questions, a boolean indicating whether the quiz is timed, and the time limit in seconds.
9. **_getQuizID()**: Returns a **Future<String>** representing the quiz ID.
10. **addImagesToFirestore(String question, Image1url, image2url, image3url, image4url, image5url, image6url, int index)**: Adds image URLs associated with a question to the Firestore database. It takes the question, six image URLs, and an index.
11. **resetPassword(String email)**: Sends a password reset email to the provided email address.
12. **getQuizInformation(String x)**: Retrieves information about a quiz from the database. It takes a parameter **x** (possibly a quiz ID) and returns a **Future<List<Map<String, dynamic>>>** containing the quiz information.
13. **getQuizID()**: Returns a **Future<String>** representing the quiz ID.
14. **updateQuizzesStattus()**: Updates the status of quizzes in the database.
15. **PublishDataToFirestore(int index, int Quiztype, List<String> questions, List<String> answers)**: Publishes data related to a quiz to the Firestore database. It takes an index, quiz type, a list of questions, and a list of answers.

Database class:

This is a service class that is responsible for interacting with the Firebase database and the Firebase Authentication. Overall, this class has different methods that interact with the Firebase database and Firebase Authentication for tasks such as creating users, managing quizzes, retrieving data, and updating records.

1. There's a method **updateQuizzesStattus()** that updates the status of a quiz to "Finished" in the Firestore database.
2. The method **PublishDataToFirestore()** adds data to the Firestore collection "Questions" based on the quiz type (MCQ or Short Answer) and the provided questions and answers.
3. The method **getQuizID()** retrieves the quiz ID for the currently logged-in user based on their username and the most recent quiz created by that user.
4. The method **getUser()** retrieves the username of the current user from the Firestore database.
5. The method **addSignupToFirestore()** creates a new user in Firebase Authentication and adds user-related data to the Firestore collection "Users".
6. The method **addUpdatedScore()** adds the updated score of a user for a specific quiz to the Firestore collection "QuizResults".
7. The methods **getMCQQuestionsAnswers()**, **getShortQuestionsAnswers()**, and **getMAQQuestionsAnswers()** retrieve questions and answers from the Firestore collection "Questions" based on the provided quiz ID and question type.
8. The method **addDataToCreateaQuizFirestore()** adds data for a new quiz created by a user to the Firestore collection "Quizzes".
9. The method **addNumberOfQuestions()** updates the number of questions and other properties for a specific quiz in the Firestore collection "Quizzes".
10. The method **addImagesToFirestore()** adds image-related data for a question to the Firestore collection "Questions".
11. The method **resetPassword()** sends a password reset email to the provided email address using Firebase Authentication.

12. The method **getQuizInformation()** retrieves information about quizzes from the Firestore collection "Quizzes" based on the provided quiz category or all quizzes if "All" is specified.
13. The method **getQuizName()** retrieves the name of a quiz based on the provided quiz ID.

Mock_database class:

This class creates a mock database that is used for unit testing.

1. **getUser**: This method retrieves the username from the Firestore collection "Users" and returns it as a **Future<String?>**. It sets the **username** variable with the retrieved value and returns it.
2. **getUserID**: This method retrieves the quiz ID from the Firestore collection "Quizzes" and returns it as a **Future<String?>**. It sets the **userID** variable with the retrieved value and returns it.
3. **addUpdatedScore**: This method adds quiz result data to the Firestore collection "QuizResults". It creates a new document with a unique ID, sets the quiz result data, retrieves the data from the created document, and returns it as a **Map<String, dynamic>**.
4. **updateQuizzesStattus**: This method updates the status of a quiz document in the Firestore collection "Quizzes" to "Finished". It sets the **Status** field of the document to "Finished" and retrieves the updated status as a **String**.
5. **addSignupToFirestore**: This method adds user data to the Firestore collection "Users". It creates a new document with a specific ID, sets the user data, and retrieves the data from the created document as a **Map<String, dynamic>**.
6. **getQuizInformation1**: This method retrieves quiz information from the Firestore collection "Quizzes" based on a given category (**x**). It creates a new document with a unique ID, sets the quiz information, and returns it as a **Map<String, dynamic>**.
7. **getQuizName1**: This method retrieves the quiz name from the Firestore collection "Quizzes" and returns it as a **Future<String?>**. It sets the **QuizName** variable with the retrieved value and returns it.

Answer a Quiz Classes

The next few classes are part of the Answer a Quiz section. These classes contain the code for a user to answer a specific type of quiz. The classes are the answerMAQ.dart, answerMCQ.dart, answerShortAnswer.dart.

answerMAQ (Multiple Answer Questions) class:

1. The class has several properties, including **quizID**, **bTimed**, and **iTime**, which are passed as parameters to the constructor.
2. The **initState** method is used to initialize the necessary variables and set up the timer if the quiz is timed.
3. The **dispose** method is overridden to cancel the timer when the widget is disposed.
4. The **getScore** method calculates the score based on the number of correct answers.
5. The **_submitAnswer** method is called when the user submits the answers. It updates the score, sets the **isSubmited** flag to true, and cancels the timer.
6. The **getQuestionsAnswers** method is an asynchronous function that retrieves the quiz questions and answers from the database.
7. The **_buildTimerWidget** method returns a **Text** widget displaying the remaining time if the quiz is timed.
8. The **validateAnswer** method is used for input validation. It checks for empty answers, duplicate answers, and incorrect answers.
9. The **_showAlertDialog** method displays a dialog asking the user if they want to submit the quiz when it's not complete.
10. The **build** method builds the UI for the answer submission page. This will also show the user the possible correct answers once the quiz has been submitted.
11. The **showRating** function displays a dialog box for rating the quiz. It shows a dialog using **showDialog** with a custom **RatingDialog** as the content of the dialog which allows the user to provide a rating for the quiz. When the user submits the rating, the **onSubmitted** callback is called where the **rating** variable is updated with the submitted rating. The **addOrUpdateQuizRating** function is then called to add or update the quiz rating in the Firestore database. Finally, it navigates back to the **MenuPage**. If the user cancels the rating dialog, the **onCancelled** callback is called. It simply navigates back to the **MenuPage** without submitting a rating.
12. The **addOrUpdateQuizRating** function handles adding or updating the quiz rating in the Firestore database.

answerMCQ (multiple Choice Question) class:

1. The class has several instance variables, including **answerControllers** (a list for handling user input), **isSubmited** and **isCorrect** (boolean flags for tracking submission and correctness of answers), **_currentIndex** (the index of the current question), and **quizSelected** (the ID of the selected quiz) as well as some related to timed quizzes, such as **isTimed** (a boolean flag indicating whether the quiz is timed) and **time** (the duration of the quiz in seconds).

2. In the **initState** method, the necessary initialization is performed, including setting up the timer if the quiz is timed.
3. The **dispose** method is overridden to cancel the timer when the widget is disposed.
4. The **getScore** method calculates the user's score based on the number of correct answers.
5. The **_submitAnswer** method is called when the user submits their answers. It saves the user's answers, displays a dialog with the score, and updates the score in the database.
6. The **_goToPreviousQuestion** method allows the user to navigate to the previous question and re-answer it.
7. The **_goToNextQuestion** method loads the next question and clears the previous answer.
8. The **getQuestionsAnswers** method fetches the quiz questions and answers from the database.
9. The **shuffleOptions** method shuffles the answer options for each question.
10. The **_buildTimerWidget** method creates a widget to display the remaining time if the quiz is timed.
13. The **build** method constructs the widget tree for the quiz answer page and builds the UI for the answer submission page. This will also show the user the possible correct answers once the quiz has been submitted.
14. The **showRating** function displays a dialog box for rating the quiz. It shows a dialog using **showDialog** with a custom **RatingDialog** as the content of the dialog which allows the user to provide a rating for the quiz. When the user submits the rating, the **onSubmitted** callback is called where the **rating** variable is updated with the submitted rating. The **addOrUpdateQuizRating** function is then called to add or update the quiz rating in the Firestore database. Finally, it navigates back to the **MenuPage**. If the user cancels the rating dialog, the **onCancelled** callback is called. It simply navigates back to the **MenuPage** without submitting a rating.
15. The **addOrUpdateQuizRating** function handles adding or updating the quiz rating in the Firestore database.

answerShortAnswer class:

1. The class has several instance variables, including **_currentIndex** to keep track of the current question index, **quizSelected** to store the selected quiz ID, **answerControllers** to manage the text editing controllers for user answers, and **isSubmited** and **isCorrect** flags to track the submission status and correctness of answers.
2. The **initState** method is used to initialize the necessary variables when the page loads, including setting up a timer if the quiz is timed.
3. The **dispose** method is overridden to cancel the timer and clean up resources when the page is disposed.
4. The class defines lists to store the quiz questions, correct answers, and user answers. It also defines a variable **count** to keep track of the number of correct answers.
5. The **getScore** method calculates and returns the user's score based on the number of correct answers.
6. The **_submitAnswer** method is called when the user submits the quiz. It saves the user's answer, calculates the score, shows a dialog with the score, and updates the score in the database.

7. The **_goToPreviousQuestion** method allows the user to go back to the previous question and re-answer it.
8. The **_goToNextQuestion** method loads the next question and clears the previous answer.
9. The **getQuestionsAnswers** method retrieves the quiz questions and answers from the database.
10. The **_buildTimerWidget** method returns a widget to display the remaining time if the quiz is timed.
11. The **build** method builds the UI of the page using various Flutter widgets. It displays the current question, a text field for the user's answer, and buttons for navigating between questions. It also shows the timer widget if the quiz is timed. This will also show the user the possible correct answers once the quiz has been submitted.
12. The **showRating** function displays a dialog box for rating the quiz. It shows a dialog using **showDialog** with a custom **RatingDialog** as the content of the dialog which allows the user to provide a rating for the quiz. When the user submits the rating, the **onSubmitted** callback is called where the **rating** variable is updated with the submitted rating. The **addOrUpdateQuizRating** function is then called to add or update the quiz rating in the Firestore database. Finally, it navigates back to the **MenuPage**. If the user cancels the rating dialog, the **onCancelled** callback is called. It simply navigates back to the **MenuPage** without submitting a rating.
13. The **addOrUpdateQuizRating** function handles adding or updating the quiz rating in the Firestore database.

Create a Quiz Classes

The next few classes are part of the Create a Quiz section. These classes contain the code for a user to create a specific type of quiz. The user also has the option to let ChatGPT create questions for them once they have chosen their category and number of questions. The publish page is shown when a user clicks “done” once they have entered their questions and answers. The classes are the create_Quiz.dart, createMAQ.dart, createMCQ.dart, createShortAnswer.dart, publishPage.dart.

create_quiz class:

1. The code defines a function called **sendChatGPTRequest** to send a request to the ChatGPT API for generating responses based on user messages. It takes a **message** parameter as input. The function uses the **http** package to make a POST request to the API endpoint. The API key is included in the request headers for authentication. The function returns a **Future** object containing the API response as a **Map** of key-value pairs.
2. The **generateQuizQuestions** function is defined to generate quiz questions and answers using the ChatGPT API. It takes parameters such as **quizName**, **quizDescription**, **quizType**, and **NumofQuestions**. The function sets the number of questions to 5 by default. If the quiz type is "Short-Answer," the function calls the **sendChatGPTRequest** function to generate questions and answers in a specific format. The API response is processed to extract the generated questions and answers. The questions and answers are split based on the format and stored in separate lists (**questions** and **answers**). The generated quiz is stored as a list of maps, where each map contains a question and its corresponding answer. If the quiz type is "Multiple Choice Questions," a similar process is followed with a different request format. The function splits the response into questions and answers and stores them accordingly. For multiple-choice questions, the answers are split into individual options. The generated quiz is stored as a list of maps, where each map contains a question, its answer options, and the correct answer. If the quiz type is "Multiple Answer," the function follows a similar process with a different request format. The response is split into questions and answers, and the answers are stored as separate options. The generated quiz is stored as a list of maps, where each map contains a question and its corresponding answer options. The function returns a **Map** object containing the generated quiz, questions, and answers.
3. The **selectFile** function is used to select an image file from the device and upload it to Firebase Storage.
4. The **getUser** function retrieves the current user's information from Firestore.
5. The **showImageSelectedDialog** function displays a dialog to inform the user that an image has been selected.
6. The **addDataToFirestore** function adds quiz data to Firestore using the **DatabaseService** class.
7. The **getQuizType** and **getQuizCategory** functions retrieve the selected quiz type and category.
8. The **_submit** function is called when the form is submitted. It validates the form and then adds the quiz data to Firestore.
9. The **build** method defines the UI layout using various Flutter widgets.
10. The **validateName** and **validateDescription** functions are used to validate the input fields.
11. The **_showDialog** function displays a dialog with a given message.

createMAQ (Multiple Answer Question) class:

1. The **CreateMAQ** class is defined as a **StatefulWidget** that represents the screen for creating a Multiple Answer Quiz.
2. There are several **TextEditingController** objects created for handling user input for the question, expected number of answers, and possible answers.
3. The **validateQuestion**, **validateAnswer** and **validateExpected** methods are all validations methods that check if the value is empty or not. If it is empty, it returns a specific message prompting the user otherwise it returns null.
4. The **SingleChildScrollView** widget allows scrolling when the content exceeds the screen height.
5. The **Form** widget is used to encapsulate the form fields and handle validation. The form consists of several widgets, including a cantered text widget for the quiz title, a **TextFormField** for the question, and a **ListView.builder** for displaying and managing the possible answers.
6. The **TextFormField** widgets are used for capturing user input, and each has its own validation function such as prompting the user to enter anther potential answer.
7. The "Add Another Potential Answer" button allows the user to add more possible answer text boxes dynamically by tapping on it.
8. The "Done" button validates the form and, if valid, retrieves the entered question, answers, and quiz type, and navigates to the **publishPage**.

createMCQ (Multiple Choice Question) class:

1. The state class contains several lists and variables to manage the questions and their corresponding answers.
2. The **loadExisting** method is used to load an existing question from the list based on the current question index.
3. The **convertLists** method converts the question-and-answer lists into a format that can be passed to another widget.
4. The **_nextQuestion** method is called when the user wants to move to the next question. It performs validation checks and either loads an existing question or adds a new question to the list.
5. The **build** method constructs the UI for the multiple-choice question page. It uses various **TextFormField** widgets to capture the question-and-answer options from the user. The **ElevatedButton** widgets are used for navigation and submission.
6. The **validateQuestion** and **validateAnswer** methods are used for form validation that check if the value is empty or not. If it is empty, it returns a specific message prompting the user otherwise it returns null.
7. The **Question** class represents a single multiple-choice question and its options.

createShortAnswer class:

1. The widget maintains state variables like **currentQuestionIndex**, **questions**, and **answers** to keep track of the current question index and store the entered questions and answers.

2. It defines two lists of **TextEditingController** instances: **questionControllers** and **answerControllers**. These controllers are used to retrieve the values entered in the question-and-answer text fields.
3. The **loadExisting** function is responsible for loading existing question details if they were already inputted by the user. It sets the text of the controllers and adds listeners to update the respective question and answer lists when the user changes the details.
4. The **_nextQuestion** function is called when the user wants to proceed to the next question. It performs validation using the **_formKey** and adds the question and answer to the respective lists based on the current question index. It also clears the text fields for the next question. If the user is navigating back to a previous question, it loads the existing details for that question.
5. The **build** method constructs the UI of the widget using various Flutter widgets. It includes a form with a text field for the question and another for the answer. Buttons for navigating to the previous question, next question, and completing the quiz are also present.
6. The code defines validation functions **validateQuestion** and **validateAnswer** to ensure that the question-and-answer fields are not empty. It checks if the value is empty or not. If it is empty, it returns a specific message prompting the user otherwise it returns null.

publishPage class:

This page will appear after a user completed one of the above answer a quiz pages, they will click the "Publish" button which will take them to the publish page where they will review the quiz they have created as well as decide if they would like the quiz to be timed, have pre-requisites, and/or convert it to another language.

1. The **publishPage** class is defined as a **StatefulWidget** that represents the page for publishing a quiz. It receives the quiz details (questions, answers, quiz type) as constructor arguments.
2. The state class initializes some variables and controllers, such as **_formKey**, **_QuizDetails**, **_QuizName**, **_QuizID**, **isTimed, sLang**, **hasPrerequisites**, **timeLimit**, **ID**, **quizID**, and **timeLimitController**. It also overrides the **initState** method to set up the page when it loads.
3. The **_showDialog** method displays a dialog with a message to the user.
4. In the **sendChatGPTRequest** function, the **'Accept': 'application/json'** header is added to specify the expected response format. The response body is decoded using **utf8.decode(response.bodyBytes)** to correctly decode the response using UTF-8 encoding.
5. The **translatequizquestions** function is introduced to translate quiz questions and answers to the specified language. It takes a **Language** parameter as input. The function prepares the list of questions and answers from **_questions** and **_ans**. The **sendChatGPTRequest** function is called with a modified prompt. The prompt instructs the model to translate the questions and answers into the specified language. The API response is processed to extract the translated questions and answers. The extracted sections are split and stored as **translatedQuestions** and **translatedAnswers**, respectively. A map named **translatedQuiz** is created to store the translated questions and answers. The **_questions** and **_ans** state variables are updated with the translated values using **setState**. The function returns the **translatedQuiz** map containing the translated questions and answers.
6. The **getQuizInformation** method retrieves quiz information from the database using a **DatabaseService** object.

7. The **addDataToFirestore** method adds quiz data to Firestore based on the quiz type (1 for short answer, 2 for MCQ, 3 for multiple answers).
8. The **mcqDisplay** method formats the multiple-choice answers for display.
9. The **maqDisplay** method formats the multiple-answer answers for display.
10. The **addExtraDetails** method adds extra details to the quiz in Firestore, such as the number of questions, time limit, prerequisites.
11. The **_publish** method is called when the user clicks the "Publish quiz" button. It validates the form, retrieves quiz information, sets time limit if applicable, checks for prerequisites, adds quiz details to Firestore, and updates the quiz status. It also displays a dialog to inform the user about the quiz creation and navigates back to the menu page.
12. The **build** method builds the UI of the page using various Flutter widgets. It displays the questions and answers, checkboxes for timed and prerequisite quizzes, and buttons for editing questions and publishing the quiz.
13. The **validateTime** method is a validator for the time limit field, checking if the entered time is in the correct format and returns the necessary message to prompt users to enter the time correctly.

Forgot Password Class

The forgot password class includes a form where users will get an email and will be taken to a link where they can change/reset their password if they have forgotten it. The class is the forgotpassword.dart.

1. Inside the **_ForgotPasswordPageState** class, the necessary variables are declared: **_formKey**: A **GlobalKey<FormState>** used to identify and validate the form, **_emailController**: A **TextEditingController** used to control the email text field and **service**: An instance of the **DatabaseService** class.
2. The **dispose** method is overridden to dispose of the **_emailController** when the widget is disposed.
3. The **build** method is used to build the UI for the **ForgotPasswordPage** widget. The UI includes an **AppBar** with a back button, a form with an email text field, and a "Reset Password" button.
4. The **validateEmail** method is defined to validate the email address entered by the user. It uses a regular expression to check if the email format is valid. If it is valid it returns null otherwise it returns a message to prompt the user to enter the correct email address.
5. The **resetPassword** method is defined to handle the logic for resetting the password. If the password reset is successful, a dialog is shown with a success message. Otherwise, an error message is shown.
6. The **_showDialog** method is defined to show an alert dialog with a given message.

Login Class

The Login class is used to login a user who has already created an account. It gets the users email and password and checks it against what is stored in the database and will successfully login the user if the credentials are correct, otherwise it will provide error messages to prompt the user to enter the correct details. It also contains a “Forgot Password” button that will allow the user to reset their password (that class is explained above). It also has a “Sign Up” button that allows the user to navigate to the Sign-Up page if they do not have an account (will be explained).

1. Inside the **LoginPageState** class, the necessary variables are declared: **_formKey**: A **GlobalKey<FormState>** used to identify and validate the form, **usernameController** and **passwordController**: **TextEditingController** used to control the username and password text fields, **passwordVisible**: A boolean variable to toggle the visibility of the password, **user**: A **User** object representing the logged-in user (initialized to **null**), **service**: An instance of the **DatabaseService** class.
2. The **clearInputs** method is defined to clear the values of **usernameController** and **passwordController**.
3. The **validateAndSave** method is defined to validate the form inputs and perform the login operation. If the form is valid, it calls the **setUserID** method from **service** and clears the input fields. Then it navigates to the **MenuPage**, which is the main home page.
4. The **build** method is used to build the UI for the **LoginPage** widget. The UI includes an **AppBar** with a logo, a search field, a "Home" button, a "Sign Up" button, and a form with email and password text fields, a "Forgot Password?" button, a login button, and a "Don't have an account? Sign up" button.

Quiz Stats Class

The Quiz Stats page is used as a summary page for the user to see how other people have done on quizzes that they have created. It shows the average score, the minimum score and the maximum score for each quiz a user has created.

1. The **QuizStatsPage** class displays statistics for quizzes.
2. The **DatabaseService** class is instantiated for accessing database methods.
3. The **getAllUniqueQuizIds** function retrieves unique quiz IDs and corresponding data from Firestore. It loops through the query snapshot to extract the quiz ID, scores, total answers, and date completed.
4. The **getMinScore** function calculates the minimum score and returns it, based on the quiz ID.
5. The **getMaxScore** function calculates the maximum score and returns it, based on the quiz ID.
6. The **getAverageScore** function calculates the average score and returns it, based on the quiz ID.
7. The **build** method constructs the UI for the quiz statistics page.
8. A **FutureBuilder** is used to asynchronously fetch unique quiz IDs and display the statistics. If an error occurs during future execution, an error message is shown. Once the future is completed, the quiz names are sorted alphabetically.
9. The quiz name, average score, max score, min score, and average quiz ranking are displayed in a column layout.

Sign-Up Class

The Sign-Up page is for new users who have not yet created an account and want to do so. They will be asked to enter personal details and once they have successfully signed in, they will be taken to the main home page.

1. Widget build method: This method builds the UI of the signup page using Flutter widgets. It includes a **Scaffold** widget as the root, an **AppBar** for the top bar, and a **body** that contains the signup form.
2. Signup form: The form includes text input fields for username, name, email, password, and confirm password. It also includes a dropdown button for selecting the users date of birth.
3. Validation methods: There are several validation methods defined for validating user inputs, such as **validateEmail**, **validateUsername**, **validateName, validateConfrim, validateMonth, validateDay,** and **validateYear** . These methods are used in the form validation process. They return null if what the user has entered is correct otherwise, they return a message that will prompt the user to enter the correct details for that specified field.
4. Submit method: The **_submit** method is triggered when the signup button is pressed. It performs form validation and, if successful, adds the user to the database shows a success message and navigates to the main home page.
5. Other methods: There are additional methods like **clearInputs** for clearing the input fields, **getDate** for getting the selected date of birth, and **showDialog1** for displaying a dialog with a message.

Landing Page Class

Landing page (or SelectaPage) – this page is the first page the user sees when they view the website. It shows the user the quizzes that are currently trending and gives them the options of logging in and signing up. Users can browse different quizzes but unless they have logged in, they will not be allowed to attempt them. Once a user successfully logs in, they will be taken back to this same page but now they will have more options (such as viewing quiz stats and ranking) and they can now attempt quizzes and create them.

1. Inside the **SelectaPage** class, there are several lists and variables to store quiz information.
2. The **_selectedFilter** variable holds the currently selected filter for quizzes.
3. The **getQuizInformation** method retrieves quiz information from a Firestore database based on the selected filter.
4. The **handleRandomQuizButtonPress** function enters a loop to generate a random index until a quiz with **_QuizPrereq[randomQuizIndex] = "none"** is found. This ensures that only quizzes without prerequisites are selected. Inside the loop, a random index is generated using **Random().nextInt(_Quiz_ID.length)**. If the **_Quiz_ID** list is not empty and the generated index satisfies the prerequisite condition, the loop exits. After the loop, an **AlertDialog** is displayed to show the information (quiz title, an image related to the quiz, category, type, and the number of questions) about the randomly selected quiz. Two actions are provided in the dialog: "Start Quiz" and "Cancel". If the user selects "Start Quiz", the corresponding quiz type is checked, and the appropriate quiz page is navigated to. For example, if the quiz type is "Short-Answer", the **ShortQuizAnswer** page is pushed to the navigator stack. If the user selects "Cancel", the dialog is dismissed.The second dialog also provides an "OK" button to dismiss the dialog.
5. The **build** method builds the UI of the page using **Scaffold** and other Flutter widgets. The UI includes an app bar with a logo, search bar, and navigation buttons for home, creating a quiz, login, and sign-up.
6. The **body** section displays the trending quizzes using a **ListView** and a **CarouselSlider.builder** widget. Each quiz is displayed as a card with its details such as name, category, type, and number of questions.
7. Users can start a quiz by tapping the "Start Quiz" button, if they have logged in they can attempt the quiz otherwise they will be prompted to login.

Colour Pallet Class

The **ColourPallet.dart** class represents the colours that are used for the website. It has a background and border colour set to a dark grey, with two other gradients set to a purple and cyan. This class is used to keep a constant colour theme for the whole website.

Firebase Options Class

The **firebase_options.dart** class is a generated file by FlutterFire CLI and contains the default **FirebaseOptions** for different platforms in the Firebase app. It has a **web**, **android**, **ios** and **macos** constant to represent the Firebase options for each platform. Each constant has the following properties**: apikey, appId, messagingSenderID, projectID, databaseURL, storageBucket** and other platform specific settings.

Main Class

The Main Page is the starting page for the app. The **main.dart** class has the **main** function which initialises Firebase and starts the website. It contains the **MyApp** class that sets the overall theme and sets the home page to the **SelectaPage** (class described above).

Rankings Class

The rankings page can be used to view the rankings of all the users on the app. It displays the user’s level, total score, highest score, average score, lowest score and number of quizzes they completed.

1. The class defines a **rankings** list to store the rankings data.
2. The **fetchRankings** method is an asynchronous function that fetches the rankings data from Firestore. It retrieves the user data from the "Users" collection and iterates over the documents. For each user, it fetches the quiz results from the "QuizResults" collection and calculates various statistics such as total score, highest score, average score, lowest score, and the number of quizzes completed. It stores the ranking data in the **updatedRankings** list. Finally, it assigns the **updatedRankings** list to the **rankings** list.
3. The **sortRankings** method is called when the user taps on a column header to sort the rankings. It sorts the **rankings** list based on the selected column (identified by the **header** parameter) and the sort order (**isAscending**), it returns it in this order.
4. The **build** method builds the UI for the **RankingsPage**.
5. Inside the **FutureBuilder**, the **fetchRankings** method is called to fetch the rankings data asynchronously. The **FutureBuilder** displays different UI states based on the connection state and the snapshot.
6. The **DataTable** widget displays the rankings data with columns for different attributes such as name, level, total score, highest score, average score, lowest score, and the number of quizzes completed. Each column has an **onSort** callback that calls the **sortRankings** method to sort the rankings based on the selected column.

Select a Quiz Class

The **selectAQuiz.dart** class is used to show users all the possible quizzes that they could attempt.

1. The **SelectPage** class represents a page for selecting a quiz to answer.
2. It retrieves quiz information from a database using the **DatabaseService** class.
3. The **getQuizInformation** method fetches quiz details based on a category filter.
4. The **goToQuiz** method navigates to a specific quiz page based on its type.
5. The **build** method displays the quiz selection UI. It includes a dropdown button for category selection.
6. It uses a **FutureBuilder** widget to fetch and display the username asynchronously. It shows the number of unique quizzes completed by the user. It displays a list of quizzes with their details, including name, category, type, and number of questions. It checks for prerequisite quizzes and shows a dialog if necessary. The user can select a quiz and navigate to the respective quiz page.