**REPORT ON**

Career Recommendation System

*Submitted in partial fulfilment of the requirements for the award of the degree of*

**BACHELOR OF COMPUTER APPLICATIONS**

****

**Batch: 2021 - 24**

***Under the Guidance of Submitted By***

Dr. Ruchi Sawhney Ankit Alex Minz 04328402021

Dibya Bharat 00928402021

**Bosco Technical Training Society**

**Don Bosco Technical School, Okhla Road, New Delhi-110025**

**9643868820, 8527787221, 011-41033889**

**Affiliated to**

****

**Guru Gobind Singh Indraprastha University**

### Sector - 16C, Dwarka, New Delhi - 110078 (India)

### Phone: +91-11-25302170, Fax: +91-11- 25302111

### E-Mail Id: ggsipu.pr@rediffmail.com, pro@ipu.ac.in

# 

## 

## **Acknowledgement**

The note starts with thanks to Almighty who actually created this piece of work and helped us when things were not easy for us.

I am very grateful and indebted to my Faculty/Guide Mrs Ruchi Sawhney who immensely helped and rendered her valuable advice, precious time, knowledge and relevant information regarding the collection of material. She has been a major source of inspiration throughout the project as she not only guided me throughout this Project Report “Career Recommendation System” but also encouraged me to solve problems that arose during this report.

Her guidance and suggestions about this Project report have really enlightened me. It has been a great help to support to have him around.

And finally, I would like to mention appreciation to our parents and friends who have been instrumental throughout this period by providing unrelenting encouragement.

Signature

Ankit Alex Minz

04328402021

**CERTIFICATE**

This is to certify that the dissertation/project report entitled “Career Recommendation System” done by me is an authentic work carried out for the partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Applications under the guidance of Mrs Ruchi Sawhney. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

Signature of the student

ANKIT ALEX MINZ

04328402021

MAIN REPORT

**Chapter 1: Introduction**

The career recommendation system is a desktop application project created using the python programming language. This project was created in light of a lack of system to judge and provide students with a straightforward list of jobs based on their scores and interest to not only get view of what they can do with their current scores and skill as well as a view of the weaknesses they need to make up for in order to do what they interested.

The various components used in this project are:

**Python:**

Python is a high-level, general-purpose, and versatile programming language. Python programming language is being used in web development, Machine Learning applications, along with all cutting-edge technology in Software Industry. Python’s Background in Desktop Application Development and Machine Learning comes from its cross-platform nature and various libraries for Example: Pyqt5, Pandas, NumPy, Sci-kit Learn etc.

**Pyqt5:**

There are so many options provided by Python to develop GUI application and PyQt5 is one of them. PyQt5 is cross-platform GUI toolkit, it has a set of python bindings for Qt v5. One can develop an interactive desktop application with much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called ‘QtDesigner’ to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.

Due to the nature of Desktop Application development and Pyqt5, Event Driven Programming come into play. It is a programming paradigm where different parts of the application or entities communicate by sending messages to one another through an intermediary. In the context of Gui development and desktop application development this refers to different components reacting to each other through signals and events written out by the programmer. For example, a button being clicked in a Pyqt5 Gui interface is connected to function which would then be executed.

**Scikit-Learn:**

Scikit-learn is a powerful open-source library in Python used for machine learning tasks such as classification, regression, clustering, and more. It is built on top of other popular scientific computing libraries, such as NumPy, SciPy, and Matplotlib. It offers a wide range of tools for machine learning and statistical modelling, including various algorithms and utilities for data preprocessing, model selection, evaluation, and data visualization.

In practical terms, scikit-learn serves as a powerful toolbox for tasks such as data preprocessing, model training, and model evaluation. It includes utilities for handling data, implementing feature selection, and assessing model performance through cross-validation and grid search.

**Pandas:**

Pandas is an open-source Python library used for data manipulation and analysis. It provides powerful data structures and tools for working with structured data, primarily in the form of DataFrame objects. Pandas is widely utilized in machine learning for its ability to handle various data formats, such as CSV files, Excel sheets, SQL databases, and more. Its main data structure, the DataFrame, allows for easy indexing, slicing, cleaning, transforming, and aggregating data, making it an essential tool for data preprocessing in machine learning workflows.

It enables practitioners to load datasets, handle missing values, perform feature engineering by creating new features or transforming existing ones, encode categorical variables, and split data into training and testing sets. The ability to manipulate and prepare data efficiently using pandas is fundamental for ensuring that machine learning models receive high-quality input, leading to more accurate predictions or classifications.

**NumPy:**

NumPy, short for Numerical Python, is a fundamental open-source library in Python for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. In machine learning, NumPy is extensively used for data manipulation and preparation. It plays a central role in handling datasets, transforming features, and performing mathematical operations required during the preprocessing phase. Many machine learning libraries, including scikit-learn, TensorFlow, and PyTorch, leverage NumPy arrays as the underlying data structure. NumPy's efficient implementation of array operations significantly speeds up computations, making it an essential tool for implementing algorithms and models efficiently in machine learning workflows.

**Machine Learning:**

Machine Learning is a branch of artificial intelligence that develops algorithms by learning the hidden patterns of the datasets used it to make predictions on new similar type data, without being explicitly programmed for each task.

Machine learning is used in many different applications, from image and speech recognition to natural language processing, recommendation systems, fraud detection, portfolio optimization, automated task, and so on. Machine learning models are also used to power autonomous vehicles, drones, and robots, making them more intelligent and adaptable to changing environments.  
A typical machine learning tasks are to provide a recommendation. Recommender systems are a common application of machine learning, and they use historical data to provide personalized recommendations to users.

Types of Machine Learning:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Reinforcement Machine Learning

**Supervised Learning:**

In this project supervised learning has been chosen for usage, supervised learning is a type of machine learning in which the algorithm is trained on the labeled dataset i.e., the data may have correct values as well. It learns to map input features to targets based on labeled training data. In supervised learning, the algorithm is provided with input features and corresponding output labels, and it learns to generalize from this data to make predictions on new, unseen data.

There are two main types of supervised learning:

1. Regression
2. Classification

**Classification:**

Classification is a type of supervised learning where the algorithm learns to assign input data to a specific category or class based on input features. The output labels in classification are discrete values. Classification algorithms can be binary, where the output is one of two possible classes, or multiclass, where the output can be one of several classes. The different Classification algorithms in machine learning are: Logistic Regression, Naive Bayes, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), etc.

**KNN Algorithm:**

In this project the KNN algorithm has been chosen from classification type supervised learning, K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It can also handle both numerical and categorical data, making it a flexible choice for various types of datasets in classification and regression tasks. It is a non-parametric method that makes predictions based on the similarity of data points in a given dataset. K-NN is less sensitive to outliers compared to other algorithms. The applications of KNN are data preprocessing, pattern recognition, Recommendation Engines.

**Content based filtering:**

Content-based filtering is a machine learning approach used in recommendation systems to suggest items to users based on the characteristics of the items and the preferences of the users. Instead of relying on the collaborative behaviour of users (as seen in collaborative filtering), content-based filtering considers the features or attributes of items and recommends new items that are similar to those a user has liked or interacted with in the past.

In machine learning applications, content-based filtering finds its use in various recommendation systems, such as suggesting movies, music, articles, or products to users based on their historical preferences or explicit feedback.

Using content based filtering in machine learning models can recommend new items that align with users' tastes, making the recommendation process more personalized and potentially increasing user satisfaction and engagement.

**Chapter 2: Objective and Scope**

**Objectives:**

* Develop a machine learning-driven recommendation system that analyzes students' academic performance, interests, strengths, and career aspirations which provides course, stream, and career recommendations based on content filtering.
* Identify and address subject-specific weaknesses that may hinder students' chosen career paths.
* Implement a user-friendly questionnaire to refine recommendations based on user tendencies and make further recommendations on things of their interest that they lack awareness of or even provide new things to get interested in for those who have exhausted their option.
* Suggest improvement strategies: For subjects in which students are weak, the system will offer tailored recommendations on strengthening their skills and improving their performance.

**Scope:**

* The project's scope encompasses providing students with informed career recommendations following their 10th-grade examinations.
* Recommendations are personalized through an in-depth analysis of academic data, interests, and subject weaknesses.
* Strategies to enhance proficiency in relevant subjects are offered.
* A user-centric questionnaire refines recommendations by evaluating user inclinations.

**Chapter 3: Definition of Problem**

**Problem:**

The "Career Recommendation System" project addresses a pressing issue in the educational landscape, primarily concerning students transitioning from their 10th-grade examinations. At this critical juncture, students often find themselves at a crossroads, faced with the formidable task of choosing academic courses or streams that will significantly shape their future careers. However, the educational system currently lacks a structured and comprehensive decision-making framework to guide students through this crucial phase of their academic journey.

In particular, students grapple with several challenges:

* **Lack of Systematic Guidance:** The absence of a systematic guidance system leaves many students in the dark, unsure of which path to follow. Without clear direction, they may make ill-informed decisions that can impact their career prospects for years to come.
* **Balancing Performance and Aspirations:** Striking the right balance between their academic performance, personal interests, and inherent strengths is a complex and often bewildering task. Students must align their educational choices with their career aspirations, a process that is far from straightforward.

As a result, many students are left navigating this critical decision-making process with minimal support or personalized advice. This lack of guidance can have profound consequences on their educational and career trajectories, potentially leading to mismatches between their chosen paths and their true potential.

**Background:**

The existing educational framework has long grappled with the challenge of equipping students with the tools and knowledge necessary to make informed career choices. Traditionally, students have relied on a mix of limited resources, including career counsellors and educational advisors, which often fall short of providing the personalized guidance needed.

The complexities of aligning academic performance, personal interests, and career goals further exacerbate the problem. The evolving job market demands that students choose their paths with care, as they prepare for careers in fields that are continually changing and diversifying. Consequently, there is a growing need for an innovative solution that addresses these challenges effectively.

**Relevance:**

The "Career Recommendation System" project is profoundly relevant in the context of today's educational landscape. It directly addresses the critical issue of career decision-making for students, offering a tailored and data-driven approach to support their academic and vocational aspirations. Several key aspects underline the system's relevance:

* **Personalized Recommendations:** By leveraging data on academic performance, personal interests, and strengths, the system provides personalized course and career recommendations. This tailoring ensures that students are directed toward paths that genuinely resonate with their capabilities and aspirations.
* **Subject Weakness Mitigation:** The system goes a step further by diagnosing subject-specific weaknesses. This critical feature empowers students to address these weaknesses and bolster their educational foundations, thereby enhancing their chances of success in their chosen careers.
* **Improved Academic and Career Trajectories:** The project holds the potential to significantly improve students' academic and career trajectories. By offering precise, data-driven recommendations, it helps students make more informed decisions, reducing the likelihood of misaligned choices.

**Chapter 4: System Requirements**

**Hardware Requirements:**

* RAM: 4 Gb minimum
* Storage: 464 mb minimum

**Software Requirements:**

* Operating system platform- atleast windows 10
* Database management system- SQLite
* Programming languages – Python
* Machine learning libraries and frameworks – sci-kit learn
* Data analysis tools – Pandas, NumPy, Matplotlib, seaborn
* GUI generation libraries: PyQT5
* IDEs: Python IDLE, Anaconda Jupyter

**Chapter 5: System Analysis & Design**

**Iterative model**

The iterative model in software engineering is an approach where the project is broken down into smaller cycles or iterations. Each iteration goes through the entire software development life cycle (SDLC), encompassing planning, design, implementation, testing, and deployment. Here's a breakdown of the steps in the iterative model:

* **Planning:** Define the project scope, objectives, and goals. Identify the features or functionalities to be developed in the current iteration. Plan resources, timelines, and iteration goals.
* **Requirement Analysis:** Gather and analyze user requirements specific to the current iteration. Focus on understanding what users need for the features or functionalities planned for this cycle.
* **Design:** Create a design based on the gathered requirements. This includes architectural design, system design, and detailed design of individual components or features targeted for the current iteration.
* **Implementation/Coding:** Develop the software based on the design specifications. Write code for the identified features or functionalities planned for this iteration.
* **Testing:** Perform testing activities such as unit testing, integration testing, and system testing. Ensure that the developed features meet the requirements and function correctly.
* **Evaluation/Review**: Evaluate the results of the iteration. Collect feedback from stakeholders, users, and testing. Identify areas for improvement and assess whether the iteration met its goals.
* **Deployment:** If the iteration meets the quality standards and requirements, deploy the developed features. This might involve releasing the software to a limited user base or integrating it into the existing system.
* **Feedback/Refinement:** Gather feedback from users and stakeholders about the deployed features. Use this feedback to refine and improve the product. This input often feeds back into the planning phase for subsequent iterations.
* **Iterations:** Repeat the cycle for the next iteration. Incorporate changes based on feedback and update the plan to address new requirements or modifications.



**Use Case:**

A use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams.

Following are the purposes of a use case diagram given below:

* It gathers the system's needs.
* It depicts the external view of the system.
* It recognizes the internal as well as external factors that influence the system.
* It represents the interaction between the actors.

**Components of use case diagram**

* **Actors:** Actors represent the users or external entities interacting with the system. An actor can be a human user, another system, or any entity that interacts with the system. Actors are depicted as stick figures or named entities outside the system boundary.
* **Use Cases:** Use cases represent the specific functionalities or services that the system provides to its users. Each use case describes a set of actions or interactions between the user and the system to achieve a specific goal. Use cases are depicted as ovals or ellipses within the system boundary.
* **Relationships and Associations:**

1. **Association (Communication Line):** Lines connecting actors to use cases represent interactions between actors and use cases. These lines show that an actor is involved in or can trigger a particular use case.
2. **Inclusion (Include):** Indicates that one use case includes the functionality of another use case. For example, a "Login" use case might include an "Authenticate" use case.
3. **Extension (Extend):** Shows that one use case extends or adds optional behaviour to another use case under certain conditions. It signifies optional and conditional behaviour.

* **Generalization:** Represents a relationship between more general and more specific use cases, showing inheritance or specialization. It's similar to inheritance in object-oriented programming.
* **System Boundary:** The boundary or box around the use cases represents the scope or boundary of the system being modeled. It encapsulates all the use cases and actors involved within the system.





**Explanation:**The above Diagram show how the user would interact with the system.   
**Login:** The login process would occur in a login page developed through Pyqt5 whose functionality has been connected to the database, the Login Window contains a link to the Register Window, for those who have not registered / new users.

**Register:** The register process would occur in a register page developed through Pyqt5 whose functionality has been connected to the database, the Register Window will go back to message window showing success in the Registration, the message window contains a button leading back to the Login Window.

**Chapter 6: System Planning (PERT CHART)**

Project Evaluation and Review Technique (PERT) is a procedure through which activities of a project are represented in its appropriate sequence and timing. It is a scheduling technique used to schedule, organize and integrate tasks within a project. PERT is basically a mechanism for management planning and control which provides blueprint for a particular project. All of the primary elements or events of a project have been finally identified by the PERT. In this technique, a PERT Chart is made which represent a schedule for all the specified tasks in the project.

**Measures of PERT Chart:**

**1. Early Start (ES):** Early Start refers to the earliest possible point in time when a particular task or activity can begin. It's determined by considering the immediate predecessors of the task—activities that must be completed before it can start.

**2. Early Finish (EF):** Early Finish denotes the earliest possible point in time when a task or activity can be completed. It is calculated by adding the task duration to its Early Start time.

**3. Late Start (LS):** Late Start represents the latest point in time when a task can start without delaying the project's overall completion. It considers the project's deadline and the time required for the task.

**4. Late Finish (LF):** Late Finish indicates the latest allowable completion time for a task without impacting the project's completion date. It is determined by subtracting the task duration from its Late Start time.

**5. Slack (or Float):** Slack or Float refers to the amount of time a task can be delayed without affecting subsequent tasks or the overall project completion time. It's calculated as the difference between the Late Start and Early Start times (or Late Finish and Early Finish times) for a task.

**Components of PERT Chart:**

**Arrows:** Connects one activity to the next.

**Table:** Contains the various measure of regarding the duration of the activity and the activity’s name.

**Start:** Represents the start of the project:

**Finish:**  Represents the end the of the project.

****

**Chapter 7: Methodology**

We have followed the steps given below:

**Step 1: Dataset creation**

In this step, we have created our own dataset which is a part of primary data collection techniques. Primary data collection is a type of data collection where data is obtained directly from the first-hand source through experiments, surveys or observations. This dataset was created using python random numbers generator function to create random number which acts as marks of student in different subjects. The datasets consist of around 12000 rows and each row contain marks of student in different subject, courses and interest. We did not perform any data filtering or data cleaning techniques because the data was created in correct form to fulfil the desired requirements of the project.

**2. Model creation**

After the datasets created successfully, we have moved towards creation of machine learning model. This model is created using KNN( K nearest neighbour ) technique , a type of classification algorithm of supervised machine learning. This model recommends the best course for the user according to their marks in different subject.

**3. UI creation**

When the model creation was finished, the creation of used interface of our application started. We use Qt designer an UI designer application which provide various components for creating UI in python. It provides drag and drop feature which makes the creation of UI easy and faster. It also has a benefit which converts the .ui files into .py files. We have created so many UI pages like registration, login , menu page etc.

**4. Database Creation**

In this project we need a database to store user information such as name, marks, userid and interest. We have created two table named

**5. UI function implementation**

To make it running desktop application , we have implemented different function on various components of user interface. We can take example of user registration, if a user enters the required information on the registration page and click register button then the information of user will be store in the database and a successful registration message appears.

**6. Testing**

In testing, we have performed unit testing in each and every step of the project such as accuracy of model, buttons of user interface, etc. We have checked each programming file by taking various inputs. After the completion of unit testing, we connect the different units and then we check the working of combined units like transferring of data from one page to another page, storing of data etc.

**Chapter 8: Detail life cycle of the project**

**ERD (Entity Relationship Diagram)**

The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. In short, the ER Diagram is the structural format of the database.

**Components of ER diagram**

**1. Entity:** An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

* **Strong Entity:** A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These are called Strong Entity Types.
* **Weak Entity:** An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can’t be defined. These are called Weak Entity types.



**2. Attributes:** Attributes are the properties that define the entity type. For example, Roll\_No, Name, DOB, Age, Address, and Mobile\_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

* **Key Attribute:** The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll\_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with underlying lines.
* **Composite Attribute:** An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.
* **Multivalued Attribute:** An attribute consisting of more than one value for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.
* **Derived Attribute:** An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

**Cardinality**

The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

* **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one.
* **One-to-Many:** In one-to-many mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is 2.
* **Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one.
* **Many-to-Many:** When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



**ER Diagram :**

****

**DFD**

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. It is a graphical tool, useful for communicating with users ,managers and other personnel. it is useful for analyzing existing as well as proposed system.

**Components of DFD:**

* **Process:** Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle. The process is named a short sentence, in one word or a phrase to express its essence.
* **Data Flow:** Data flow describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved. Data flow also represents material along with information that is being moved. Material shifts are modeled in systems that are not merely informative. A given flow should only transfer a single type of information. The direction of flow is represented by the arrow which can also be bi-directional.
* **Data store:** The data is stored in the data store for later use. Two horizontal lines represent the symbol of the store. The data store is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet. The data store can be viewed independent of its implementation. When the data flow from the data store it is considered as data reading and when data flows to the data store it is called data entry or data updating.
* **Source or Sink:** The Source is an external entity that stands outside of the system and communicates with the system. It can be, for example, organizations like banks, groups of people like customers or different departments of the same organization, which is not a part of the model system and is an external entity. Modeled systems also communicate with Source.



**Levels of DFD:**

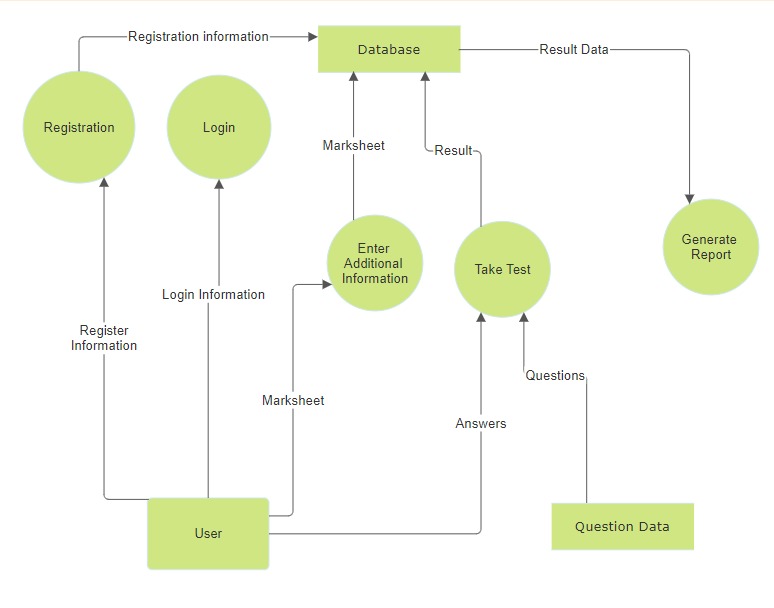
DFD uses hierarchy to maintain transparency thus multilevel DFD’s can be created. Levels of DFD are as follows:

* **0-level DFD:** It represents the entire system as a single bubble and provides an overall picture of the system.
* **1-level DFD:** It represents the main functions of the system and how they interact with each other.
* **2-level DFD:** It represents the processes within each function of the system and how they interact with each other.
* **3-level DFD:** It represents the data flow within each process and how the data is transformed and stored.

**DFD Level 0:**



**DFD Level 1:**

****

**Input and output screen design**

**PyQt5**

PyQt5 is cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called ‘QtDesigner’ to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.



**Here are some key aspects of PyQt5:**

* **Cross-platform:** Applications developed with PyQt5 can run on various operating systems like Windows, macOS, Linux, etc., without many modifications. This cross-platform capability is due to Qt's nature.
* **GUI Development:** PyQt5 provides tools and classes to design and build graphical user interfaces using Qt Designer or programmatically within Python code. It includes widgets, layouts, dialogs, and more for creating interactive interfaces.
* **Event Handling:** It allows developers to handle user interactions, events, signals, and slots effectively. This enables the creation of responsive applications by linking user actions to specific functions or methods.
* **Integration with Python:** PyQt5 seamlessly integrates with Python, allowing developers to leverage the ease of Python programming while harnessing the power of the Qt framework.
* **Support for Multimedia:** PyQt5 supports multimedia functionalities, such as playing audio and video files, thanks to Qt's multimedia modules.
* **Database Integration:** It provides tools for integrating with databases, allowing developers to create applications that interact with various database systems

**Components of pyqt5**

PyQt5 offers a range of components and classes that enable developers to create Graphical User Interfaces (GUIs) in Python. Here are some key components commonly used in PyQt5 for GUI creation:

* **Widgets:** PyQt5 provides a variety of widgets that form the building blocks of a GUI. These include:

1. **QPushButton:** Button widget for triggering actions.
2. **QLabel:** Display text or an image.
3. **QLineEdit:** Single-line text input field.
4. **QTextEdit:** Multiline text input/editing field.
5. **QCheckBox and QRadioButton:** Checkboxes and radio buttons for user selections.
6. **QComboBox and QListWidget:** Dropdown lists and list widgets for item selection.
7. **QSlider and QSpinBox:** Widgets for selecting numerical values.

* **Layout Management:** PyQt5 includes layout classes that help in organizing and arranging widgets within a window or container. Common layout classes are:

1. **QVBoxLayout and QHBoxLayout:** Vertical and horizontal box layouts.
2. **QGridLayout:** Grid-based layout for arranging widgets in rows and columns.
3. **QFormLayout**: Organizes input fields and their labels in a structured form.

* **Dialogs:** PyQt5 provides pre-built dialog windows for specific purposes, such as file dialogs, message boxes, input dialogs, and more. These include:

1. **QFileDialog:** Dialog for file selection.
2. **QMessageBox:** Dialog for displaying messages or alerts.
3. **QInputDialog:** Dialog for user input.

* **Custom Widgets:** Developers can create custom widgets by subclassing existing PyQt5 widgets or by creating entirely new ones to suit specific application needs.
* **Events and Signals:** PyQt5 uses a signals and slots mechanism to handle events and inter-widget communication. This allows actions in one widget to trigger specific functions or methods in another widget.
* **Graphics and Multimedia:** PyQt5 includes classes for working with graphics and multimedia elements, allowing developers to incorporate images, videos, and graphics within their applications.

**Qt Designer**

Qt Designer is a visual design tool provided by the Qt framework for creating graphical user interfaces (GUIs). It is part of the Qt development environment and is commonly used in conjunction with PyQt, a set of Python bindings for the Qt framework. Here are the key features of Qt Designer:

* **Visual Design:** Qt Designer allows developers to design GUIs visually by dragging and dropping UI components onto a form. This provides a WYSIWYG (What You See Is What You Get) environment, allowing designers and developers to see the appearance of the UI as they design it.
* **Widget Box:** Qt Designer provides a widget box that contains a variety of standard UI components (widgets) such as buttons, labels, text fields, and more. Developers can easily select and place these widgets on the form.
* **Layout Management:** It supports the design of complex layouts through various layout managers, including vertical and horizontal box layouts, grid layouts, and form layouts. This helps in arranging and organizing widgets within a window.
* **Property Editor:** Developers can use the property editor to customize the properties of UI components, such as their size, position, text, and appearance. This allows for fine-tuning the visual aspects of the GUI.
* **Signal and Slot Editing:** Qt Designer provides an interface for connecting signals and slots, which facilitates event handling and interaction between different UI components. This is a fundamental aspect of Qt's programming model.
* **Integration with Code Editors:** While Qt Designer is primarily a visual tool, it seamlessly integrates with code editors. Developers can easily switch between the visual design in Qt Designer and the corresponding code in their preferred integrated development environment (IDE).
* **Custom Widget Integration:** Qt Designer allows developers to integrate custom widgets into the design process. This is particularly useful when developers have created their own specialized UI components.
* **Preview Mode:** Developers can preview the appearance and behaviour of the GUI directly within Qt Designer before generating the code. This helps in identifying any design issues before deploying the application.
* **UI File Generation:** Qt Designer saves the designed UI in a .ui file format, which is an XML-based file. This file can be loaded and used by the application code, enabling the separation of UI design and logic.



**Process Involved**

**Activity Diagram**

A UML Activity diagram is a graphical representation used to model workflows and business processes within a system. It primarily focuses on the flow of activities or actions performed within a system, showcasing the sequence, conditions, and parallelism of these actions. Here are the main components of a UML Activity diagram:

* **Initial Node:** Denotes the starting point of the activity diagram. It represents the initiation of the workflow.
* **Activity or Action**: Represents a specific task or action within the system. It can be any operation, such as a calculation, decision-making process, or interaction.
* **Control Flow:** Indicates the flow or sequence of activities. It's represented by arrows connecting different activities, showing the order in which actions occur.
* **Decision or Merge Node:** Represents a branching point in the workflow where a decision is made or where different flows merge back together.
* **Fork and Join Nodes:** Fork nodes split the flow into multiple concurrent flows, while join nodes synchronize these concurrent flows back into a single flow.
* **Final Node:** Denotes the end of the activity diagram, representing the completion of the workflow.
* **Guard Condition**: Represents the condition that determines which path to follow in a decision point. It's often depicted as text near the decision node or control flow arrow.
* **Swimlanes (Partition):** Used to group activities performed by different actors or system components. They visually separate actions performed by different entities within the system.
* **Object Nodes:** Represent objects or data consumed or produced during activities.

****

**Methodology used testing**  
**Testing**

Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free. The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements. It mainly aims at measuring the specification, functionality, and performance of a software program or application.



Types of software testing:

1. Manual Testing

2. Automation Testing

**Manual Testing**

Manual testing includes testing software manually, i.e., without using any automation tool or script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behaviour or bug.

There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing. Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Types of manual testing:

1. White Box

2. Black Box

3. Grey Box

**White Box**

White box testing techniques analyse the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

White box testing is also known as structural testing or code-based testing, and it is used to test the software’s internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

**Chapter 9: Code and Screenshots**

**Login page:**

from PyQt5 import QtCore, QtGui, QtWidgets

import sqlite3

con = sqlite3.connect("G:\minorproject\Database\Career\_Recommedation\_System.db")

class Ui\_loginWindow(object):

def OpenRegistration(self, loginWindow):

from registration\_page import Ui\_RegistrationWIndow

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_RegistrationWIndow()

self.ui.setupUi(self.window)

self.window.show()

loginWindow.hide()

def setupUi(self, loginWindow):

self.cur = con.cursor()

loginWindow.setObjectName("loginWindow")

loginWindow.resize(422, 343)

self.centralwidget = QtWidgets.QWidget(loginWindow)

self.centralwidget.setObjectName("centralwidget")

self.UserNameLbl = QtWidgets.QLabel(self.centralwidget)

self.UserNameLbl.setGeometry(QtCore.QRect(70, 90, 71, 16))

self.UserNameLbl.setObjectName("UserNameLbl")

self.UserNameLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.UserNameLineEdit.setGeometry(QtCore.QRect(150, 85, 113, 22))

self.UserNameLineEdit.setObjectName("UserNameLineEdit")

self.LoginButton = QtWidgets.QPushButton(self.centralwidget)

self.LoginButton.setGeometry(QtCore.QRect(30, 200, 93, 28))

self.LoginButton.setObjectName("LoginButton")

self.LoginButton.clicked.connect(lambda: self.Login(loginWindow))

self.RegisterButton = QtWidgets.QPushButton(self.centralwidget)

self.RegisterButton.setGeometry(QtCore.QRect(290, 200, 93, 28))

self.RegisterButton.setObjectName("RegisterButton")

self.RegisterButton.clicked.connect(lambda: self.OpenRegistration(loginWindow))

loginWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(loginWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 422, 26))

self.menubar.setObjectName("menubar")

loginWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(loginWindow)

self.statusbar.setObjectName("statusbar")

loginWindow.setStatusBar(self.statusbar)

self.retranslateUi(loginWindow)

QtCore.QMetaObject.connectSlotsByName(loginWindow)

def retranslateUi(self, loginWindow):

\_translate = QtCore.QCoreApplication.translate

loginWindow.setWindowTitle(\_translate("loginWindow", "Login\_Page"))

self.UserNameLbl.setText(\_translate("loginWindow", "Username:"))

self.LoginButton.setText(\_translate("loginWindow", "Login"))

self.RegisterButton.setText(\_translate("loginWindow", "Register"))

def gotomenu(self, loginWindow, message, name):

from menu\_page import Ui\_MenuWindow

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MenuWindow()

self.ui.setupUi(self.window)

self.ui.LoginMsgLbl.setText(message)

self.ui.User\_id = name

self.window.show()

loginWindow.hide()

def LoginMsg(self, loginWindow, message):

from after\_registration import Ui\_MsgWindow

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MsgWindow()

self.ui.setupUi(self.window)

self.ui.RegistrationMsgLbl.setText(message)

self.window.show()

loginWindow.hide()

def fetchname(self, user\_id):

statement = f"SELECT First\_name from user\_info WHERE User\_id='{user\_id}';"

self.cur.execute(statement)

result = self.cur.fetchone()

name = result[0] if result else ""

return name

def Login(self, loginWindow):

user\_id = self.UserNameLineEdit.text()

statement = f"SELECT User\_id from user\_info WHERE User\_id='{user\_id}';"

self.cur.execute(statement)

if not self.cur.fetchone():

self.LoginMsg(loginWindow, "Login failed")

else:

name = self.fetchname(user\_id)

self.gotomenu(loginWindow, f"Welcome, {name}. Please enter your information here:", user\_id)

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

loginWindow = QtWidgets.QMainWindow()

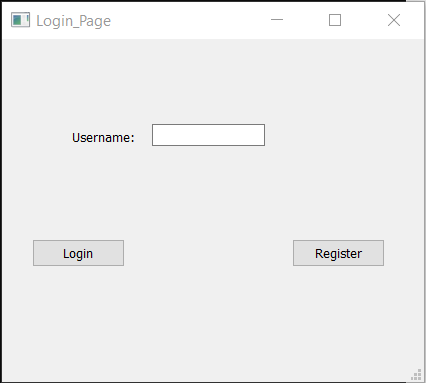
ui = Ui\_loginWindow()

ui.setupUi(loginWindow)

loginWindow.show()

sys.exit(app.exec\_())

**Output:**



**Registeration page:**

from PyQt5 import QtCore, QtGui, QtWidgets

from after\_registration import Ui\_MsgWindow

import sqlite3

con = sqlite3.connect("G:\minorproject\Database\Career\_Recommedation\_System.db")

class Ui\_RegistrationWIndow(object):

def setupUi(self, RegistrationWIndow):

self.cur = con.cursor()

RegistrationWIndow.setObjectName("RegistrationWIndow")

RegistrationWIndow.resize(339, 440)

font = QtGui.QFont()

font.setFamily("Calibri")

font.setPointSize(16)

font.setBold(True)

font.setWeight(75)

RegistrationWIndow.setFont(font)

self.centralwidget = QtWidgets.QWidget(RegistrationWIndow)

self.centralwidget.setObjectName("centralwidget")

self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)

self.verticalLayout.setObjectName("verticalLayout")

self.splitter = QtWidgets.QSplitter(self.centralwidget)

self.splitter.setOrientation(QtCore.Qt.Horizontal)

self.splitter.setObjectName("splitter")

self.RegistrationLbl = QtWidgets.QLabel(self.splitter)

self.RegistrationLbl.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))

self.RegistrationLbl.setAlignment(QtCore.Qt.AlignCenter)

self.RegistrationLbl.setObjectName("RegistrationLbl")

self.verticalLayout.addWidget(self.splitter)

self.FirstNameLbl = QtWidgets.QLabel(self.centralwidget)

self.FirstNameLbl.setObjectName("FirstNameLbl")

self.verticalLayout.addWidget(self.FirstNameLbl)

self.FirstNameLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.FirstNameLineEdit.setObjectName("FirstNameLineEdit")

self.verticalLayout.addWidget(self.FirstNameLineEdit)

self.LastNameLbl = QtWidgets.QLabel(self.centralwidget)

self.LastNameLbl.setObjectName("LastNameLbl")

self.verticalLayout.addWidget(self.LastNameLbl)

self.LastNameLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.LastNameLineEdit.setObjectName("LastNameLineEdit")

self.verticalLayout.addWidget(self.LastNameLineEdit)

self.UserIdLbl = QtWidgets.QLabel(self.centralwidget)

self.UserIdLbl.setLayoutDirection(QtCore.Qt.LeftToRight)

self.UserIdLbl.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)

self.UserIdLbl.setObjectName("UserIdLbl")

self.verticalLayout.addWidget(self.UserIdLbl)

self.UserIdLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.UserIdLineEdit.setObjectName("UserIdLineEdit")

self.verticalLayout.addWidget(self.UserIdLineEdit)

spacerItem = QtWidgets.QSpacerItem(20, 20, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)

self.verticalLayout.addItem(spacerItem)

self.RegisterButton = QtWidgets.QPushButton(self.centralwidget)

self.RegisterButton.setObjectName("RegisterButton")

self.RegisterButton.clicked.connect(lambda: self.Register(RegistrationWIndow))

self.verticalLayout.addWidget(self.RegisterButton)

RegistrationWIndow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(RegistrationWIndow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 339, 26))

self.menubar.setObjectName("menubar")

RegistrationWIndow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(RegistrationWIndow)

self.statusbar.setObjectName("statusbar")

RegistrationWIndow.setStatusBar(self.statusbar)

self.retranslateUi(RegistrationWIndow)

QtCore.QMetaObject.connectSlotsByName(RegistrationWIndow)

def retranslateUi(self, RegistrationWIndow):

\_translate = QtCore.QCoreApplication.translate

RegistrationWIndow.setWindowTitle(\_translate("RegistrationWIndow", "Registration\_Page"))

self.RegistrationLbl.setText(\_translate("RegistrationWIndow", "Registration"))

self.FirstNameLbl.setText(\_translate("RegistrationWIndow", "First Name"))

self.LastNameLbl.setText(\_translate("RegistrationWIndow", "Last Name"))

self.UserIdLbl.setText(\_translate("RegistrationWIndow", "User ID"))

self.RegisterButton.setText(\_translate("RegistrationWIndow", "Register"))

def RegistrationMsg(self, RegistrationWIndow, message):

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MsgWindow()

self.ui.setupUi(self.window)

self.ui.RegistrationMsgLbl.setText(message)

self.window.show()

RegistrationWIndow.hide()

def Register(self,RegistrationWIndow):

try:

firstname=self.FirstNameLineEdit.text()

lastname=self.LastNameLineEdit.text()

userid=self.UserIdLineEdit.text()

statement= f"INSERT INTO user\_info VALUES('{userid}', '{firstname}', '{lastname}')"

self.cur.execute(statement)

con.commit()

self.RegistrationMsg(RegistrationWIndow, "Your Registration was successful")

except Exception as e:

errmsg=f"Error during registration: {str(e)}"

self.RegistrationMsg(RegistrationWIndow, errmsg)

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

RegistrationWIndow = QtWidgets.QMainWindow()

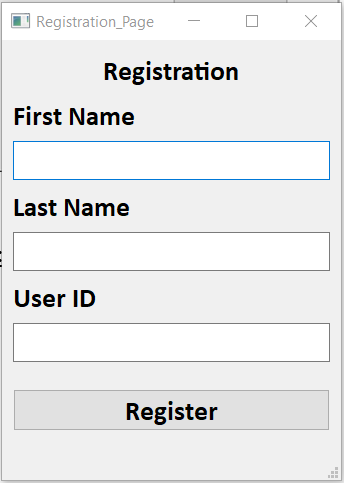
ui = Ui\_RegistrationWIndow()

ui.setupUi(RegistrationWIndow)

RegistrationWIndow.show()

sys.exit(app.exec\_())

**Output:**



**Msg page:**

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MsgWindow(object):

def OpenLogin(self,MsgWindow):

from login\_page import Ui\_loginWindow

self.window=QtWidgets.QMainWindow()

self.ui = Ui\_loginWindow()

self.ui.setupUi(self.window)

self.window.show()

MsgWindow.hide()

def setupUi(self, MsgWindow):

MsgWindow.setObjectName("MsgWindow")

MsgWindow.resize(500, 489)

self.centralwidget = QtWidgets.QWidget(MsgWindow)

self.centralwidget.setObjectName("centralwidget")

self.layoutWidget = QtWidgets.QWidget(self.centralwidget)

self.layoutWidget.setGeometry(QtCore.QRect(130, 40, 283, 90))

self.layoutWidget.setObjectName("layoutWidget")

self.verticalLayout = QtWidgets.QVBoxLayout(self.layoutWidget)

self.verticalLayout.setContentsMargins(0, 0, 0, 0)

self.verticalLayout.setObjectName("verticalLayout")

self.RegistrationMsgLbl = QtWidgets.QLabel(self.layoutWidget)

font = QtGui.QFont()

font.setFamily("Calibri")

font.setPointSize(14)

self.RegistrationMsgLbl.setFont(font)

self.RegistrationMsgLbl.setAlignment(QtCore.Qt.AlignCenter)

self.RegistrationMsgLbl.setObjectName("RegistrationMsgLbl")

self.verticalLayout.addWidget(self.RegistrationMsgLbl)

spacerItem = QtWidgets.QSpacerItem(232, 27, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

self.verticalLayout.addItem(spacerItem)

self.GotoLoginButton = QtWidgets.QPushButton(self.layoutWidget)

font = QtGui.QFont()

font.setFamily("Calibri")

font.setPointSize(11)

font.setBold(True)

font.setWeight(75)

self.GotoLoginButton.setFont(font)

self.GotoLoginButton.setObjectName("GotoLoginButton")

self.GotoLoginButton.clicked.connect(lambda: self.OpenLogin(MsgWindow))

self.verticalLayout.addWidget(self.GotoLoginButton)

MsgWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MsgWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 500, 26))

self.menubar.setObjectName("menubar")

MsgWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MsgWindow)

self.statusbar.setObjectName("statusbar")

MsgWindow.setStatusBar(self.statusbar)

self.retranslateUi(MsgWindow)

QtCore.QMetaObject.connectSlotsByName(MsgWindow)

def retranslateUi(self, MsgWindow):

\_translate = QtCore.QCoreApplication.translate

MsgWindow.setWindowTitle(\_translate("MsgWindow", "Msg\_Page"))

self.RegistrationMsgLbl.setText(\_translate("MsgWindow", "Registration message"))

self.GotoLoginButton.setText(\_translate("MsgWindow", "Go to Login"))

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

MsgWindow = QtWidgets.QMainWindow()

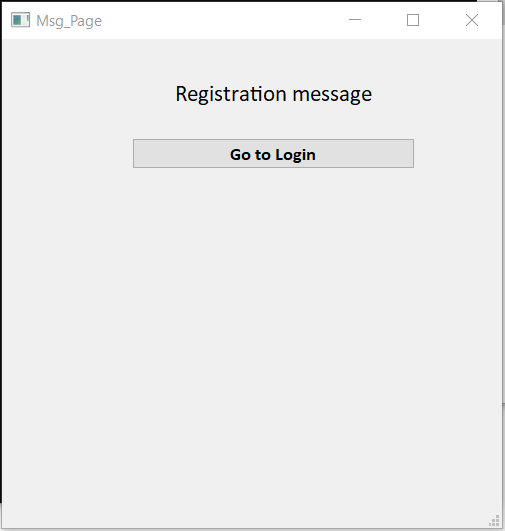
ui = Ui\_MsgWindow()

ui.setupUi(MsgWindow)

MsgWindow.show()

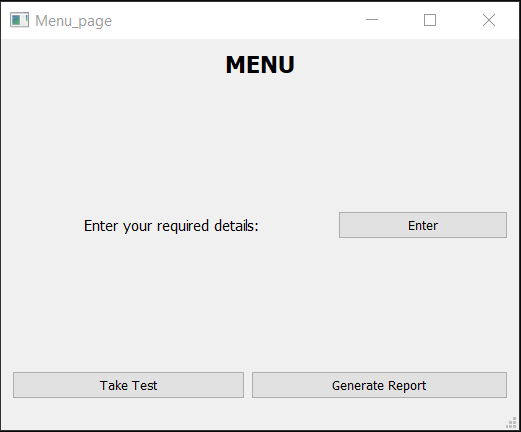
sys.exit(app.exec\_())

**Output:**



**Menu page:**

**Output:**



**Information page:**

from PyQt5 import QtCore, QtGui, QtWidgets

import sqlite3

con = sqlite3.connect("G:\minorproject\Database\Career\_Recommedation\_System.db")

class Ui\_EnterInfoWindow(object):

def setupUi(self, EnterInfoWindow):

self.cur = con.cursor()

User\_id = ""

EnterInfoWindow.setObjectName("EnterInfoWindow")

EnterInfoWindow.resize(798, 611)

self.centralwidget = QtWidgets.QWidget(EnterInfoWindow)

self.centralwidget.setObjectName("centralwidget")

self.EnterInfoButton = QtWidgets.QPushButton(self.centralwidget)

self.EnterInfoButton.setGeometry(QtCore.QRect(150, 460, 93, 28))

self.EnterInfoButton.setObjectName("EnterInfoButton")

self.EnterInfoButton.clicked.connect(lambda: self.EnterInfo(User\_id))

self.BackButton = QtWidgets.QPushButton(self.centralwidget)

self.BackButton.setGeometry(QtCore.QRect(390, 460, 93, 28))

self.BackButton.setObjectName("BackButton")

self.BackButton.clicked.connect(lambda: self.Back(EnterInfoWindow))

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(20, 10, 171, 41))

font = QtGui.QFont()

font.setPointSize(10)

font.setBold(True)

font.setWeight(75)

self.label.setFont(font)

self.label.setObjectName("label")

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(80, 115, 71, 16))

self.label\_2.setObjectName("label\_2")

self.EnglishLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.EnglishLineEdit.setGeometry(QtCore.QRect(190, 110, 113, 22))

self.EnglishLineEdit.setObjectName("EnglishLineEdit")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(80, 185, 81, 16))

self.label\_3.setObjectName("label\_3")

self.MathsLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.MathsLineEdit.setGeometry(QtCore.QRect(190, 180, 113, 22))

self.MathsLineEdit.setObjectName("MathsLineEdit")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(80, 255, 91, 16))

self.label\_4.setObjectName("label\_4")

self.SstLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.SstLineEdit.setGeometry(QtCore.QRect(190, 250, 113, 22))

self.SstLineEdit.setObjectName("SstLineEdit")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(470, 115, 55, 16))

self.label\_5.setObjectName("label\_5")

self.ScienceLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.ScienceLineEdit.setGeometry(QtCore.QRect(560, 110, 113, 22))

self.ScienceLineEdit.setObjectName("ScienceLineEdit")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(470, 185, 61, 16))

self.label\_6.setObjectName("label\_6")

self.ComputerLineEdit = QtWidgets.QLineEdit(self.centralwidget)

self.ComputerLineEdit.setGeometry(QtCore.QRect(560, 180, 113, 22))

self.ComputerLineEdit.setObjectName("ComputerLineEdit")

self.label\_7 = QtWidgets.QLabel(self.centralwidget)

self.label\_7.setGeometry(QtCore.QRect(80, 340, 171, 16))

self.label\_7.setObjectName("label\_7")

self.label\_8 = QtWidgets.QLabel(self.centralwidget)

self.label\_8.setGeometry(QtCore.QRect(20, 60, 241, 16))

font = QtGui.QFont()

font.setPointSize(9)

self.label\_8.setFont(font)

self.label\_8.setObjectName("label\_8")

self.InterestsCB = QtWidgets.QComboBox(self.centralwidget)

self.InterestsCB.setGeometry(QtCore.QRect(280, 335, 331, 22))

self.InterestsCB.setObjectName("InterestsCB")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.InterestsCB.addItem("")

self.label\_9 = QtWidgets.QLabel(self.centralwidget)

self.label\_9.setGeometry(QtCore.QRect(30, 510, 681, 41))

font = QtGui.QFont()

font.setBold(True)

font.setWeight(75)

self.label\_9.setFont(font)

self.label\_9.setWordWrap(True)

self.label\_9.setObjectName("label\_9")

EnterInfoWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(EnterInfoWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 798, 26))

self.menubar.setObjectName("menubar")

EnterInfoWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(EnterInfoWindow)

self.statusbar.setObjectName("statusbar")

EnterInfoWindow.setStatusBar(self.statusbar)

self.retranslateUi(EnterInfoWindow)

QtCore.QMetaObject.connectSlotsByName(EnterInfoWindow)

def retranslateUi(self, EnterInfoWindow):

\_translate = QtCore.QCoreApplication.translate

EnterInfoWindow.setWindowTitle(\_translate("EnterInfoWindow", "Information\_Page"))

self.EnterInfoButton.setText(\_translate("EnterInfoWindow", "ENTER INFO"))

self.BackButton.setText(\_translate("EnterInfoWindow", "Back"))

self.label.setText(\_translate("EnterInfoWindow", "Enter your Details"))

self.label\_2.setText(\_translate("EnterInfoWindow", "English:"))

self.label\_3.setText(\_translate("EnterInfoWindow", "Mathematics:"))

self.label\_4.setText(\_translate("EnterInfoWindow", "Social Studies:"))

self.label\_5.setText(\_translate("EnterInfoWindow", "Science:"))

self.label\_6.setText(\_translate("EnterInfoWindow", "Computer:"))

self.label\_7.setText(\_translate("EnterInfoWindow", "what are you interested in:"))

self.label\_8.setText(\_translate("EnterInfoWindow", "Enter your class 10 marks below:"))

self.InterestsCB.setItemText(0, \_translate("EnterInfoWindow", "Select"))

self.InterestsCB.setItemText(1, \_translate("EnterInfoWindow", "Computer Science and Information Technology"))

self.InterestsCB.setItemText(2, \_translate("EnterInfoWindow", "Mechanical and Electrical"))

self.InterestsCB.setItemText(3, \_translate("EnterInfoWindow", "Electronics and Communication"))

self.InterestsCB.setItemText(4, \_translate("EnterInfoWindow", "Construction and Design"))

self.InterestsCB.setItemText(5, \_translate("EnterInfoWindow", "Hospitality and Event Management"))

self.InterestsCB.setItemText(6, \_translate("EnterInfoWindow", "Life Sciences and Environment"))

self.InterestsCB.setItemText(7, \_translate("EnterInfoWindow", "Arts and Media"))

self.InterestsCB.setItemText(8, \_translate("EnterInfoWindow", "Physical Education and Wellness"))

self.InterestsCB.setItemText(9, \_translate("EnterInfoWindow", "FInance, Bussiness and Marketing"))

self.InterestsCB.setItemText(10, \_translate("EnterInfoWindow", "Culinary Studies and Cooking"))

self.label\_9.setText(\_translate("EnterInfoWindow", "Note: if you are not satisfied with your marks in certain subject then feel free to give a test for them! Note: Please Give a Test for Logical Reasoning from the Menu before checking your Report!"))

def Back(self, EnterInfoWindow):

from menu\_page import Ui\_MenuWindow

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MenuWindow()

self.ui.setupUi(self.window)

self.window.show()

EnterInfoWindow.hide()

def EnterInfo(self, User\_id):

counter=0

english=self.EnglishLineEdit.text()

maths=self.MathsLineEdit.text()

sst=self.SstLineEdit.text()

science=self.ScienceLineEdit.text()

comp=self.ComputerLineEdit.text()

interests=self.InterestsCB.currentText()

if counter == 0:

counter= counter + 1

statement= f"INSERT INTO User\_Marks VALUES({english}, {maths}, {sst}, {science}, {comp}, 0, {interests}, {counter})"

else:

statement= f"UPDATE User\_Marks SET English = {english}, Mathematics = {maths}, Social\_Studies = {sst}, Science = {science}, Computer = {comp} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

EnterInfoWindow = QtWidgets.QMainWindow()

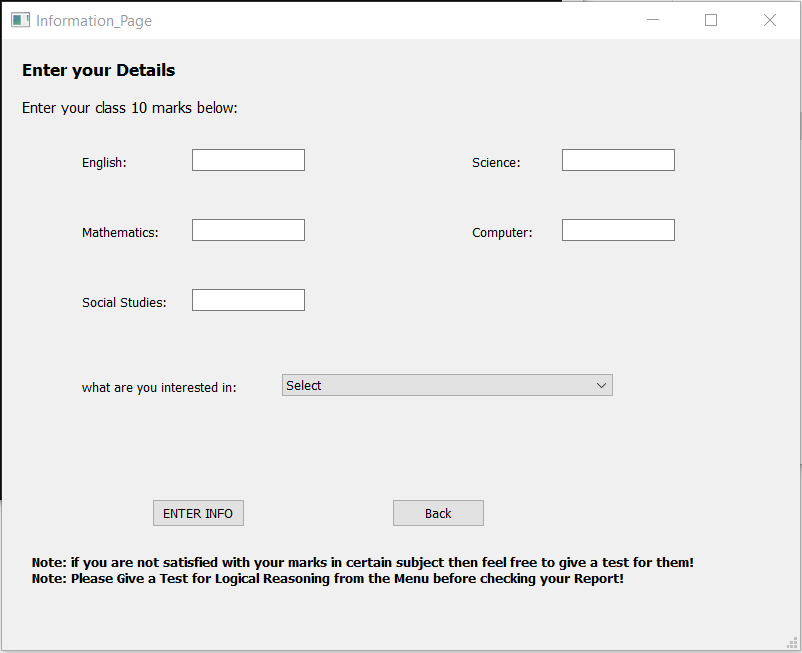
ui = Ui\_EnterInfoWindow()

ui.setupUi(EnterInfoWindow)

EnterInfoWindow.show()

sys.exit(app.exec\_())

**Output:**



**Choose Subject page:**

from PyQt5 import QtCore, QtGui, QtWidgets

import random as rndm

import pandas as pd

import numpy as np

class Ui\_SubjectWindow(object):

def setupUi(self, SubjectWindow):

self.User\_id = ""

SubjectWindow.setObjectName("SubjectWindow")

SubjectWindow.resize(371, 428)

self.centralwidget = QtWidgets.QWidget(SubjectWindow)

self.centralwidget.setObjectName("centralwidget")

self.EnglishButton = QtWidgets.QPushButton(self.centralwidget)

self.EnglishButton.setGeometry(QtCore.QRect(50, 70, 93, 28))

self.EnglishButton.setObjectName("EnglishButton")

self.EnglishButton.clicked.connect(lambda: self.EngTest(SubjectWindow))

self.ScienceButton = QtWidgets.QPushButton(self.centralwidget)

self.ScienceButton.setGeometry(QtCore.QRect(230, 70, 93, 28))

self.ScienceButton.setObjectName("ScienceButton")

self.ScienceButton.clicked.connect(lambda: self.SciTest(SubjectWindow))

self.MathsButton = QtWidgets.QPushButton(self.centralwidget)

self.MathsButton.setGeometry(QtCore.QRect(50, 150, 93, 28))

self.MathsButton.setObjectName("MathsButton")

self.MathsButton.clicked.connect(lambda: self.MathTest(SubjectWindow))

self.LogicalReasoningButton = QtWidgets.QPushButton(self.centralwidget)

self.LogicalReasoningButton.setGeometry(QtCore.QRect(230, 150, 121, 28))

self.LogicalReasoningButton.setObjectName("LogicalReasoningButton")

self.LogicalReasoningButton.clicked.connect(lambda: self.LGTest(SubjectWindow))

self.SSTButton = QtWidgets.QPushButton(self.centralwidget)

self.SSTButton.setGeometry(QtCore.QRect(50, 230, 93, 28))

self.SSTButton.setObjectName("SSTButton")

self.SSTButton.clicked.connect(lambda: self.SStTest(SubjectWindow))

self.ComputerButton = QtWidgets.QPushButton(self.centralwidget)

self.ComputerButton.setGeometry(QtCore.QRect(230, 230, 93, 28))

self.ComputerButton.setObjectName("ComputerButton")

self.ComputerButton.clicked.connect(lambda: self.CompTest(SubjectWindow))

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(40, 10, 301, 21))

font = QtGui.QFont()

font.setPointSize(9)

font.setBold(True)

font.setWeight(75)

self.label.setFont(font)

self.label.setObjectName("label")

self.BackButton = QtWidgets.QPushButton(self.centralwidget)

self.BackButton.setGeometry(QtCore.QRect(140, 310, 93, 28))

self.BackButton.setObjectName("BackButton")

self.BackButton.clicked.connect(lambda: self.Back(SubjectWindow))

SubjectWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(SubjectWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 371, 26))

self.menubar.setObjectName("menubar")

SubjectWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(SubjectWindow)

self.statusbar.setObjectName("statusbar")

SubjectWindow.setStatusBar(self.statusbar)

self.retranslateUi(SubjectWindow)

QtCore.QMetaObject.connectSlotsByName(SubjectWindow)

def retranslateUi(self, SubjectWindow):

\_translate = QtCore.QCoreApplication.translate

SubjectWindow.setWindowTitle(\_translate("SubjectWindow", "Choose Subject"))

self.EnglishButton.setText(\_translate("SubjectWindow", "English"))

self.ScienceButton.setText(\_translate("SubjectWindow", "Science"))

self.MathsButton.setText(\_translate("SubjectWindow", "Mathematics"))

self.LogicalReasoningButton.setText(\_translate("SubjectWindow", "Logical Reasoning"))

self.SSTButton.setText(\_translate("SubjectWindow", "Social Studies"))

self.ComputerButton.setText(\_translate("SubjectWindow", "Computer"))

self.label.setText(\_translate("SubjectWindow", "Click the Subject you to Take a Test In"))

self.BackButton.setText(\_translate("SubjectWindow", "Back"))

def Back(self, SubjectWindow):

from menu\_page import Ui\_MenuWindow

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MenuWindow()

self.ui.setupUi(self.window)

self.window.show()

SubjectWindow.hide()

def EngTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\english\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,50)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.English = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

def MathTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\Mathematics\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,53)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.Mathematics = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

def SStTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\Social\_Studies\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,51)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.Social\_Studies = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

def SciTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\Science\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,50)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.Science = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

def LGTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\Logical\_Reasoning\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,50)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.Logical\_Reasoning = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

def CompTest(self, SubjectWindow):

from take\_test\_all\_other\_questions import Ui\_TakeTestWindow

data = pd.read\_excel(r"G:\reps\minor-project\Datasets\Computer\_questions.xlsx")

Questions = []

ans=[]

while(len(Questions)!=25):

r = rndm.randint(0,42)

if(r not in Questions):

Questions.append(r)

else:

continue

for i in Questions:

ans.append(data.iloc[i,5])

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_TakeTestWindow()

self.ui.setupUi(self.window)

self.ui.User\_id = self.User\_id

self.ui.Computer = Questions

self.ui.ans = ans

self.ui.RenderQ(0)

self.window.show()

SubjectWindow.hide()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

SubjectWindow = QtWidgets.QMainWindow()

ui = Ui\_SubjectWindow()

ui.setupUi(SubjectWindow)

SubjectWindow.show()

sys.exit(app.exec\_())

**Take Test page:**

from PyQt5 import QtCore, QtGui, QtWidgets

import random as rndm

import pandas as pd

import numpy as np

import sqlite3

con = sqlite3.connect("G:\minorproject\Database\Career\_Recommedation\_System.db")

class Ui\_TakeTestWindow(object):

def setupUi(self, TakeTestWindow):

self.cur = con.cursor()

self.User\_id = ""

self.marks = 0

self.finished = False

self.ans = []

self.selected\_answer = ""

self.count = 0

self.English = []

self.Mathematics = []

self.Social\_Studies = []

self.Science = []

self.Logical\_Reasoning = []

self.Computer = []

font = QtGui.QFont()

font.setFamily("Noto Sans")

font.setPointSize(9)

TakeTestWindow.setObjectName("TakeTestWindow")

TakeTestWindow.resize(800, 600)

self.centralwidget = QtWidgets.QWidget(TakeTestWindow)

self.centralwidget.setObjectName("centralwidget")

self.Option2Button = QtWidgets.QRadioButton(self.centralwidget)

self.Option2Button.setGeometry(QtCore.QRect(40, 230, 731, 31))

self.Option2Button.setObjectName("Option2Button")

self.AnswerbuttonGroup = QtWidgets.QButtonGroup(TakeTestWindow)

self.AnswerbuttonGroup.setObjectName("AnswerbuttonGroup")

self.AnswerbuttonGroup.addButton(self.Option2Button)

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(10, 80, 781, 40))

font = QtGui.QFont()

font.setPointSize(9)

self.label\_2.setFont(font)

self.label\_2.setWordWrap(True)

self.label\_2.setObjectName("label\_2")

self.Option4Button = QtWidgets.QRadioButton(self.centralwidget)

self.Option4Button.setGeometry(QtCore.QRect(40, 330, 741, 31))

self.Option4Button.setObjectName("Option4Button")

self.AnswerbuttonGroup.addButton(self.Option4Button)

self.Option1Button = QtWidgets.QRadioButton(self.centralwidget)

self.Option1Button.setGeometry(QtCore.QRect(40, 180, 721, 31))

self.Option1Button.setObjectName("Option1Button")

self.AnswerbuttonGroup.addButton(self.Option1Button)

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(20, 20, 151, 31))

font = QtGui.QFont()

font.setPointSize(10)

font.setBold(True)

font.setWeight(75)

self.label.setFont(font)

self.label.setObjectName("label")

self.NextButton = QtWidgets.QPushButton(self.centralwidget)

self.NextButton.setGeometry(QtCore.QRect(620, 460, 93, 28))

self.NextButton.setObjectName("NextButton")

self.NextButton.clicked.connect(lambda: self.nextButtonClicked(TakeTestWindow))

self.Option3Button = QtWidgets.QRadioButton(self.centralwidget)

self.Option3Button.setGeometry(QtCore.QRect(40, 280, 741, 31))

self.Option3Button.setObjectName("Option3Button")

self.AnswerbuttonGroup.addButton(self.Option3Button)

self.BackButton = QtWidgets.QPushButton(self.centralwidget)

self.BackButton.setGeometry(QtCore.QRect(80, 460, 93, 28))

self.BackButton.setObjectName("BackButton")

self.BackButton.clicked.connect(lambda: self.back\_button\_clicked())

TakeTestWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(TakeTestWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 26))

self.menubar.setObjectName("menubar")

TakeTestWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(TakeTestWindow)

self.statusbar.setObjectName("statusbar")

TakeTestWindow.setStatusBar(self.statusbar)

self.Option2Button.setFont(font)

self.label\_2.setFont(font)

self.Option4Button.setFont(font)

self.Option1Button.setFont(font)

self.label.setFont(font)

self.NextButton.setFont(font)

self.Option3Button.setFont(font)

self.BackButton.setFont(font)

self.retranslateUi(TakeTestWindow)

QtCore.QMetaObject.connectSlotsByName(TakeTestWindow)

def retranslateUi(self, TakeTestWindow):

\_translate = QtCore.QCoreApplication.translate

TakeTestWindow.setWindowTitle(\_translate("TakeTestWindow", "Test\_Page"))

self.Option2Button.setText(\_translate("TakeTestWindow", "Option2"))

self.label\_2.setText(\_translate("TakeTestWindow", "Question"))

self.Option4Button.setText(\_translate("TakeTestWindow", "Option4"))

self.Option1Button.setText(\_translate("TakeTestWindow", "Option1"))

self.label.setText(\_translate("TakeTestWindow", "Question NO. 0"))

self.NextButton.setText(\_translate("TakeTestWindow", "Next"))

self.Option3Button.setText(\_translate("TakeTestWindow", "Option3"))

self.BackButton.setText(\_translate("TakeTestWindow", "Back"))

def back\_button\_clicked(self):

if self.count > 0:

self.count -= 2

self.RenderQ(self.count)

def get\_question\_data(self, data, subject\_list, count):

question = data.iloc[subject\_list[count], 0]

a = data.iloc[subject\_list[count], 1]

b = data.iloc[subject\_list[count], 2]

c = data.iloc[subject\_list[count], 3]

d = data.iloc[subject\_list[count], 4]

return question, a, b, c, d

def get\_selected\_answer(self):

if self.Option1Button.isChecked():

return "A"

elif self.Option2Button.isChecked():

return "B"

elif self.Option3Button.isChecked():

return "C"

elif self.Option4Button.isChecked():

return "D"

else:

return ""

def checkAns(self):

if self.selected\_answer == self.ans[self.count - 1]:

self.marks += 4

elif self.selected\_answer == "":

pass

self.marks = min(self.marks, 100)

def nextButtonClicked(self, TakeTestWindow):

if not self.finished:

self.NextButton.setEnabled(False)

self.selected\_answer = self.get\_selected\_answer()

self.checkAns()

self.RenderQ(self.count)

else:

self.finish(TakeTestWindow)

def RenderQ(self, count):

english\_file = r"G:\reps\minor-project\Datasets\english\_questions.xlsx"

mathematics\_file = r"G:\reps\minor-project\Datasets\Mathematics\_questions.xlsx"

social\_studies\_file = r"G:\reps\minor-project\Datasets\Social\_Studies\_questions.xlsx"

science\_file = r"G:\reps\minor-project\Datasets\Science\_questions.xlsx"

logical\_reasoning\_file = r"G:\reps\minor-project\Datasets\Logical\_Reasoning\_questions.xlsx"

computer\_file = r"G:\reps\minor-project\Datasets\Computer\_questions.xlsx"

data = None

question, a, b, c, d = "", "", "", "", ""

if len(self.English) > 0:

data = pd.read\_excel(english\_file)

question, a, b, c, d = self.get\_question\_data(data, self.English, count)

elif len(self.Mathematics) > 0:

data = pd.read\_excel(mathematics\_file)

question, a, b, c, d = self.get\_question\_data(data, self.Mathematics, count)

elif len(self.Social\_Studies) > 0:

data = pd.read\_excel(social\_studies\_file)

question, a, b, c, d = self.get\_question\_data(data, self.Social\_Studies, count)

elif len(self.Science) > 0:

data = pd.read\_excel(science\_file)

question, a, b, c, d = self.get\_question\_data(data, self.Science, count)

elif len(self.Logical\_Reasoning) > 0:

data = pd.read\_excel(logical\_reasoning\_file)

question, a, b, c, d = self.get\_question\_data(data, self.Logical\_Reasoning, count)

elif len(self.Computer) > 0:

data = pd.read\_excel(computer\_file)

question, a, b, c, d = self.get\_question\_data(data, self.Computer, count)

else: self.label\_2.setText("error")

if data is not None:

self.count += 1

self.label.setText(f"Question NO. {self.count}")

self.label\_2.setText(question)

self.Option1Button.setText(str(a))

self.Option2Button.setText(str(b))

self.Option3Button.setText(str(c))

self.Option4Button.setText(str(d))

else:

self.label\_2.setText("error")

self.NextButton.setEnabled(True)

if self.count >= 25:

self.finished = True

self.NextButton.setText("Finish")

def addmarkstodatabase(self):

if len(self.English) > 0:

statement= f"UPDATE User\_Marks SET English = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

elif len(self.Mathematics) > 0:

statement= f"UPDATE User\_Marks SET Mathematics = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

elif len(self.Social\_Studies) > 0:

statement= f"UPDATE User\_Marks SET Social\_Studies = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

elif len(self.Science) > 0:

statement= f"UPDATE User\_Marks SET Science = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

elif len(self.Logical\_Reasoning) > 0:

statement= f"UPDATE User\_Marks SET Logical\_Reasoning = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

elif len(self.Computer) > 0:

statement= f"UPDATE User\_Marks SET Computer = {self.marks} WHERE User\_id = {User\_id}"

self.cur.execute(statement)

con.commit()

else: self.label\_2.setText("error")

def finish(self, TakeTestWindow):

from Results import Ui\_ResultWindow

result=self.marks

self.addmarkstodatabase()

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_ResultWindow()

self.ui.setupUi(self.window)

if len(self.English) > 0:

self.ui.EngMarksLbl.setText(f"{result}")

elif len(self.Mathematics) > 0:

self.ui.MathMarksLbl.setText(f"{result}")

elif len(self.Social\_Studies) > 0:

self.ui.SstMarksLbl.setText(f"{result}")

elif len(self.Science) > 0:

self.ui.SciLbl.setText(f"{result}")

elif len(self.Logical\_Reasoning) > 0:

self.ui.LgRnLbl.setText(f"{result}")

elif len(self.Computer) > 0:

self.ui.CompLbl.setText(f"{result}")

else: self.label\_2.setText("error")

self.window.show()

TakeTestWindow.hide()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

TakeTestWindow = QtWidgets.QMainWindow()

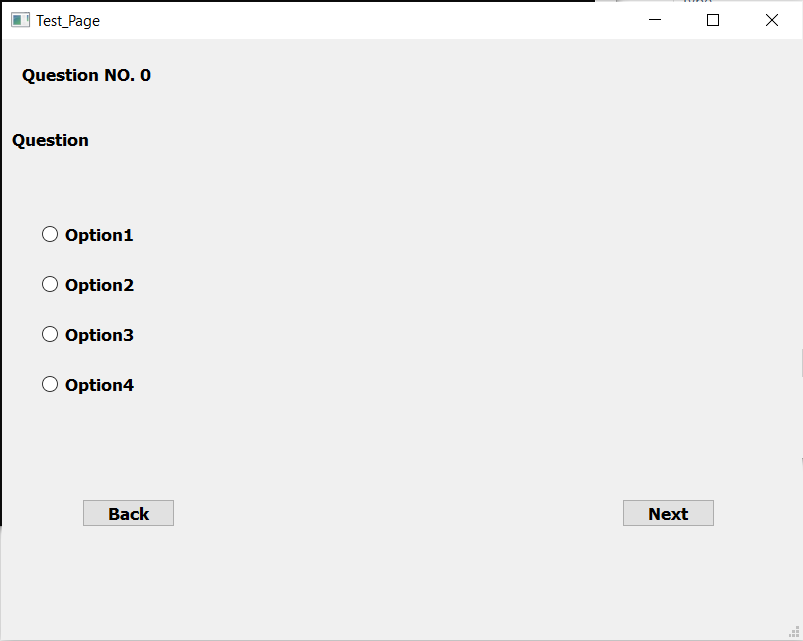
ui = Ui\_TakeTestWindow()

ui.setupUi(TakeTestWindow)

TakeTestWindow.show()

sys.exit(app.exec\_())

**output:**



**Results page:**

from PyQt5 import QtCore, QtGui, QtWidgets

from menu\_page import Ui\_MenuWindow

class Ui\_ResultWindow(object):

def setupUi(self, ResultWindow):

ResultWindow.setObjectName("ResultWindow")

ResultWindow.resize(800, 600)

self.centralwidget = QtWidgets.QWidget(ResultWindow)

self.centralwidget.setObjectName("centralwidget")

self.MenuButton = QtWidgets.QPushButton(self.centralwidget)

self.MenuButton.setGeometry(QtCore.QRect(320, 390, 93, 28))

self.MenuButton.setObjectName("MenuButton")

self.MenuButton.clicked.connect(lambda: self.backtomenu(ResultWindow))

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(90, 100, 50, 21))

self.label.setObjectName("label")

self.EngMarksLbl = QtWidgets.QLabel(self.centralwidget)

self.EngMarksLbl.setGeometry(QtCore.QRect(220, 100, 55, 16))

self.EngMarksLbl.setText("")

self.EngMarksLbl.setObjectName("EngMarksLbl")

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(90, 170, 80, 20))

self.label\_2.setObjectName("label\_2")

self.MathMarksLbl = QtWidgets.QLabel(self.centralwidget)

self.MathMarksLbl.setGeometry(QtCore.QRect(220, 170, 55, 16))

self.MathMarksLbl.setText("")

self.MathMarksLbl.setObjectName("MathMarksLbl")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(90, 240, 90, 16))

self.label\_3.setObjectName("label\_3")

self.SstMarksLbl = QtWidgets.QLabel(self.centralwidget)

self.SstMarksLbl.setGeometry(QtCore.QRect(220, 240, 55, 16))

self.SstMarksLbl.setText("")

self.SstMarksLbl.setObjectName("SstMarksLbl")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(450, 100, 55, 16))

self.label\_4.setFrameShape(QtWidgets.QFrame.NoFrame)

self.label\_4.setObjectName("label\_4")

self.SciLbl = QtWidgets.QLabel(self.centralwidget)

self.SciLbl.setGeometry(QtCore.QRect(600, 100, 55, 16))

self.SciLbl.setText("")

self.SciLbl.setObjectName("SciLbl")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(450, 170, 110, 16))

self.label\_5.setObjectName("label\_5")

self.LgRnLbl = QtWidgets.QLabel(self.centralwidget)

self.LgRnLbl.setGeometry(QtCore.QRect(600, 170, 55, 16))

self.LgRnLbl.setText("")

self.LgRnLbl.setObjectName("LgRnLbl")

self.label\_7 = QtWidgets.QLabel(self.centralwidget)

self.label\_7.setGeometry(QtCore.QRect(450, 240, 55, 16))

self.label\_7.setObjectName("label\_7")

self.CompLbl = QtWidgets.QLabel(self.centralwidget)

self.CompLbl.setGeometry(QtCore.QRect(600, 240, 55, 16))

self.CompLbl.setText("")

self.CompLbl.setObjectName("CompLbl")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(70, 30, 91, 16))

font = QtGui.QFont()

font.setPointSize(10)

font.setBold(True)

font.setWeight(75)

self.label\_6.setFont(font)

self.label\_6.setObjectName("label\_6")

ResultWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(ResultWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 26))

self.menubar.setObjectName("menubar")

ResultWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(ResultWindow)

self.statusbar.setObjectName("statusbar")

ResultWindow.setStatusBar(self.statusbar)

self.retranslateUi(ResultWindow)

QtCore.QMetaObject.connectSlotsByName(ResultWindow)

def retranslateUi(self, ResultWindow):

\_translate = QtCore.QCoreApplication.translate

ResultWindow.setWindowTitle(\_translate("ResultWindow", "Result Page"))

self.MenuButton.setText(\_translate("ResultWindow", "Go To Menu"))

self.label.setText(\_translate("ResultWindow", "English:"))

self.label\_2.setText(\_translate("ResultWindow", "Mathematics:"))

self.label\_3.setText(\_translate("ResultWindow", "Social Studies:"))

self.label\_4.setText(\_translate("ResultWindow", "Science"))

self.label\_5.setText(\_translate("ResultWindow", "Logical Reasoning:"))

self.label\_7.setText(\_translate("ResultWindow", "Computer"))

self.label\_6.setText(\_translate("ResultWindow", "Results:"))

def backtomenu(self, ResultWindow):

self.window = QtWidgets.QMainWindow()

self.ui = Ui\_MenuWindow()

self.ui.setupUi(self.window)

self.window.show()

ResultWindow.hide()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

ResultWindow = QtWidgets.QMainWindow()

ui = Ui\_ResultWindow()

ui.setupUi(ResultWindow)

ResultWindow.show()

sys.exit(app.exec\_())

**output:**