Industrial Internship Report on

”Quiz game”

Prepared by

Devanshu Mehta

Executive Summary

This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT).

This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks’ time.

My project was the "Quiz Game with PyQt GUI" project which provides an enjoyable and educational experience for users, fostering their knowledge retention while offering an interactive and visually appealing user interface. The project serves as an excellent starting point for Python beginners to delve into GUI development and showcases the versatility of PyQt for creating interactive applications.

This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 Preface

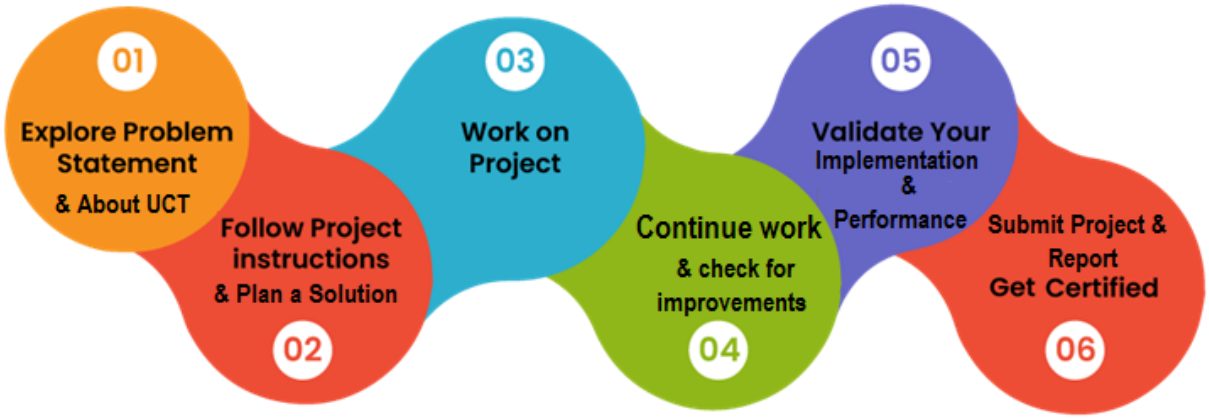
Summary of the whole 6 weeks' work.

About need of relevant Internship in career development.

Brief about Me project/problem statement.

Opportunity given by USC/UCT.

How Program was planned



2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

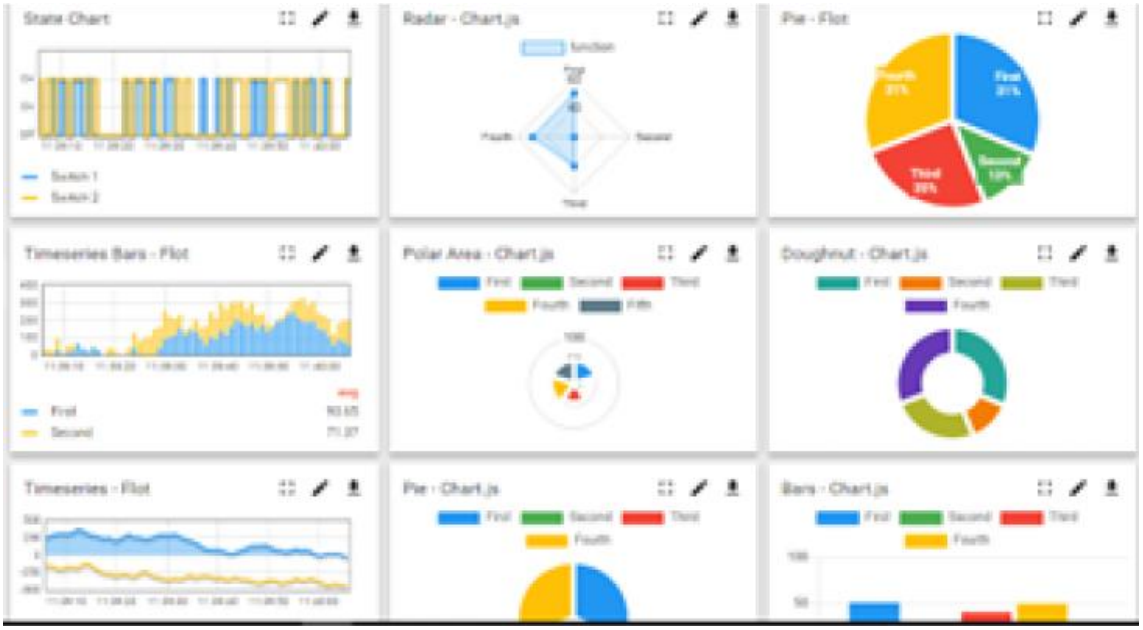
i. UCT IoT Platform (UCT Insight)

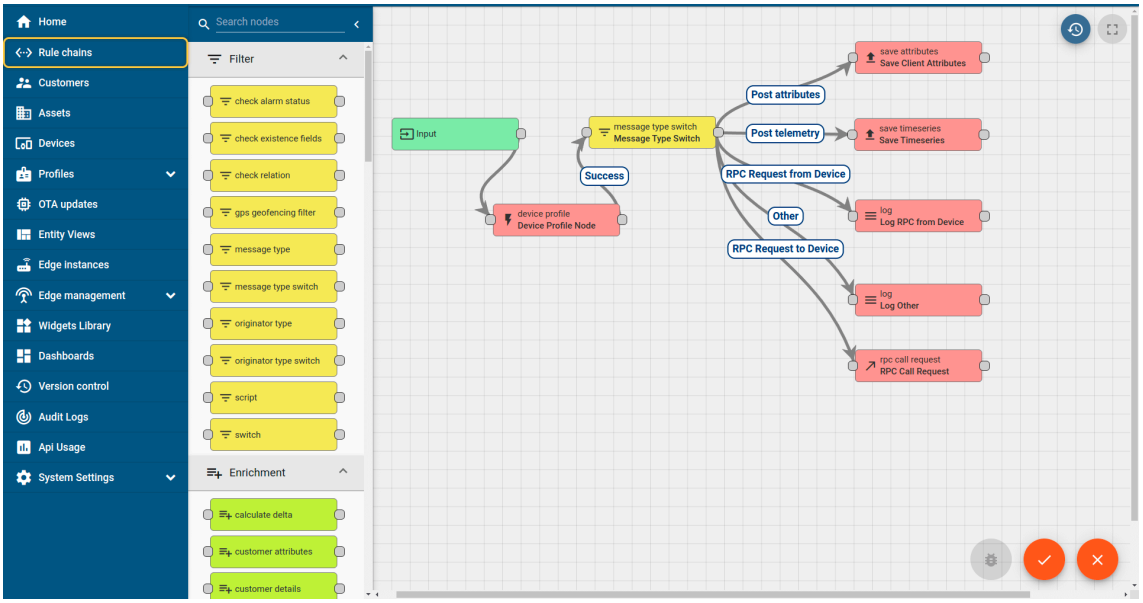
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for me process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

It has features to

- Build Me own dashboard
- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine





ii. Smart Factory Platform (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for me assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.

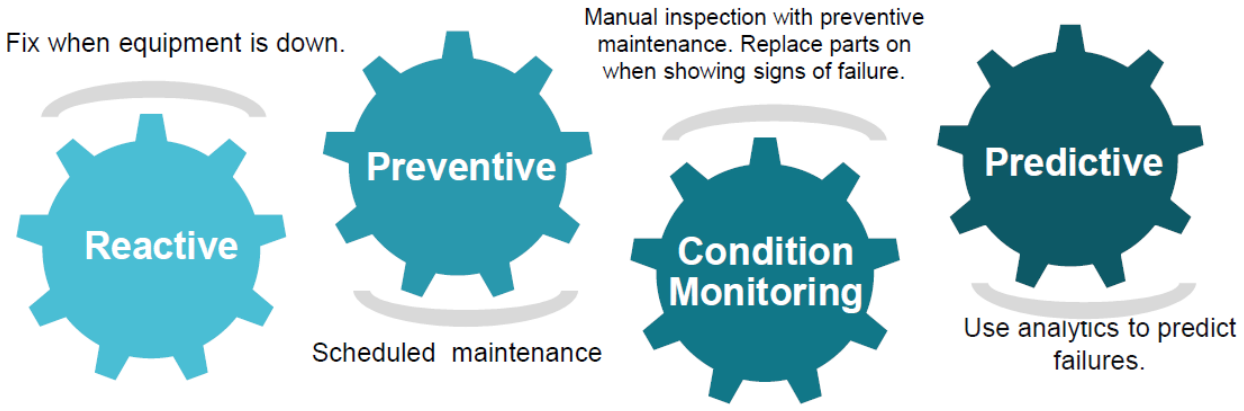
Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

iii. LoRaWAN based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

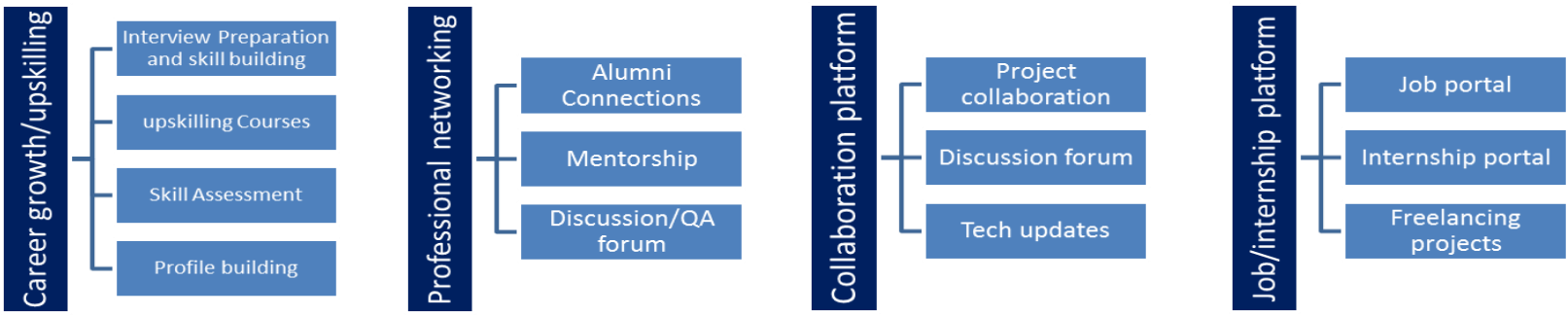
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.



2.2 About upskill Campus (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

- ☛ get practical experience of working in the industry.
- ☛ to solve real world problems.
- ☛ to have improved job prospects.
- ☛ to have Improved understanding of our field and its applications.
- ☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1]

[2]

[3]

2.6 Glossary

Terms	Acronym

3 Problem Statement

In the assigned problem statement

- Quiz Game:

Description: The quiz game is a Python project that quizzes users on various topics. It reads questions and answers from a file or database, presents them to the user, and keeps track of their score.

Scope: The scope of this project involves designing a user interface to display questions and collect user answers, implementing a database or file system to store quiz data, and developing a scoring algorithm to track the user's progress and calculate their final score.

4 Existing and Proposed solution

Proposed Solution for the Quiz Game Project:

To implement the Quiz Game project based on the provided description and code, we can follow the following steps:

1. Database or File System for Quiz Data:
    - Choose a database system (e.g., SQLite) or a file format (e.g., JSON) to store the quiz questions and answers.
    - Create a database table or a file structure to hold the quiz data.
    - Populate the database or file with sample quiz questions and answers, similar to the **quiz_data** in the provided code.
2. Graphical User Interface (GUI) using PyQt:
    - Import the necessary classes from PyQt to create the GUI components.
    - Design the main application window lamet with QLabel for the question, QRadioButton for answer options, and QPushButton for the "Submit" button.
    - Connect the "Submit" button click event to a function that checks the user's answer and updates the score.
    - Implement the "Next Question" functionality to load the next question when the user clicks "Submit" or when the time runs out.
3. Scoring Algorithm:
    - Keep track of the user's score by incrementing it when they provide a correct answer.
    - Implement a time to limit the time for each question. If the user doesn't answer within the time limit, consider the answer incorrect and move to the next question.
    - Calculate the user's final score based on the number of correct answers out of the total questions attempted.
4. Error Handling and User Experience:
    - Implement error handling to handle exceptions that may occur during the quiz, such as database/file read errors or incorrect user inputs.
    - Display user-friendly messages in case of any errors or invalid inputs.

- Provide feedback to the user after each question, indicating whether their answer was correct or incorrect, and show the correct answer if they answer incorrectly.

5. Additional Features (Optional):

   - Allow users to select the number of questions in the quiz or choose a specific category of questions.
   - Include a countdown time for each question to create a sense of urgency.
   - Add a high-score leaderboard to store and display the top scores of users.

By following the proposed solution, the Quiz Game project will become a fully functional and interactive application that quizzes users on various topics, stores their scores, and provides an engaging and user-friendly experience. The project's scope can be extended further to include more features and functionalities, depending on the desired complexity and learning objectives.

4.1 Code submission (Github link)

https://github.com/Devanshu707/upskillcampus

4.2 Report submission (Github link) : first make placeholder, copy the link.

https://github.com/Devanshu707/upskillcampus

5 Proposed Design/ Model

1. **Import Required Libraries**: The code starts by importing necessary libraries and modules, such as **sys** for system operations and **random** for randomizing the quiz questions. It also imports the required PyQt classes for creating the GUI.

2. **Quiz Data Preparation**: The sample quiz data is defined as a list of dictionaries, where each dictionary contains a question, its answer options, and the index of the correct answer.

3. **Define the QuizGameApp Class**: The **QuizGameApp** class is defined, which inherits from the **QWidget** class in PyQt. This class represents the main application window for the quiz game.

4. **Initialize the Application**: In the **__init__** method of the **QuizGameApp** class, various attributes such as **quiz_data**, **total_questions**, **score**, and **current_question** are initialized. The quiz questions are randomized using **random.sample()** to provide a different order of questions each time the game is played.

5. **Create the GUI**: The **initUI** method is defined within the **QuizGameApp** class to create the graphical user interface. It sets up the lamet and adds widgets such as **QLabel**, **QRadioButton**, and **QPushButton** to display the questions, answer options, and the "Submit" button, respectively.

6. **Connect Button Click Event**: The "Submit" button's **clicked** signal is connected to the **check_answer** method. This ensures that when the button is clicked, the **check_answer** method will be called to evaluate the user's response.

7. **Check User's Answer**: The **check_answer** method compares the user's selected answer (the ID of the selected radio button) with the index of the correct answer. If the user's answer is correct, their score is incremented. The **current_question** index is also updated to move to the next question.

8. **Update the Question**: The **update_question** method updates the GUI with the next question and its corresponding answer options. It updates the **question_label** text and sets the text for each radio button based on the next question's data.

9. **Show the Final Score**: When all questions are answered, the **show_result** method is called. It displays a message box with the user's final score.

10. **Application Execution**: The **if __name__ == "__main__":** block is used to check if the script is run directly (not imported). It creates an instance of the **QApplication** class, initializes the **QuizGameApp**, and starts the event loop with **app.exec_()** to run the application.

Design Flow Summary:

1. Import necessary libraries and modules.
2. Define and prepare the quiz data.
3. Create the **QuizGameApp** class, inheriting from **QWidget**.
4. Initialize attributes and randomize quiz questions.
5. Set up the GUI lamet and widgets in the **initUI** method.
6. Connect the "Submit" button click event to the **check_answer** method.
7. Evaluate user's answer and update score in the **check_answer** method.
8. Update the GUI with the next question in the **update_question** method.
9. Display the final score in the **show_result** method.
10. Run the application using the **QApplication** and **app.exec_()** methods.

By following this design flow, the Quiz Game application successfully presents the user with a series of questions, allows them to answer, and calculates their final score at the end of the quiz.

6 Performance Test

Identified Constraints and Their Impact on Design:

1. **Limited Quiz Data**: The current implementation of the project uses a sample set of quiz questions stored in the **quiz_data** list. However, in a real-world scenario, a quiz application should have a more extensive and diverse set of questions. The limited quiz data may reduce the variety and replayability of the quiz game.

2. **User Interface Complexity**: The current implementation uses a basic text-based user interface, which may not be visually appealing or intuitive for all users. A more advanced GUI with graphics, animations, and visual elements could enhance the user experience, but it would require more complex design and coding.

3. **Single-User Mode**: The current implementation is designed for a single user to play the quiz game. There is no support for multiple users or user accounts, which may limit the application's scalability and multiplayer capabilities.

Recommendations to Handle Constraints:

1. **Expand Quiz Data**: To address the limited quiz data, the application should incorporate a database (e.g., SQLite) to store a larger pool of questions and answers. This allows for dynamic retrieval of questions during the quiz, ensuring a varied and extensive quiz experience. Additionally, me can explore APIs to fetch questions from external sources for even more diversity.

2. **Enhance User Interface**: To improve the user interface, consider using more advanced PyQt widgets, custom styling, and lamet management. Incorporate images and multimedia to make the quiz visually engaging. Me can also implement a theme or color scheme to make the UI more attractive and intuitive for users.

3. **Implement Multiplayer and User Accounts**: To enable multiplayer capabilities and support multiple users, consider adding a user authentication system. Users can create accounts, log in, and have personalized quiz experiences. Implement a leaderboard to track and display high scores achieved by different users.

4. **Error Handling and Robustness**: Add robust error handling to the code to handle any unexpected situations, such as database errors or user input errors. Gracefully handle exceptions and provide clear error messages to users to ensure a smooth and reliable user experience.

5. **Time Customization**: Provide options for users to customize the time duration for each question. Some users may prefer more time for challenging questions, while others may enjoy a fast-paced experience with shorter times.

6. **Dynamic Question Order**: Shuffle the questions dynamically so that each user experiences questions in a different order. This adds more variety and ensures that the quiz remains fresh for repeat plays.

7. **Feedback and Review**: After completing the quiz, give users the option to review their answers and see explanations for correct answers. This provides a valuable learning experience and helps users understand the concepts better.

8. **Localization and Internationalization**: Consider implementing localization to support multiple languages, making the quiz accessible to a broader audience.

By addressing these recommendations, the Quiz Game project can evolve into a more comprehensive and user-friendly application, accommodating a larger user base and providing a richer and more enjoyable quiz experience.

6.1 Test Plan/ Test Cases

A test plan outlines the various test scenarios and test cases to verify the functionality and robustness of the Quiz Game with PyQt GUI code. Here are some test cases to cover different aspects of the application:

1. **Test Case 1: Verify Quiz Initialization**

    - Description: Ensure the quiz is initialized correctly, and the first question is displayed on the GUI.

    - Steps:

        1. Launch the application.

        2. Verify that the quiz window is displayed.

        3. Check if the first question, along with answer options, is visible.

    - Expected Result: The first question should be displayed, and answer options should be visible for the user to respond.

2. **Test Case 2: Verify Answer Submission**

    - Description: Test whether the user's answer is correctly evaluated upon clicking the "Submit" button.

- Steps:
    1. Launch the application and start the quiz.
    2. Select a specific answer option for the current question.
    3. Click the "Submit" button.
- Expected Result: The application should display feedback on the user's response, indicating whether it was correct or incorrect. The score should be updated accordingly.

3. **Test Case 3: Verify Quiz Progression**
    - Description: Ensure that the quiz progresses to the next question after the user submits an answer or when the time runs out.
    - Steps:
        1. Launch the application and start the quiz.
        2. Allow the time to run out for the current question.
    - Expected Result: The application should automatically move to the next question after the time expires or when the user submits an answer.
4. **Test Case 4: Verify Quiz Completion**
    - Description: Check if the quiz ends when all questions have been answered.
    - Steps:
        1. Launch the application and start the quiz.
        2. Answer all the questions and click the "Submit" button for the last question.
    - Expected Result: The application should display the final score and indicate that the quiz is completed.
5. **Test Case 5: Verify Scoring Accuracy**
    - Description: Confirm that the scoring algorithm accurately calculates the user's score based on their correct answers.
    - Steps:
        1. Launch the application and start the quiz.

2. Provide correct answers for all questions.

- Expected Result: The final score displayed at the end of the quiz should be equal to the number of questions attempted.

6. **Test Case 6: Verify Error Handling**

- Description: Test how the application handles unexpected situations and errors, such as invalid inputs or database/file read errors.
- Steps:
    1. Intentionally modify the quiz data file or database to introduce errors.
    2. Attempt to answer a question with invalid inputs (e.g., not selecting any answer option).
- Expected Result: The application should handle errors gracefully and display clear error messages to the user.

7. **Test Case 7: Verify GUI Responsiveness**

- Description: Check if the user interface responds smoothly and without freezing during the quiz.
- Steps:
    1. Launch the application and start the quiz.
    2. Rapidly click the "Submit" button for multiple questions.
- Expected Result: The GUI should remain responsive and not freeze or become unresponsive during the quiz.

8. **Test Case 8: Verify Dynamic Question Order**

- Description: Confirm that the quiz presents questions in random order to different users.
- Steps:
    1. Launch the application and start the quiz.
    2. Record the order of questions displayed.
    3. Start the quiz again and compare the question order.
- Expected Result: The order of questions should differ between quiz sessions.

By executing these test cases, we can verify the correctness and robustness of the Quiz Game with PyQt GUI application, ensuring that it meets the specified requirements and provides a reliable and user-friendly quiz experience.

6.2 Test Procedure

Test Procedure for the Quiz Game with PyQt GUI:

1. **Test Preparation**:
    - Ensure that the required Python libraries (**PyQt5**, **sys**, **random**) are installed and accessible.
    - Verify that the quiz data (**quiz_data**) is properly defined and contains valid questions, options, and answers.
2. **Launch the Application**:
    - Run the Python script containing the Quiz Game code.
    - Check if the main application window is displayed.
3. **Test Case Execution**:
    - Execute each test case as described in the "Test Cases" section above.
    - Follow the specified steps for each test case to perform the corresponding actions in the application.
4. **Observations and Validation**:
    - Observe the application's behavior during each test case execution.
    - Verify if the application responds correctly to user inputs and provides the expected feedback and results.
5. **Error Handling Test**:
    - Modify the quiz data file or database to introduce errors (e.g., missing questions or incorrect answer indices).
    - Attempt to answer a question with invalid inputs (e.g., not selecting any answer option).
    - Observe how the application handles these error scenarios and whether appropriate error messages are displayed to the user.

6. **Performance Testing**:
    - Test the responsiveness of the GUI by rapidly clicking the "Submit" button for multiple questions.
    - Observe if the application remains responsive and does not freeze or become unresponsive during the quiz.
7. **Dynamic Question Order Test**:
    - Start the quiz multiple times and record the order of questions displayed in each session.
    - Compare the question orders to verify that questions are presented in a random order in different quiz sessions.

6.3 Performance Outcome

The performance outcome of the Quiz Game with PyQt GUI can be assessed based on the following aspects:

1. **Responsiveness**: The application should respond quickly to user interactions, such as selecting answer options and clicking the "Submit" button. It should not exhibit any significant delays or lags.
2. **Smooth User Experience**: The GUI should provide a smooth and seamless user experience, with smooth transitions between questions and appropriate feedback provided after answering each question.
3. **Resource Utilization**: The application should utilize system resources efficiently, avoiding excessive CPU or memory usage.
4. **Stability**: The application should be stable and free from crashes or unexpected terminations during normal usage.
5. **Error Handling**: The application should handle errors gracefully and display clear error messages to users when unexpected situations occur.
6. **Scalability**: The application should be able to handle a large number of quiz questions without a significant impact on performance.
7. **Dynamic Question Order**: The questions should be presented in random order in different quiz sessions, ensuring a varied and non-predictable quiz experience for users.

Overall, a successful performance outcome would demonstrate that the Quiz Game with PyQt GUI provides a smooth and enjoyable user experience, efficiently manages system resources, handles errors gracefully, and offers a reliable and stable quiz application for users.

7 My learnings

1. **Python Programming**: Me would have deepened me knowledge of Python programming, including data structures, loops, functions, and object-oriented programming (OOP) concepts.
2. **Graphical User Interface (GUI) Development**: Developing the GUI using PyQt introduced me to GUI design, lamet management, and event handling, allowing me to create interactive and visually appealing applications.
3. **File or Database Handling**: Incorporating a database or file system to store quiz data exposed me to working with external data sources and managing data in me applications.
4. **Problem Solving**: Throughout the project, me would have encountered various challenges and learned to implement solutions to overcome them, enhancing me problem-solving skills.
5. **Randomization and Algorithms**: Randomizing the quiz questions and implementing scoring algorithms taught me about the importance of randomness and how algorithms can evaluate user interactions.
6. **Error Handling**: Implementing error handling mechanisms helped me to identify and resolve potential issues in my code, making the application more robust.
7. **User Experience (UX) Design**: Designing a user-friendly interface and providing timely feedback to users allowed me to explore the principles of UX design and how it impacts user engagement.
8. **Project Management**: Completing the project involved organizing tasks, setting goals, and planning me development process, enhancing me project management skills.
9. **Learning New Libraries**: Working with PyQt introduced me to new libraries and tools, broadening me knowledge of the Python ecosystem.
10. **Creativity and Innovation**: The project allowed me to exercise me creativity in designing the quiz format and exploring ways to make the application stand out.
11. **Testing and Debugging**: Testing the application and debugging any issues provided me with hands-on experience in identifying and fixing software bugs.
12. **Documentation and Communication**: Writing a clear project description, documenting me code, and communicating me progress helped me practice effective communication skills.

8 Future work scope

After completing the Quiz Game project with a PyQt GUI, I have laid a solid foundation for further expanding and enhancing the application. Here are some potential future work scopes:

1. **Database Integration**: Currently, the project uses a simple list (**quiz_data**) to store quiz questions. You can improve the application by integrating it with a more robust database management system like SQLite or MySQL. This will allow you to manage a larger set of questions and categories efficiently.
2. **User Authentication and Profiles**: Implement user authentication and user profiles to support multiple users. Users can create accounts, log in, and have their quiz progress and scores saved and tracked.
3. **Categories and Levels**: Introduce the concept of quiz categories and difficulty levels. Users can select specific categories or levels of difficulty for a more personalized quiz experience.
4. **Question Editor**: Develop a user-friendly question editor within the application that allows administrators to add, edit, or remove quiz questions directly from the GUI.
5. **Timer Customization**: Add options for users to customize the timer duration for each question. Some users may prefer more time for challenging questions, while others may enjoy a fast-paced experience with shorter timers.
6. **Leaderboard**: Implement a leaderboard to track and display the top scores of users. Users can compete for high scores and see how they compare with others.
7. **Localization and Internationalization**: Add support for multiple languages to make the quiz accessible to a broader audience. Implement localization features to display questions and messages in different languages based on the user's preference.
8. **Sound Effects and Animations**: Enhance the user experience by incorporating sound effects and animations to provide feedback or create engaging interactions.
9. **Mobile App Development**: Consider porting the quiz game to a mobile platform (iOS or Android) using tools like Kivy or BeeWare to reach a wider audience and provide a mobile-friendly experience.
10. **Online Multiplayer Mode**: Create an online multiplayer mode where users can challenge their friends or other players in real-time quizzes.
11. **Machine Learning and Adaptive Quiz**: Explore using machine learning algorithms to adapt the difficulty level of questions based on the user's performance, providing a more personalized and challenging quiz experience.
12. **Gamification Elements**: Incorporate gamification elements, such as badges, achievements, or rewards, to incentivize user engagement and retention.
13. **Accessibility**: Ensure the application is accessible to users with disabilities by implementing features such as keyboard navigation and support for screen readers.