Software Design Description

(SDD)

for

<ECNG 2005 CSL Group 4 Project>

at The Department of Electrical and Computer Engineering,

The University of the West Indies

St Augustine Campus

Trinidad

Prepared by:

<Kashanna Joseph, Adesh Goordeen, Kyle Ali, Aliya Miller, Raja Bissesar>

<Tuesday 26th March 2024>

Document Control

Title: Software Design Description

Version: 0.2

Date: 26/03/2024

Author(s): Kashanna Joseph, Adesh Goordeen, Kyle Ali, Aliya Miller, Raja Bissessar

Document Signoff

|  |  |  |  |
| --- | --- | --- | --- |
| Nature of Signoff | Person | Signature with Date | Role |
| Author        Reviewees |  |  |  |

*N.B. Signoff happens when moving from draft to a stable version. Reviewers can recommend returning to another draft or moving to a stable version. In the case of an StRS, this document is customer/user/stakeholder facing. Therefore, it needs their signoff to move to a stable version.*

Document Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Change Details |
| 09/03/2024 | 0.1 | Kashanna Joseph | Edited to the cover page to reflect the correct information regarding the project |
| 13/03/2024 | 0.1 | Aliya Miller | Added Section 5 “Restrictions, limitations and constraints” |
| 14/03/2024 | 0.1 | Kashanna Joseph | Updated “Sections 1.1 Goals and Objectives, 1.2 Statement of scope, 1.3 Software Context and 1.4 Major Constraints” |
| 14/03/2024 | 0.1 | Adesh Goordeen | Updated “Sections 2 Data Design, 2.1 Internal Software Data Structure, 2.2 Temporary data structure and 2.3 Database description” |
| 16/03/2024 | 0.1 | Kyle Ali | Updated “Section 3 Architectural and component-level design” |
| 22/03/2024 | 0.2 | Kyle Ali | Added Figure 2 to “Section 3.1.1 Architecture Diagram” |
| 22/03/2024 | 0.2 | Kyle Ali | Added Figure 3 to “Section 3.2.3.2 Algorithmic model (e.g., PDL) |
| 23/03/2024 | 0.2 | Aliya Miller | Added Section 4.2 “Interface design rules” |
| 23/03/2024 | 0.2 | Aliya Miller | Updated Section 5 “Restrictions, limitations and constraints” |
| 26/03/2024 | 0.2 | Kashanna Joseph | Updated “Sections 4.1 Description of the user interface and 4.1.1 Screen images” |

[1 Introduction 4](#_Toc299365197)

[2 Data design 5](#_Toc906144679)

[2.1 Internal software data structure 5](#_Toc1996587109)

[2.2 Temporary data structure 6](#_Toc1516596631)

[2.3 Database description 6](#_Toc1607551163)

[3 Architectural and component-level design 6](#_Toc1901410473)

[3.1 Program Structure 7](#_Toc132664305)

[3.1.1 Architecture diagram 7](#_Toc683584711)

[3.2 Description for Component n 9](#_Toc1782364715)

[3.2.1 Processing narrative (PSPEC) for component n 10](#_Toc522910779)

[3.2.2 Component n interface description. 12](#_Toc1880214224)

[3.2.3 Sub-Component n.m processing detail 14](#_Toc1882333366)

[3.3 Software Interface Description 28](#_Toc1505335117)

[3.3.1 External machine interfaces 29](#_Toc193367821)

[3.3.2 External system interfaces 29](#_Toc1313163036)

[3.3.3 Human interface 29](#_Toc302829131)

[4 User interface design 30](#_Toc1802126853)

[4.1 Description of the user interface 30](#_Toc1695707836)

[4.1.1 Screen images 31](#_Toc903195754)

[4.2 Interface design rules 35](#_Toc1657434044)

[5 Restrictions, limitations, and constraints 36](#_Toc776321974)

[6 Testing Issues 36](#_Toc933973960)

[6.1 Classes of tests 36](#_Toc153146563)

[6.2 Expected software response 36](#_Toc1307671527)

[6.3 Performance bounds 36](#_Toc1422452510)

[6.4 Identification of critical components 36](#_Toc337705029)

### 1 Introduction

1.1 Goals and objectives

The goals and objectives of this project are as follows:

(1) The group wishes to develop a digitized system of the mental health screening forms provided to us by Ms. Harripersad from Princes Town West Secondary School

(2) The software would be in the form of a local computer application that we will hand over to Ms. Harripersad once completion is done

(3) The application would be user friendly and contain features such as a general information page (where general information of the students will be filled out), an instruction page for each screening test, three different screening tests, preview of the student’s score for each test, and a print function so that the guidance counselor may print the students results to store them in a file as she wishes.

1.2 Statement of scope

The scope of this project involves the development of a software system to the mental health screening forms that guidance counsellor uses at Princes Town West Secondary. The software would be in the form of a local computer app that will contain all the information from the forms provided to us. These digitized forms will allow for students to be able to fill out information easily and then based on their answers, there will be a generated score which the guidance counsellor would use to assess the student’s current mental health state. The student’s information is stored on a database so that if the guidance counsellor wishes to administer the test to the student again and compare scores, she may be able to do so. This software is an easier and enhanced approach for inputting, accessing and storing data whilst keeping the information private, which would greatly help the guidance counsellor going forward.

1.3 Software context

The Ministry of Education in Trinidad and Tobago has implemented the use of the screening forms to better evaluate the mental wellbeing of students in secondary schools. These forms are the Stress form, Depression form and Self-esteem form. Currently, guidance counsellors print these forms and give students to fill them out to which they then tally the score. This application can be used as an alternative for filling these forms and saving the data. This alternative to manually filling these forms also eliminate the waiting time for retrieving and storing student data.

1.4 Major constraints

Some major constraints are:

1. The time in which we are given to design, test and implement the software
2. The specifications of the application. It must be compatible with the computer which we are transferring it to (the guidance counsellor’s computer)
3. The application must be user friendly and age appropriate for individuals aged 12-21
4. There must be a printing function within the application so that the guidance counsellor can print the results for hard copy storage
5. Not all tests would be taken at the same time, so the product must be developed in such a way that it can allow for only one test to be taken at a time.
6. If possible, a printed version of the student’s answers to the questions in the screening form should be available for viewing.

# 2 Data design

In the application, the design has three structured codes to represent the requirements. The application consists of the user interface where users can navigate throughout the application, the relevant forms integrated, a preview of the application’s contents and lastly the database. These are termed internal, global, and temporary data structures.

## 2.1 Internal software data structure

The data from the database will be used by the user. The data store can be printed for further processing by the ministry. The data is passed from the main user interface to the forms, and then to the database as seen in figure 1.

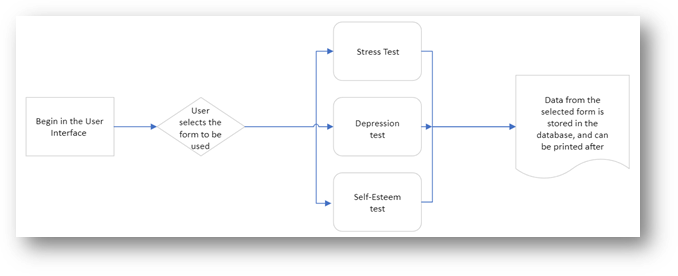


Figure 1: A Simplified Flow chart of the Internal Software data Structure.

## 2.2 Temporary data structure

The temporary data structure was used to temporarily retain the information in the user interface before storing the data on the database. This was implemented with the use of the complier code implemented in the application as the preview tab.

## 2.3 Database description

Information that is entered into the application is soon after stored in the database to be accessed later. The table below shows the breakdown of the data before being stored in the database.

Table 1: Breakdown of Data Before Being Stored in the Database

|  |  |  |
| --- | --- | --- |
| Data stored | Data label | Data type |
| User information | Name, Date, Gender, age, School. | Strings, integers |
| Stress form | Stress form | Classes/loop |
| Depression form | Depression form | Classes/loop |
| Self-Esteem form | Self-Esteem form | Classes/loop |
| Preview | Preview tab | Classes/loop |
| Database | Database | classes |

# 3 Architectural and component-level design

The system software follows a modular architecture, where each major component (self-esteem, depression, stress, general information, preview, and database) is implemented as a separate class and user interface. The main application window serves as the container, and a stacked widget is used to switch between the different components. This modular approach promotes code organization, reusability, and extensibility, allowing for easier maintenance and future enhancements.

## 3.1 Program Structure

Initially, a monolithic architecture was considered, with all components integrated into a single user interface. However, this approach was deemed less maintainable and extensible. The modular architecture was chosen to improve code organization, reusability, and future extensibility. The system software is designed using a modular architecture, where each major component (self-esteem, depression, stress, general information, preview, and database) is implemented as a separate class and user interface. This modular approach promotes code organization, reusability, and extensibility. The main application window serves as the container for these components, and a stacked widget is used to switch between them. The stacked widget allows for easy navigation between the different components by displaying one component at a time, while the others remain hidden. This modular architecture was chosen over a monolithic architecture, where all components would be integrated into a single user interface. The monolithic approach was deemed less maintainable and extensible, making it more challenging to modify or add new features in the future. By separating the components into individual modules, the code becomes more organized and easier to manage. Each component can be developed, tested, and maintained independently, without affecting the functionality of other components. This modularity also allows for better code reuse, as components can be easily shared or integrated into other projects if needed. The architecture diagram illustrates the main application window acting as the container for the stacked widget, which holds the individual components. The stacked widget manages the switching between components, allowing users to navigate through the different sections of the application, such as the self-esteem test, depression test, stress test, general information, preview, and database.

### 3.1.1 Architecture diagram

The architecture diagram illustrates the main application window acting as the container for the stacked widget, which holds the individual components. The stacked widget manages the switching between components, allowing users to navigate through the different sections of the application, such as the self-esteem test, depression test, stress test, general information, preview, and database.

A diagram of a company

Description automatically generated

Figure 2: The Architecture of the Program

### 3.1.2 Alternatives

A monolithic architecture was considered at first, where all components would be integrated into a single user interface. However, this approach was not feasible, making it more challenging to modify or add new features in the future. Another alternative considered was a client-server architecture, where the user interface would act as the client, communicating with a separate server component responsible for data processing and storage. While this approach could potentially allow for multi-user support and centralized data management, it was deemed unnecessarily complex for the current scope of the system software, which is primarily designed for single-user desktop usage.

The modular architecture presented in Section 3.1.1 was selected for several reasons:

1. Code organization: By separating the components into individual modules, the code becomes more organized and easier to manage. Each component can be developed, tested, and maintained independently, without affecting the functionality of other components.

2. Reusability: The modular design promotes code reuse, as individual components can be easily shared or integrated into other projects if needed.

3. Extensibility: Adding new features or components to the software is more straightforward in a modular architecture, as it involves creating a new module rather than modifying the existing monolithic codebase.

4. Maintainability: With separate components, bug fixing, and updates can be targeted to specific modules, reducing the risk of introducing unintended changes or side effects in other parts of the software.

5. Scalability: As the project grows or requirements change, the modular architecture allows for better scalability by enabling the addition or modification of individual components without disrupting the entire system.

While the client-server architecture could provide benefits in terms of multi-user support and centralized data management, the added complexity was deemed unnecessary for the current scope of the system software, which is designed as a desktop application for single-user usage. The modular architecture strikes a balance between simplicity and flexibility, making it the most suitable choice for the system software project.

## 3.2 Description for Component

The software consists of the following main components:

1. SelfEsteemTab

2. DepressionTab

3. StressTab

4. GeneralTab

5. PreviewTab

6. DataBaseTab

SelfEsteemTab - The SelfEsteemTab component is responsible for administering and displaying the self-esteem test. It presents a series of questions related to self-esteem, and users can provide their responses.

DepressionTab - The DepressionTab component handles the depression test. It consists of multiple questions, each with two alternative answers. Users can select their preferred answer for each question by clicking the corresponding button.

StressTab - The StressTab component is dedicated to the stress test. It displays a set of stress-related questions, and users can rate their responses.

GeneralTab - The GeneralTab component allows users to enter general information, such as their name, age, sex, school, and the date of the test. This information is used to identify and contextualize the test results.

PreviewTab - The PreviewTab component provides a summary of the user's responses and test results. It displays the general information entered, as well as the scores or ratings from the self-esteem, depression, and stress tests.

DataBaseTab - The DataBaseTab component facilitates data storage and retrieval. It includes a table widget that displays previously recorded test results and user information. Users can add new records or retrieve existing ones from the database.

Each of these components is integrated into the main application window through a stacked widget, which allows for seamless navigation between them using buttons or menu options. The components are designed to work together, providing a comprehensive solution for administering and evaluating psychological tests related to self-esteem, depression, and stress.

### 3.2.1 Processing narrative (PSPEC) for component n

The processing narrative (PSPEC) for each component in the MindMeter, is as follows:

Processing narrative (PSPEC) for SelfEsteemTab:

1. The SelfEsteemTab component is displayed in the stacked widget.

2. The component presents a series of self-esteem-related questions, each with a slider and spin box for user input.

3. The user moves the slider or adjusts the spin box value to indicate their response to each question.

4. As the user provides input, the component validates and stores the responses internally.

5. When the user navigates to another component (e.g., DepressionTab), the SelfEsteemTab component saves the user's responses for future processing or display.

Processing narrative (PSPEC) for DepressionTab:

1. The DepressionTab component is displayed in the stacked widget.

2. The component presents multiple depression-related questions, each with two alternative answer options.

3. For each question, the user clicks one of the two buttons to select their preferred answer.

4. The component registers the user's selection and stores the response internally.

5. When the user navigates to another component, the DepressionTab component saves the user's responses for future processing or display.

Processing narrative (PSPEC) for StressTab:

1. The StressTab component is displayed in the stacked widget.

2. The component displays a set of stress-related questions, each with a slider and spin box for user input.

3. The user moves the slider or adjusts the spin box value to indicate their response to each question.

4. As the user provides input, the component validates and stores the responses internally.

5. When the user navigates to another component, the StressTab component saves the user's responses for future processing or display.

Processing narrative (PSPEC) for GeneralTab:

1. The GeneralTab component is displayed in the stacked widget.

2. The component presents input fields for the user to enter their general information, such as name, age, sex, school, and the date of the test.

3. The user fills in the required information in the respective input fields.

4. As the user provides input, the component validates and stores the information internally.

5. When the user navigates to another component, the GeneralTab component saves the user's general information for future use or display.

Processing narrative (PSPEC) for PreviewTab:

1. The PreviewTab component is displayed in the stacked widget.

2. The component retrieves the user's general information from the GeneralTab component and displays it in the preview section.

3. The component retrieves the user's responses from the SelfEsteemTab, DepressionTab, and StressTab components and calculates or formats the results for display.

4. The component presents the user's general information and test results in a readable format for preview.

5. If the user navigates to another component, the PreviewTab component remains available for future reference or printing.

Processing narrative (PSPEC) for DataBaseTab:

1. The DataBaseTab component is displayed in the stacked widget.

2. The component displays a table widget containing previously recorded test results and user information.

3. The user can add a new record by entering the required information (name, age, sex, school, test results) and clicking an "Add" button or similar control.

4. The component validates the input and stores the new record in the database.

5. The user can retrieve existing records by searching or filtering the table based on specific criteria (e.g., name, age, school).

6. The component retrieves and displays the requested records from the database in the table widget.

These processing narratives provide an overview of how each component functions and interacts with the user, as well as the flow of data within the PsycheEval software.

### 3.2.2 Component n interface description.

The detailed description of the input and output interfaces for each component in the MindMeter software, is as follows:

1. SelfEsteemTab

Input Interface:

- A series of sliders and spin boxes, one for each self-esteem question, allowing the user to provide a numerical rating as input.

Output Interface:

- The user's responses to the self-esteem questions are stored internally for further processing or display.

2. DepressionTab

Input Interface:

- For each depression question, two buttons representing the alternative answer options, allowing the user to select one option as input.

Output Interface:

- The user's selected answers for the depression questions are stored internally for further processing or display.

3. StressTab

Input Interface:

- A series of sliders and spin boxes, one for each stress-related question, allowing the user to provide a numerical rating as input.

Output Interface:

- The user's responses to the stress questions are stored internally for further processing or display.

4. GeneralTab

Input Interface:

- Text input fields for the user to enter their name, age, sex, school, and the date of the test.

Output Interface:

- The user's general information is stored internally for further use or display.

5. PreviewTab

Input Interface:

- This component does not have a direct user input interface; it retrieves data from other components.

Output Interface:

- Displays the user's general information and test results (self-esteem, depression, stress) in a readable format for preview.

6. DataBaseTab

Input Interface:

- A table widget where the user can add new records by entering information (name, age, sex, school, test results) and clicking an "Add" button or similar control.

- Controls or filters for searching or retrieving existing records from the database.

Output Interface:

- The table widget displays the retrieved records from the database, including the user's general information and test results.

These interfaces define how the user interacts with each component, providing input through various controls (sliders, spin boxes, buttons, text fields) and receiving output in the form of stored data or visual representations of the test results and user information.

### 3.2.3 Sub-Component n.m processing detail

The detailed algorithmic description for some of the sub-components within each main component of the MindMeter software, presented in Section 3.2.3, is as follows:

1. SelfEsteemTab

1.1 SelfEsteemQuestion Sub-Component

Interface description:

- Input: A slider and spin box for the user to provide a numerical rating.

- Output: The user's rating for the specific self-esteem question.

1.2 Algorithmic model:

1. Display the self-esteem question text and the slider/spin box controls.

2. Listen for user input from the slider or spin box.

3. Validate the user's input to ensure it falls within the allowed range.

4. Store the user's rating for the question.

2. DepressionTab

2.1 DepressionQuestion Sub-Component

Interface description:

- Input: Two buttons representing the alternative answer options.

- Output: The user's selected answer for the specific depression question.

2.2 Algorithmic model:

1. Display the depression question text and the two answer option buttons.

2. Listen for user input by clicking one of the answer buttons.

3. Store the user's selected answer for the question.

3. StressTab

3.1 StressQuestion Sub-Component

Interface description:

- Input: A slider and spin box for the user to provide a numerical rating.

- Output: The user's rating for the specific stress question.

3.2 Algorithmic model:

1. Display the stress question text and the slider/spin box controls.

2. Listen for user input from the slider or spin box.

3. Validate the user's input to ensure it falls within the allowed range.

4. Store the user's rating for the question.

4. GeneralTab

4.1 UserInfoInput Sub-Component

Interface description:

- Input: Text input fields for the user to enter their name, age, sex, school, and the

date of the test.

- Output: The user's general information.

4.2 Algorithmic model:

1. Display the input fields for name, age, sex, school, and date.

2. Listen for user input in each field.

3. Validate the user's input for each field (e.g., age should be a number, date

should be in a valid format).

4. Store the user's general information.

5. PreviewTab

5.1 ResultDisplay Sub-Component

Interface description:

- Input: The user's general information and test results from other components.

- Output: Visual representation of the user's information and test results.

5.2 Algorithmic model:

1. Retrieve the user's general information from the GeneralTab component.

2. Retrieve the user's self-esteem test results from the SelfEsteemTab component.

3. Retrieve the user's depression test results from the DepressionTab component.

4. Retrieve the user's stress test results from the StressTab component.

5. Format and display the user's general information and test results in a readable

format.

6.1 DataBaseTab

6.1.1 DataEntry Sub-Component

Interface description:

- Input: The user's general information and test results.

- Output: A new record added to the database.

6.1.2 Algorithmic model

1. Display input fields or controls for the user to enter their general information and

test results.

2. Listen for user input in each field or control.

3. Validate the user's input for each field or control.

4. Create a new record with the user's information and test results.

5. Add the new record to the database.

6.2 DataRetrieval Sub-Component

6.2.1 Interface description:

- Input: Search criteria or filters provided by the user.

- Output: Existing records from the database matching the search criteria.

6.2.2 Algorithmic model:

1. Display controls or filters for the user to specify search criteria (e.g., name, age,

school).

2. Listen for user input in the search controls or filters.

3. Query the database to retrieve records matching the search criteria.

4. Display the retrieved records in the table widget.

These algorithmic descriptions provide a detailed overview of the processing logic and data flow within each sub-component, including user input handling, data validation, storage, retrieval, and output formatting.

3.2.3.1 Interface description  
 The description of sub-components with it inputs and outputs presented, following the format specified in Section 3.2.3 of the SDD document, is as follows:

1. PreviewTab

1.1 ResultDisplay Sub-Component

Interface description Input:

- The user's general information (name, age, sex, school, date) from the

GeneralTab component.

- The user's self-esteem test results from the SelfEsteemTab component.

- The user's depression test results from the DepressionTab component.

- The user's stress test results from the StressTab component.

Output:

- Visual representation of the user's general information and test results in a

readable format.

2. Restrictions/limitations

- The ResultDisplay sub-component relies on the availability and

correctness of data from other components (GeneralTab, SelfEsteemTab,

DepressionTab, StressTab).

- The visual representation of the results may be limited by the capabilities

of the user interface framework (PyQt5) and the available widgets or

rendering options.

3. Local data structures

- A data structure (e.g., a dictionary or a class instance) to store the user's

general information and test results retrieved from other components.

- Formatting rules or templates for presenting the data in a readable format.

4. Performance issues

- No significant performance issues are expected, as the ResultDisplay

sub-component primarily deals with data formatting and display, which is

typically, a lightweight operation.

5. Design constraints

- The design of the ResultDisplay sub-component needs to accommodate

various data formats and structures from the different test components

(self-esteem, depression, stress).

- The visual representation of the results should follow user interface design

guidelines and principles for readability and accessibility.

The inputs and outputs of the ResultDisplay sub-component within the PreviewTab component are outlined above. It also provides additional details regarding the algorithmic model, restrictions/limitations, local data structures, performance issues, and design constraints specific to this sub-component.

*3.2.3.2 Algorithmic model (e.g., PDL)*  
 The pseudocode listing for sub-component m, is as follows:

ResultDisplay Sub-Component:

// Retrieve user's general information

userInfo = GeneralTab.getUserInfo()

// Retrieve self-esteem test results

selfEsteemResults = SelfEsteemTab.getTestResults()

// Retrieve depression test results

depressionResults = DepressionTab.getTestResults()

// Retrieve stress test results

stressResults = StressTab.getTestResults()

// Format and display user's general information

displayUserInfo(userInfo)

// Format and display self-esteem test results

displaySelfEsteemResults(selfEsteemResults)

// Format and display depression test results

displayDepressionResults(depressionResults)

// Format and display stress test results

displayStressResults(stressResults)

Function displayUserInfo(userInfo):

// Extract individual fields from userInfo

name = userInfo.name

age = userInfo.age

sex = userInfo.sex

school = userInfo.school

date = userInfo.date

// Display the user's general information in the desired format

print("Student Name: " + name)

print("Age: " + age)

print("Sex: " + sex)

print("School: " + school)

print("Date: " + date)

Function displaySelfEsteemResults(selfEsteemResults):

// Iterate over the self-esteem test results

for result in selfEsteemResults:

questionText = result.questionText

userRating = result.userRating

// Display the question text and user's rating

print(questionText + ": " + userRating)

// Optionally, calculate and display an overall self-esteem score

overallScore = calculateSelfEsteemScore(selfEsteemResults)

print("Overall Self-Esteem Score: " + overallScore)

// Similar functions for displaying depression and stress test results

// ... (omitted for brevity)

A diagram of a diagram

Description automatically generated with medium confidence

Figure 2: Flow Diagram of the Application

#### 

#### 3.2.3.3 Restrictions/limitations

The ResultDisplay sub-component relies on the availability and correctness of data from other components (GeneralTab, SelfEsteemTab, DepressionTab, StressTab). The visual representation of the results may be limited by the capabilities of the user interface framework (PyQt5) and the available widgets or rendering options. For proper operation of the ResultDisplay sub-component, the following external environment and infrastructure requirements must be met:

1. Dependency on other components:

- The GeneralTab component must be correctly implemented and operational,

providing accurate user general information (name, age, sex, school, date).

- The SelfEsteemTab component must be correctly implemented and

operational, providing accurate self-esteem test results.

- The DepressionTab component must be correctly implemented and

operational, providing accurate depression test results.

- The StressTab component must be correctly implemented and operational,

providing accurate stress test results.

2. User interface framework:

- The PyQt5 library and its dependencies must be installed and properly

configured on the target system.

- The PyQt5 framework must support the required widgets and rendering

capabilities necessary for displaying the user's general information and test

results in the desired format.

3. System requirements:

- The target system must meet the minimum hardware and software

requirements for running the MindMeter application and the PyQt5

framework. The minimum hardware requirements were determined to be 4 - 8

GB ram, 1.32 GHz (could be intel for turbo clocking to 2.4 GHz), 256GB.

- Sufficient memory and processing power should be available to handle the

rendering and display of the user interface components.

4. Operating system compatibility:

- The PsycheEval application, including the ResultDisplay sub-component,

must be compatible with the target operating system (e.g., Windows, macOS,

Linux).

- Any platform-specific dependencies or compatibility issues should be

addressed to ensure proper functionality.

5. User interface design guidelines:

- The visual representation of the results should adhere to established user

interface design guidelines and principles for readability, accessibility, and

usability.

- Appropriate font sizes, colors, and layout should be used to ensure a positive

user experience.

The external environment and infrastructure requirements were met, however the device used by the Guidance Officer was determined to be unreliable. The ResultDisplay sub-component can operate correctly and provide the intended functionality of displaying the user's general information and test results in a readable format within the MindMeter application, using the print function the data is converted to the pdf file type. The pdf file type allows for the data to be transferred from the application and then be printed to be stored as hard-copy documents.

*3.2.3.4 Local data structures*  
 the local data structures that could be used within the ResultDisplay sub-component were determined to be:

1. User Information Data Structure:

- A dictionary or a class instance to store the user's general information retrieved from the GeneralTab component.

- Example structure:

userInfo = {

'name': 'John Doe',

'age': 25,

'sex': 'Male',

'school': 'University of Example',

'date': '2023-05-15'

}

2. Self-Esteem Results Data Structure:

- A list of dictionaries or class instances, where each element represents a self-esteem question and the user's rating.

- Example structure:

selfEsteemResults = [

{

'questionText': 'I feel confident about my abilities.',

'userRating': 4

},

{

'questionText': 'I have a positive attitude toward myself.',

'userRating': 3

},

# ... more questions and ratings

]

3. Depression Results Data Structure:

- A list of dictionaries or class instances, where each element represents a depression question and the user's selected answer.

- Example structure:

depressionResults = [

{

'questionText': 'I feel sad most of the time.',

'userAnswer': 'Yes'

},

{

'questionText': 'I have trouble concentrating.',

'userAnswer': 'No'

},

# ... more questions and answers

]

4. Stress Results Data Structure:

- A list of dictionaries or class instances, where each element represents a stress question and the user's rating.

- Example structure:

stressResults = [

{

'questionText': 'I feel overwhelmed by my responsibilities.',

'userRating': 2

},

{

'questionText': 'I have difficulty relaxing.',

'userRating': 4

},

# ... more questions and ratings

]

5. Formatting Rules or Templates:

- A dictionary or a set of predefined strings that define the formatting rules or templates for displaying the user's information and test results in a readable format.

- Example structure:

formatRules = {

'userInfo': 'Student Name: {name}\nAge: {age}\nSex: {sex}\nSchool: {school}\nDate: {date}',

'selfEsteemQuestion': '{questionText}: {userRating}',

'depressionQuestion': '{questionText}: {userAnswer}',

'stressQuestion': '{questionText}: {userRating}'

}

These data structures were used to store and organize the user's general information and test results retrieved from the respective components. The ResultDisplay sub-component can then be accessed to process the data structures to format and display the information according to the defined rules of the system application.

*3.2.3.5 Performance issues*  
 It was determined that the privacy of the student was of utmost importance, therefore the application was made to hold the information in the database. The data computed, the tallied scores for each test, from the system application can only be accessed in the application directly or when the print function is used to allow the Guidance Officer copies of the data computed for record keeping.

*3.2.3.6 Design constraints*  
 The design constraints considered in the development process were such that, the ResultDisplay sub-component can be designed and implemented in a way that aligns with the overall software architecture, ensures compatibility with existing components and frameworks, and provides flexibility for future enhancements or integration with external systems.

1. Data Structures:

- The data structures used by other components (GeneralTab, SelfEsteemTab,

DepressionTab, StressTab) to store and represent user information and test

results may constrain how the ResultDisplay sub-component accesses and

processes this data.

- If the data structures are not consistently defined or documented across components, it may introduce challenges in retrieving and formatting the data for display.

2. User Interface Framework (PyQt5):

- The capabilities and limitations of the PyQt5 framework could constrain the

design and implementation of the ResultDisplay sub-component.

- Certain visual rendering options, widget types, or layout management

features may not be available or may have limitations, affecting how the

user's information and test results are presented.

- Cross-platform compatibility and differences in behavior across operating

systems could also impact the design of the sub-component.

3. Operating System Features:

- Specific features or APIs provided by the underlying operating system may

influence the design of the ResultDisplay sub-component.

- For example, if the sub-component needs to interact with the file system or

handle printing functionality, the available OS-level APIs and their

compatibility across different operating systems could constrain the design

choices.

4. Input/Output (I/O) Handling:

- If the ResultDisplay sub-component needs to handle user input or provide

output beyond visual rendering (e.g., printing, file export), the design may be

constrained by the I/O capabilities and APIs available in the software

architecture.

- Limitations in I/O handling could restrict the sub-component's ability to provide certain features or functionalities.

5. Interoperable Systems:

- Although the current scope of the MindMeter software does not include interoperability with external systems, if such a requirement arises in the future, it could constrain the design of the ResultDisplay sub-component.

- For example, if the software needs to integrate with a reporting system or a data storage platform, the sub-component may need to adapt its design to accommodate the necessary data formats, communication protocols, or APIs required for interoperability.

6. Performance and Scalability:

- While not a major concern for the current version of the software, if future requirements demand higher performance or scalability, the design of the ResultDisplay sub-component may need to be optimized or refactored.

- This could include considerations such as efficient data processing, caching mechanisms, or asynchronous rendering techniques.

7. Accessibility and Usability:

- The overall software design may impose certain accessibility and usability guidelines or constraints that the ResultDisplay sub-component must adhere to.

- This could include guidelines related to font sizes, color schemes, layout arrangements, or support for assistive technologies, which could influence the sub-component's design decisions.

## 3.3 Software Interface Description

The system software is a desktop application designed for single-user usage, and it does not have direct interfaces to external systems or machines. However, it does provide interfaces for user interaction and potential data exchange with external sources.

External Machine Interfaces - The software does not interact with external machines or devices

directly.

External System Interfaces - The software does not currently interface with external systems or networks. However, the DataBaseTab component could potentially be extended to support data exchange with external databases or storage systems in the future.

Human Interface - The system software provides a graphical user interface (GUI) for human interaction, developed using the PyQt5 framework. The main application window serves as the primary interface, allowing users to navigate between different components and complete various tasks.

The human interface includes the following elements:

1. Main Application Window:

- The main window acts as the container for all the components and provides navigation controls (buttons, menus) for users to switch between different sections of the application.

2. Component-specific User Interfaces:

- Each component (SelfEsteemTab, DepressionTab, StressTab, GeneralTab, PreviewTab, DataBaseTab) has its own user interface designed for specific tasks.

- These interfaces include various UI, allowing users to input data, provide responses,

and interact with the application.

3. User Input and Feedback:

- Users can provide input through various controls.

- The application provides visual feedback and displays the user's responses and test results

through labels, previews, and tabular representations.

4. Data Storage and Retrieval:

- The DataBaseTab component allows users to store and retrieve test results and user

information through a table widget interface.

- Users can add new records by entering the required information.

- Users can also retrieve existing records by searching or filtering the table based on specific

criteria (e.g., name, age, school).

5. Printing:

- The software includes an option to print the user's general information and test results.

### 3.3.1 External machine interfaces

The system software itself does not directly interact with external machines or devices. However, the printing functionality may involve an interface with a connected printer or network printer. The software includes a "Print" option, which allows users to print the user's general information and test results displayed in the PreviewTab component.

### 3.3.2 External system interfaces

The current version of the system software does not have any external system interfaces. However, designing the software with extensibility in mind, particularly in the DataBaseTab component, it can be adapted to support interfaces with external databases, storage systems, or other relevant systems as needed.

### 3.3.3 Human interface

The system software provides a graphical user interface (GUI) for human interaction, developed using the PyQt5 framework. The main application window serves as the primary interface, allowing users to navigate between different components and complete various tasks.

# 4 User interface design

## 4.1 Description of the user interface

The user interface of the Mind Meter application consists of various tab classes such as the SelfEsteemTab, DepressionTab, StressTab, GeneralTab, PreviewTab and Database Tab. The layout includes a stackedWidget, that allows switching between different screens or tabs, a sidebar (MainWindow) and a top bar (MainWindow2) that handle navigation and the display of different sections such as General, Stress, Depression, etc. A restricted access widget (restrictedAccessWidget) is used for controlled navigation based on user roles. The swicthToCounsellorMode and switchtoStudentMode methods manage these transitions.

The startscreen class in the code defines the initial screen for a PyQt5 application. The user interface includes a large logo centered in the screen, and a customised start button positioned below the logo. The overall background of the start screen is set to a light blue colour. The start button is connected to emit the custom startClicked signal which is intended to trigger a transistion when the user clicks the button.

The GeneralTab class in a PyQt5 application is designed to collect general information from the user. It includes form fields for placing a student’s name, school, gender, date of birth and the current date. The user interface is visually appealing with clear labels, input boxes, and checkboxes that users can select once they have filled in the information. If a user is returning, the fields can be pre-populated and set to read-only. The class also includes functionalities to save the data and to handle the gender checkbox changes ensuring that only one gender option can be selected at a time. There is a “Save” button which when activated, checks that all the required fields are completed before proceeding to update a Preview Tab with the collected information. The data fields utilized QDateEdit widgets for easy date selection.

The StressTab class is part of the PyQt5 application that handles the stress assessment component. The tab features a list of ten predefined questions related to stress experiences. It is modelled off the hardcopy stress assessment provided to the group by Ms. Harripersad. It uses a tabbed widget to contain the questions and provides a button group for each question, allowing users to rate their experience on a scale. The responses are stored in a list, and a “save” button at the end of the questionnaire saves the data. The “save” button, when activated, checks if all questions are answered before allowing the user to proceed. The class also includes the functionality to update the Preview Tab with the responses and clear answers if needed.

The DepressionTab class is part of the PyQt5 application that creates a questionnaire component related to depression, allowing users to rate their experiences based on predefined question pairs that present contrasting statements. It is modelled off the hardcopy depression screening test provided to the group by Ms. Harripersad. It manages a tab widget split into two pages, each page containing a subset of questions. Each question has a unique identifier and answer options ranging from 1 to 5, with buttons to select the response. A “save” button captures all the responses and validates the completion before proceeding. The class also includes the functionality to update the Preview Tab. It also includes special handling for complex questions. The class is user friendly and has clear navigation.

The SelfEsteemTab class is a part of the PyQt5 application that creates a questionnaire component for the self-esteem assessment, featuring 20 questions. It is modelled off the hardcopy self-esteem assessment provided to the group by Ms. Harripersad. Each question has a unique identifier and a set of answer options ranging from 1 to 5, with buttons to select the response. The tab widget is split into two pages, with a “save” button on the second page to capture all responses. The class also includes the functionality to update the Preview Tab. It is user friendly and has clear navigation.

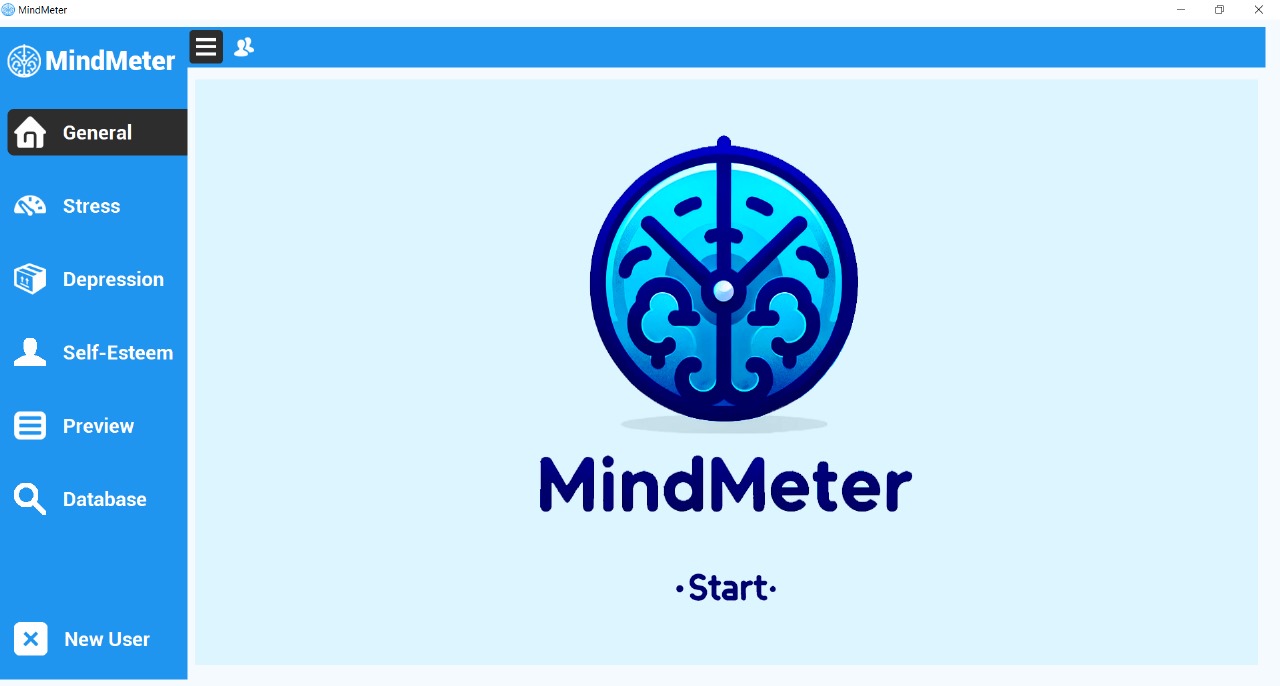
The PreviewTab class is a part of the PyQt5 application that serves as a summary and confirmation page for user input gathered from previous tabs. It displays the collected general information (name, school, gender, birth date, current date) and test scores (stress, depression, self-esteem) in a preview format, allowing users to review and confirm their entries before saving. The class also calculates the user’s age based on the birth date and handles database interaction for inserting or updating student records and test scores. It includes a “save” button that triggers validation checks and database updates, ensuring that all required fields are filled and at least one test is completed before saving. The class integrates with a database manager for data persistence and updates the database tab with the new entries upon successful save.

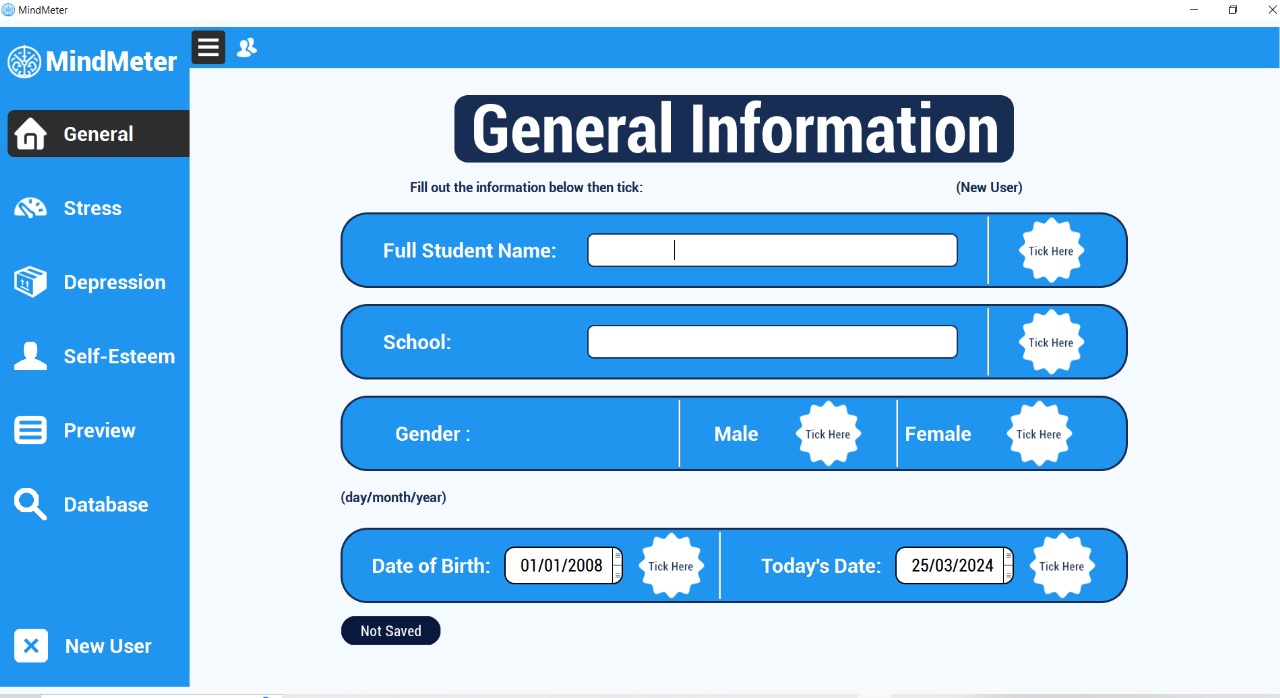
The DataBaseTab class is a part of the PyQt5 application that is responsible for displaying and managing data in a table widget. It provides functionalities such as refreshing the table, editing records, undoing changes, and saving updates. The class also handles the opening of a profile print dialogue when a row in the table is clicked. This dialogue allows users to view and print a detailed assessment profile of a student. The tab is integrated with a database manager to perform database operations such as fetching and updating records.

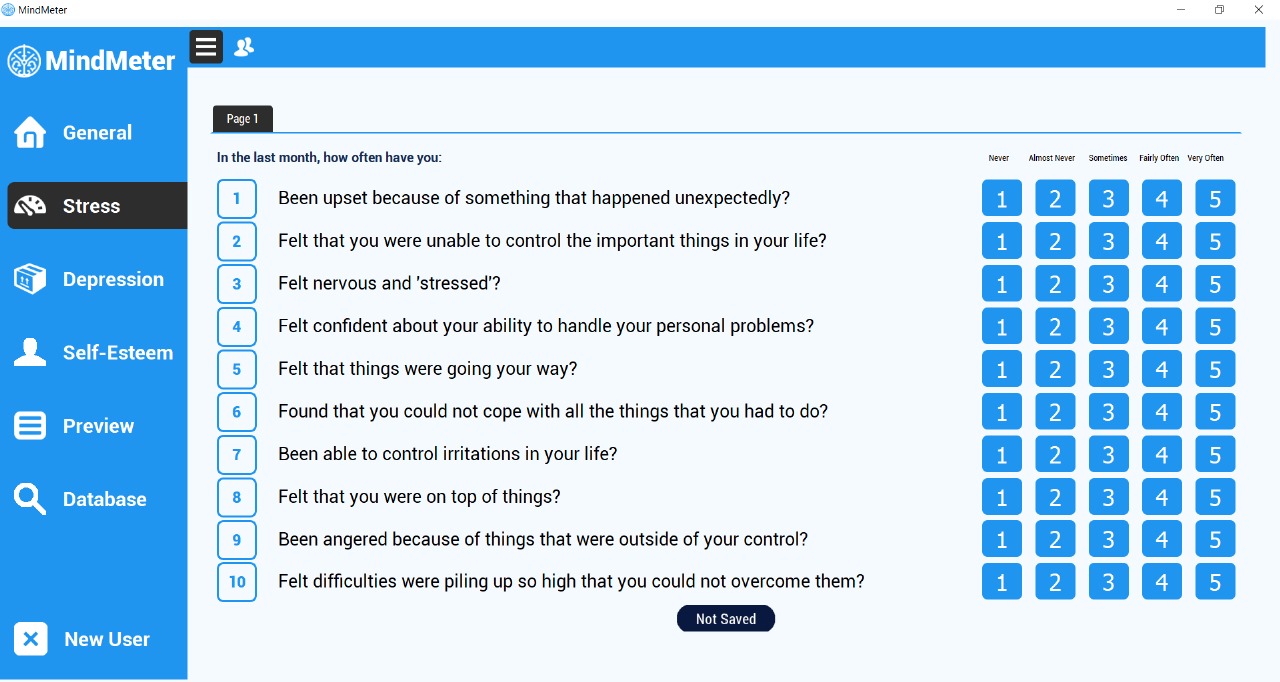
The UI\_Form class represents a profile print dialogue in the PyQt5 application. It provides the ability to view and compare test scores for stress, depression, and self-esteem. The class has methods for setting up the UI, populating test date combo boxes and handling test score viewing and comparison. It also includes functionality to create new tests and update the database. The dialogue is used to display a student’s assessment profile, including their general information and test scores.

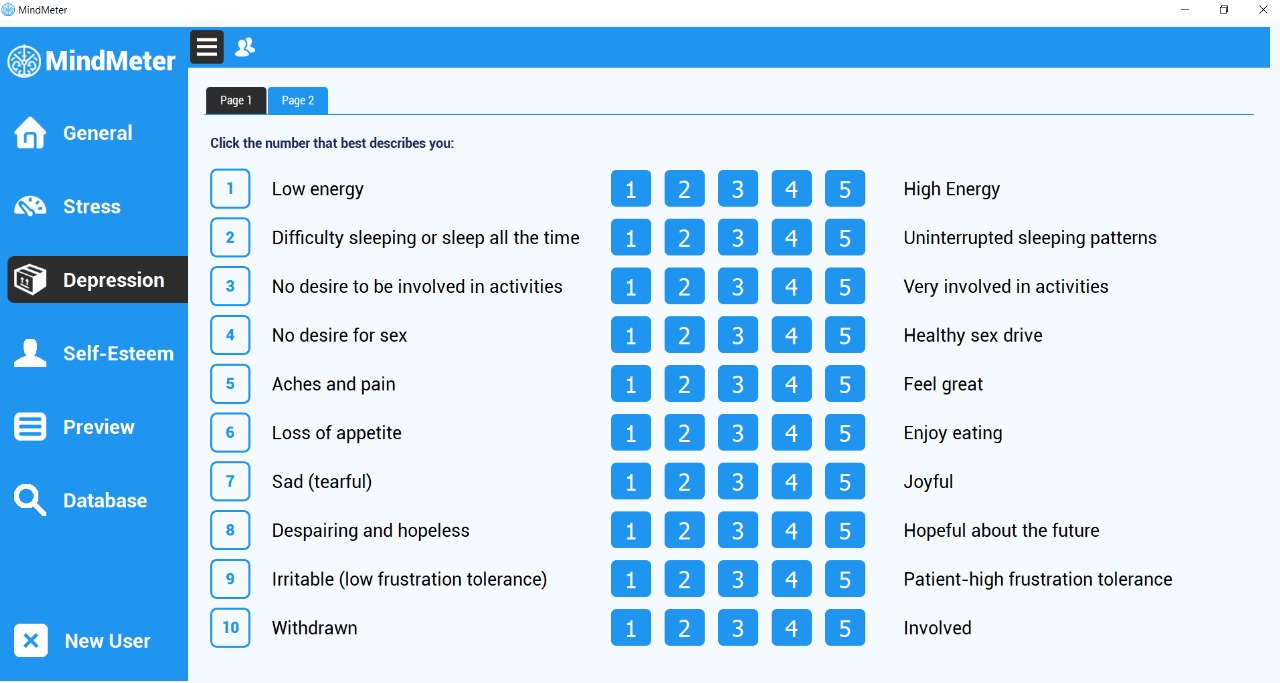
### 4.1.1 Screen images

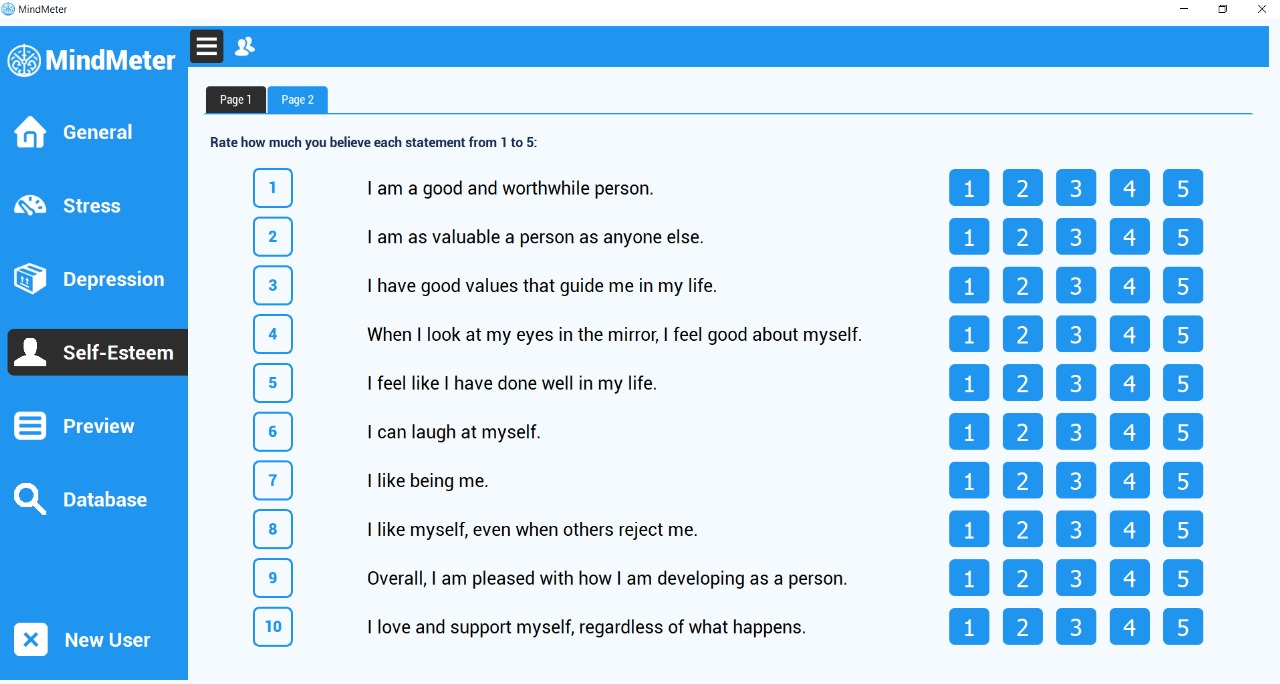
Attached are some screen images from the user’s point of view.

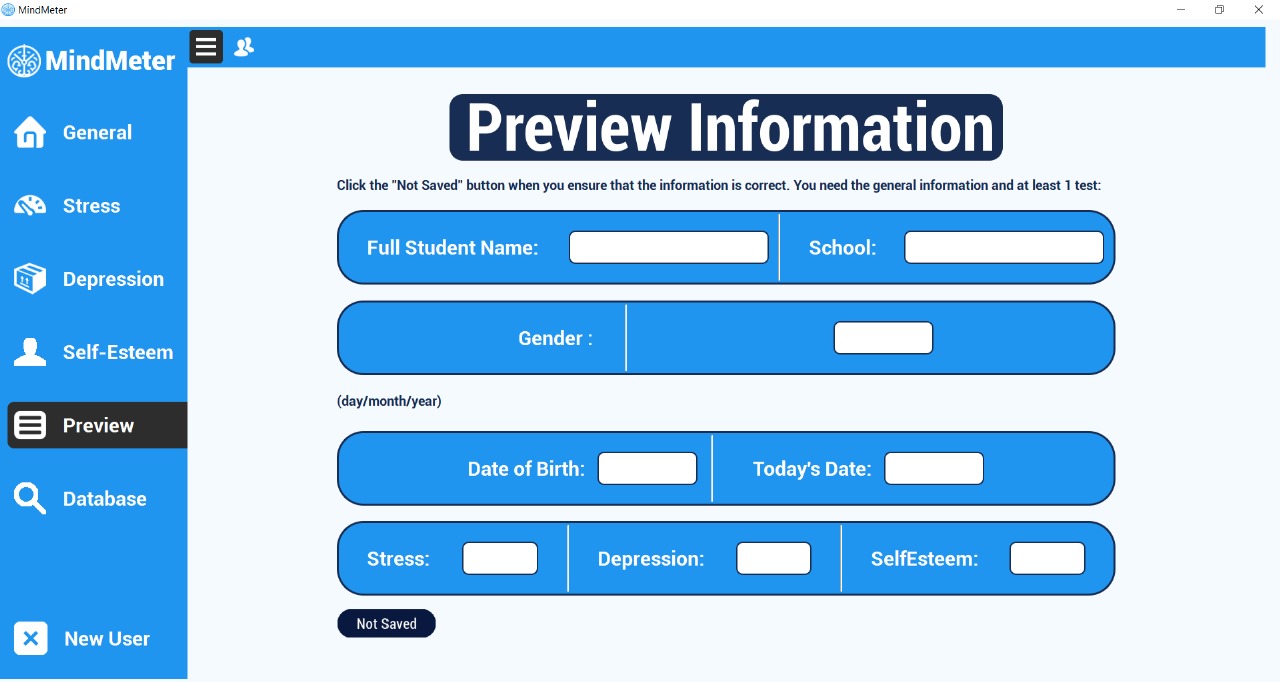
Figure 4: Start Screen

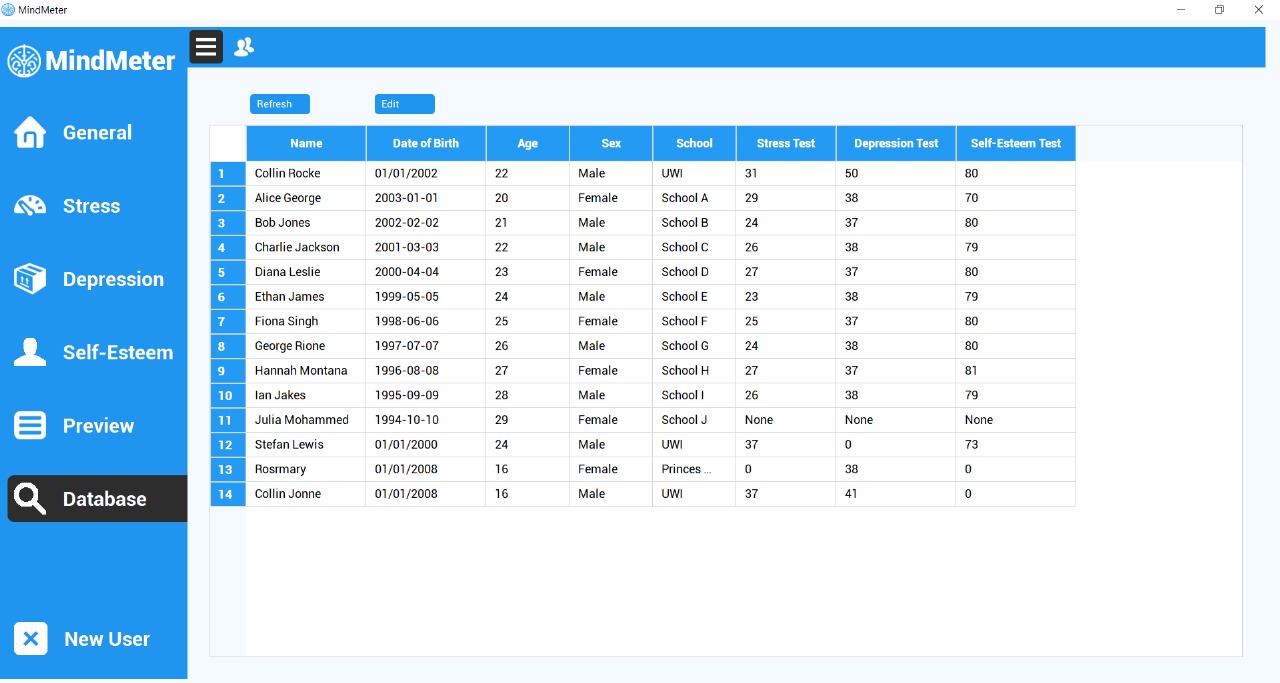
Figure 5: General Tab

Figure 6: Stress Tab

Figure 7: Depression Tab

Figure 8: Self-Esteem Tab

Figure 9: Preview Tab

Figure 10: Database Tab

## 4.2 Interface design rules

The user interface was designed with the following conventions:

1. **Consistent Layout**: To ensure the UI is easy to use, the placement of buttons and options were consistent throughout the application.
2. **Clear Navigation**: It is clearly indicated when switching between different tabs or modes within the application.
3. **Confirmation Dialogs**: Confirmation dialogs were implemented for critical actions such as erasing and saving answers or deleting data to prevent accidental actions.
4. **Print Functionality**: A print function button was provided for the counsellor to obtain a hardcopy version of the data. The function has a margin that ensures the content fits on the page properly.
5. **Save Button**: A save button was implemented at the end of the preview tab to allow users to save their progress.
6. **Student Information Retrieval**: The application allows the counsellor to easily access any student’s information in the database tab. This will allow for easy retrieval of data specific to any student.

# 5 Restrictions, limitations, and constraints

The software's design and functionality are influenced by a few restrictions, limitations and constraints which play a role in its efficacy.

**Device Constraints:**

* **Hardware Requirements**: The application can run on any modern-day PC, but it requires 30-40 RAM to operate smoothly.

# 6 Testing Issues

In this section, the testing strategy and preliminary test case specifications based on the requirements outlined in the System Requirements Specification (SyRS) document will be discussed.

## 6.1 Classes of tests

The testing methods utilized in this document to meet the customer’s requirements include:

* Unit Testing: This involves testing individual units or components of the software application in isolation to validate their performance as designed. Units refer to the smallest testable parts of the application such as functions, methods, or procedures.
* Integration Testing: This process tests the interface between two software units or modules to identify any problems or bugs arising from their combination. Different types of integration testing, such as Big-Bang, Bottom-Up, Top-Down, and Mixed Integration, are employed to ensure the correctness of the interface and expose faults in the interaction between integrated units.
* Black Box Testing: This method assesses the functionality of the system without delving into its internal workings. It ensures that the software meets specified requirements or specifications and can be performed using techniques such as Syntax-Driven Testing, Equivalence Partitioning, Boundary Value Analysis, Cause-Effect Graphing, Requirement-based Testing, and Compatibility Testing.
* White Box Testing: Also known as glass box or structural testing, this approach examines not only the functionality but also the internal structures such as data structures, design, and code of the software application. It provides testers access to the source code, enabling them to design test cases to examine the software's internal logic, flow, and structure.
* Performance Testing: This type of testing evaluates the software's performance under various conditions, ensuring it meets the expected performance criteria. Performance testing includes testing response time, scalability, and resource utilization.

## 6.2 Expected software response

The testing focuses on verifying the behavior of the Mind Meter application in response to various inputs and actions. This encompasses a range of scenarios to ensure that the application functions as intended and meets the specified requirements outlined in the System Requirements Specification (SyRS) document.

For instance, one test case (TCN-01) involves verifying that the application displays all tabs, including General, Stress test, Depression test, Self-Esteem test, Preview, and Database, upon installation and launch. This ensures that users have access to all necessary functionalities within the application interface. Similarly, TCN-05 verifies the presence of all student data within the database, ensuring comprehensive data storage and retrieval capabilities.

Overall, these test cases provide a comprehensive framework for evaluating the expected software response of the MindMeter application across various scenarios, ensuring that it meets the defined requirements and delivers a robust user experience.

## 6.3 Performance bounds

In this application the main performance requirement discussed was the option to fill in the relevant form and save the data to be further processed. The data retrieved from the student is then saved to a database allowing the administrator to have access to the digitized version of the mentioned forms. The storage capacity was considered one of the main performance requirements as it allows the guidance counsellor to retrieve the information stored. Another crucial performance requirement was the ability to create a score for the forms in addition to storage capacity. Since it is one of the primary system requirements for the guidance counsellor to utilize, the user interface was made to outline all these features. For the user interface the main performance requirement considered was navigation. To incorporate this requirement different tabs was added to the application along with instructions for students to choose.