

INFORMATION SECURITY AND VERIFICATION

SUMMER TRAINING REPORT

Submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

BY

Prateek Purohit - 40996302717



MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

(Affiliated to GURU GOBIND SINGH INDRAPASTA UNIVERSITY)

JANAKPURI, NEW DELHI - 110058

October 2019

DECLARATION

I hereby declare that the Training Report entitled (Information Security and Verification) is an authentic record of my own work as requirements of (6) weeks Training during the period from 4th June 2019 to 16th July 2019 for the award of degree of B.Tech (Computer Science & Engineering),GGSIPU, under the guidance of Mr. Subodh Sharma.

Date:

(Signature of Student)

Prateek Purohit

40996302717

Certificate



To

The Training & Placement Officer,
Maharaja Surajmal Institute of Technology,
Affiliated to Guru Gobind Singh Indraprastha University
C-4, Janak Puri, New Delhi-110058.

Ref: HIT/TRG/CERT/2019/21

Dated : 19th July, 2019

Sub : Certificate for attending Project Training Program

Name of the Student : Mr. Prateek Purohit

Roll No. : HIT/TRG/ACPT/2019/39

Period of Training : 4th June to 19th July, 2019

Assignments : Information Security and Verification

Performance : Very good

We wish him a very bright future.

Yours Sincerely

(K.N.Rama Rao)
DGM (IT)

के. एन. रामारव/ K N RAMARAO
उप महाप्रबंधक (प्रशिक्षण/अनुश्रवण)/DGM (IT)
एअर इंडिया लिमिटेड/AIR INDIA LIMITED
कंप्यूटर केंद्र/Computer Centre
इ.गं.ज. एयरपोर्ट, नई दिल्ली-110037
IGI Airport, New Delhi-110037

एअर इंडिया लिमिटेड Air India Limited
रजिस्टर्ड कार्यालय : एअरलाइन्स हाउस, 113, गुरुद्वारा रकाबगंज रोड, नई दिल्ली-110001. EPABX : 23422000
Regd. Office : Airlines House, 113, Gurudwara Rakabganj Road, New Delhi - 110001. EPABX : 23422000
वेबसाइट Website : www.airindia.in

WEEKLY EVALUATION

MAHARAJA SERAJMAL INSTITUTE OF TECHNOLOGY
Summer Industrial Training Evaluation Form J60 (MSIT EXM PA 02)
 (Year 2019 - 2020)

<p><u>Details of the Student</u></p> <p>Name: <u>Prateek Purohit</u> Roll No.: <u>40996302717</u> Branch and Semester: <u>CSE 5th Sem</u> Mobile No.: <u>9971930070</u> E-mail ID: <u>prateek007.purohit@gmail.com</u></p>	<p><u>Details of the Organisation</u></p> <p>Name and address of organisation: <u>Air INDIA</u> <u>101, Air port Terminal-1, Delhi-110037</u> Broader Area: <u>-</u> Name of Instructor: <u>Mr. Subodh Sharma</u> Designation and Contact No.: <u>AGM (IT)</u> <u>011-25671495</u></p>
---	---


Student Performance Record

	No. of days Scheduled for the training	Number of days actually attended	Curriculum Scheduled for the student	Curriculum actually covered by the student
Week 1	5	4	Deciding the project and language	Information security and verification using python
Week 2	5	5	Creating the database and search functionality	Database added search it working
Week 3	5	4	Adding more features to application	Adding, Updating and deleting employee
Week 4	5	3	Adding database security	Login page and admin user added
Week 5	5	5	Adding features like manage users	Add, update and delete database users
Week 6	5	4	Finalizing the project and checking bugs	Bugs checked and project finalized

Prateek
(Signature of the student)

Any comments or suggestions for the student performance during the training program (to be filled by instructor):

Subodh Sharma
 (Signature of the Instructor)
 19.07.2019



Note: To be sent by the MSIT and the student's signature date signed by his/her supervisor in charge by first week of September

ACKNOWLEDGEMENT

Every Summer Training big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, **Prateek Purohit**, the student of **Maharaja Surajmal Institute of Technology** (Computer Science & Engineering), am extremely grateful to **Air India** for the confidence bestowed. At this juncture I feel deeply honoured in expressing my sincere thanks to **Mr. Subodh Sharma** for providing valuable insights leading to the successful completion of my project and also **Dr. Naveen Dhaiya** for providing me this opportunity. I express my gratitude to Head of Department, CSE for arranging the summer training in good schedule. I would also like to thank all the Computer Science Department faculty members of Maharaja Institute of Technology College for their critical advice and guidance without which this seminar would not have been possible.

Prateek Purohit

(Roll No: 40996302717)

(CSE 2nd Shift)

Table of contents

SNo.	Topic	Page no
1	List of Figures	1
2	Abstract of the project	4
3	Chapter I – Company Profile	5
4	1.1) History	6
5	1.2) Corporate affairs and identity	7
6	1.3) Mascot	7
7	1.4) Logo and Livery	8
8	Chapter II - Tools and Technology used during the training	9
9	2.1) Hardware used	9
10	2.2) Computer Languages	9
11	2.3) Software Used	14
12	Chapter III – Demonstration of technology through project	16
13	3.1) Problem Statement	16
14	3.2) System requirement specification	16
15	3.3) Data Flow Diagram	19
16	3.4) Project Details	19
17	3.4.1) Tree structure	22
18	3.4.2) A Look at the GUI	23
19	3.4.3) CreateDatabase.txt	25
20	3.4.4) Setup.py	27
21	3.4.5) Login.py	30

22	3.4.6) Option.py	33
23	3.4.7) AddTrainee.py	36
24	3.4.8) SearchTrainee.py	40
25	3.4.9) UpdateTrainee.py	43
26	3.4.10) Delete Trainee.py	45
27	3.4.11) AddUser.py	46
28	3.4.12) UpdateUser.py	48
29	3.4.13) DeleteUser.py	50
30	Chapter IV: Screenshots of the project	51
31	References	57

List of Figures

No.	Topic	Page No.
Fig 1.1	Company Logo	5
Fig 1.2	Boeing 707-420	6
Fig 1.3	Headquarters of Air India	7
Fig 1.4	Air India Mascot	7
Fig 2.1	Python Logo	9
Fig 2.2	Create a class named MyClass with property x	11
Fig 2.3	A class named Person, use the __init__() function to assign values for name and age	11
Fig 2.4	Set the age of p1 to 40	11
Fig 2.5	Delete the age property from the p1 object	12
Fig 2.6	MySQL logo	12
Fig 2.7	A look at Visual Studio Code	14
Fig 2.8	Qt Designer	15
Fig 2.9	SQL	15
Fig 3.1	Entity relationship model of the project	17
Fig 3.2	Data flow diagram (Level 0)	19
Fig 3.3	Data flow diagram (Level 1)	20
Fig 3.4	Data flow diagram (Level 2)	21
Fig 3.5	Tree structure	22
Fig 3.6	A look at Qt designer	23
Fig 3.7	A look at setupDatabase.txt	25
Fig 3.8	Opening page of setup.py	27

Fig 3.9	Adding the admin user for the database	28
Fig 3.10	Successful creation of database	29
Fig 3.11	Login Window	30
Fig 3.12	database.py	31
Fig 3.13	Block diagram for showing relation between login window and MySQL	32
Fig 3.14	Option window	33
Fig 3.15	Adding Employee Window	36
Fig 3.16	Search Window	41
Fig 3.17	Update window	43
Fig 3.18	Delete employee Window	45
Fig 3.19	Create user window	46
Fig 3.20	Update user window	48
Fig 3.21	Delete user window	50
Fig 4.1	Setting up database(snapshots)	51
Fig 4.2	Setting up admin User (snapshots)	51
Fig 4.3	successfully created database (snapshots)	52
Fig 4.4	Login Page (snapshots)	52
Fig 4.5	Option Page (snapshots)	53
Fig 4.6	Add Employee window (snapshots)	53
Fig 4.7	Search Window (snapshots)	54
Fig 4.8	Update window (snapshots)	54
Fig 4.9	Delete Window (snapshots)	55
Fig 4.10	Add User window (snapshots)	55

Fig 4.11	Update User window (snapshots)	55
Fig 4.12	Delete User window (snapshots)	56
Fig 4.13	Login window with read access user (snapshots)	56
Fig 4.14	Option window for user with read rights (snapshots)	56

Abstract of the Project

India is the largest democratic country in the world with a population of 1.3 billion and 1.3 billion dreams. Many of these dreamers want to start their own business (small or big) and security plays a vital role in that. Every company is divided into departments and it is very crucial to have and maintain record of every employee at the entry point to prevent trespassing and any other event that harm the company in any means necessary. To prevent this there should be an application at every security room to cross check the employee before letting them enter in the premises. Every company should have a Security Management application. Security Management is the answer to all of these questions. A Security Managing application should be simple enough for any guard to operate and powerful enough to examine any employee in seconds. This will avoid trespassing, social engineering or any other kind of attacks in the company.

An Information Security and Verification is a Security and Employee Management Application which is basically a piece of software that allow the user (in this case the security guard) to access the employee's information using a unique ID. That information can be cross verified like Name, Photograph and Signature etc.

Using this application employees information can be managed (for e.g. Created, Updated and Deleted) using by an authenticated user through graphical user interface. The admin is allowed to manage users of the application for security and privacy purposes.

Chapter-I Company Profile



Fig 1.1 Company Logo

Name: AIR INDIA

Headquarters: Airlines House, Delhi, India

Established: 15 October 1932

Founder: J. R. D. Tata

Director/CEO: Pradeep Singh Kharola

Contact: 18602331407

Air India is the flag carrier airline of India, headquartered at New Delhi. It is owned by Air India Limited, a government-owned enterprise, and operates a fleet of Airbus and Boeing aircraft serving 94 domestic and international destinations. The airline has its hub at Indira Gandhi International Airport, New Delhi, alongside several focus cities across India. Air India is the largest international carrier out of India with an 18.6% market share.¹ Over 60 international destinations are served by Air India across four continents. The airline became the 27th member of Star Alliance on 11 July 2014.

The airline was founded by J. R. D. Tata as Tata Airlines in 1932; Tata himself flew its first Single-engine de Havilland Puss Moth, carrying air mail from Karachi to Bombay's Juhu aerodrome and later continuing to Madras (currently Chennai). After World War II, it became a public limited company and was renamed as *Air India*. On 21 February 1960, it took delivery of its first Boeing 707 named *Gauri Shankar* and became the first Asian airline to induct a jet aircraft in its fleet. In 2000–01, attempts

were made to privatise Air India and from 2006 onwards, it suffered losses after its merger with Indian Airlines.

1.1) History

1.1.1) Early years (1932–1945)

Air India had its origin as **Tata Air Services** later renamed to **Tata Airlines** founded by J. R. D. Tata of Tata Sons, an Indian aviator and business tycoon. In April 1932, Tata won a contract to carry mail for Imperial Airways and the aviation department of Tata Sons was formed with two single-engine de Havilland Puss Moths. 15 October 1932, The airline fleet consisted of a Puss Moth aircraft and a deHavilland Leopard Moth. Initial service included weekly airmail service between Karachi and Madras via Ahmedabad and Bombay. In its first year of operation, the airline flew 160,000 miles (260,000 km), carrying 155 passengers and 9.72 tonnes (10.71 tons) of mail and made a profit of ₹60,000 (US\$870).

1.1.3) Post-independence (1947–2000)



Fig 1.2 Boeing 707-420

As Air India

After World War II, regular commercial service was restored in India and Tata Airlines became a public limited company on 29 July 1946 under the name *Air India*. After Indian independence in 1947, 49% of the airline was acquired by the Government of India in 1948.

1.2) Corporate affairs and identity

1.2.1) Headquarters



Fig 1.3 Headquarters of Air India

Air India Limited is headquartered at the Indian Airlines House, New Delhi. Air India moved its headquarters from Air India Building, Mumbai to Delhi in 2013. The former Headquarters is a 23-storey tower on Marine Drive and was one of the targets of the 1993 Bombay bombings

1.3) Mascot



Fig 1.4 Air India Mascot

Air India's mascot is *the Maharajah (high king)*. It was created by Bobby Kooka, the Then-commercial director of Air India, and Umesh Rao, an artist with J. Walter Thompson Limited in 1946. Kooka stated that, "We call him a Maharajah for want of a better description. But his blood isn't blue. He may look like royalty, but he isn't royal". Air India adopted the Maharajah as its mascot in 1946. It was used in promoting it although initially designed only for the airline's memo-pads. The Maharajah was given a makeover in 2015 and the brand is represented by a younger version.

1.4) Logo and livery

Air India's colour scheme is red and white. The aircraft were painted in white with red palace style carvings on the outside of the windows and the airline's name written in red. The name is written in Hindi on the port side fuselage and in English on the port side tail. On the starboard side fuselage, the name is written in English, and in Hindi on the starboard tail. The window scheme was designed in line with the slogan *Your Palace in the Sky*. The aircraft were earlier named after Indian kings and landmarks.

CHAPTER II: Technology tools studied during training

2.1) Hardware Used:

For the development of the project, I have used the following hardware configuration:

Processor: i5-7200 CPU @ 2.50GHz

Ram: 8 GB

Hard Disk Drive: 1TB

Operating System: Windows 10

2.2) Computer Languages

2.2.1) Python (version - 3.7.3)



Fig 2.1 Python logo

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- Web development (server-side)
- Software development,
- Mathematics,
- System scripting

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Classes/Objects (OOPS)

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

```
class MyClass:  
    x = 5
```

Fig 2.2: Create a class named MyClass with property x

The `__init__()` Function

To understand the meaning of classes we have to understand the built-in `__init__()` function. All classes have a function called `__init__()`, which is always executed when the class is being initiated.

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)
```

Fig 2.3: A class named Person, use the `__init__()` function to assign values for name and age

Modify Object Properties

You can modify properties on objects like this:

```
p1.age = 40
```

Fig 2.4: Set the age of p1 to 40

Delete Object Properties

You can delete properties on objects by using the **Del** keyword:

```
del p1.age
```

Fig 2.5: Delete the age property from the p1 object

For More details about python visit the official website of python:
<https://www.python.org/>

2.2.2) MySQL



Fig 2.6 MySQL Logo

MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. This tutorial will give you a quick start to MySQL and make you comfortable with MySQL programming.

MySQL is a component of the LAMP web application software stack (and others), which is an acronym for *Linux, Apache, MySQL, Perl/PHP/Python*. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. MySQL is also used by many popular websites, including Facebook, Twitter, Flickr and YouTube.

Features

MySQL is offered under two different editions: the open source MySQL Community Server and the proprietary Enterprise Server. MySQL Enterprise Server is differentiated by a series of proprietary extensions which install as server plugins, but otherwise shares the version numbering system and is built from the same code base.

Major features as available in MySQL 5.6:

- A broad subset of ANSI SQL 99, as well as extensions
- Cross-platform support
- Stored procedures, using a procedural language that closely adheres to SQL/PSM
- Triggers
- Cursors
- Updatable views
- Online Data Definition Language (DDL) when using the InnoDB Storage Engine.
- Information schema
- Performance Schema that collects and aggregates statistics about server execution and query performance for monitoring purposes.
- A set of SQL Mode options to control runtime behaviour, including a strict mode to better adhere to SQL standards.
- X/Open XA distributed transaction processing (DTP) support; two phase commit as part of this, using the default InnoDB storage engine.
- Transactions with save points when using the default InnoDB Storage Engine. The NDB Cluster Storage Engine also supports transactions.

For more details about MySQL check the documentation at <https://dev.mysql.com/doc>

2.3) Software Used:

2.3.1) Visual Studio Code (IDE)

Visual Studio Code is a lightweight but powerful source code editor which runs on your Desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, Typescript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

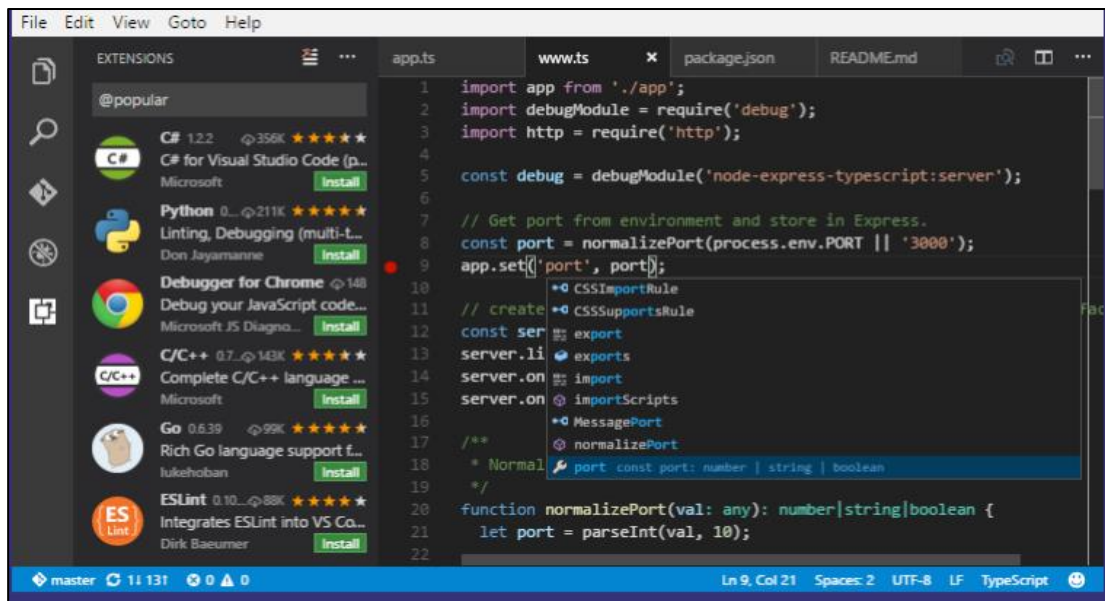


Fig 2.7: A look at Visual Studio Code

Meet IntelliSense

Go beyond syntax highlighting and autocomplete with IntelliSense, which provides smart completions based on variable types, function definitions, and imported modules.

Print statement debugging is a thing of the past.

Debug code right from the editor. Launch or attach to your running apps and debug with break points, call stacks, and an interactive console

2.3.2) QT Designer (For designing the GUI):

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. You can compose and customize your windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test those using different styles and resolutions. Widgets and forms created with *Qt Designer* integrate seamlessly with programmed code, Using Qt's signals and slots mechanism, so that you can easily assign behaviours to graphical elements. All properties set in *Qt Designer* can be changed dynamically within the code.

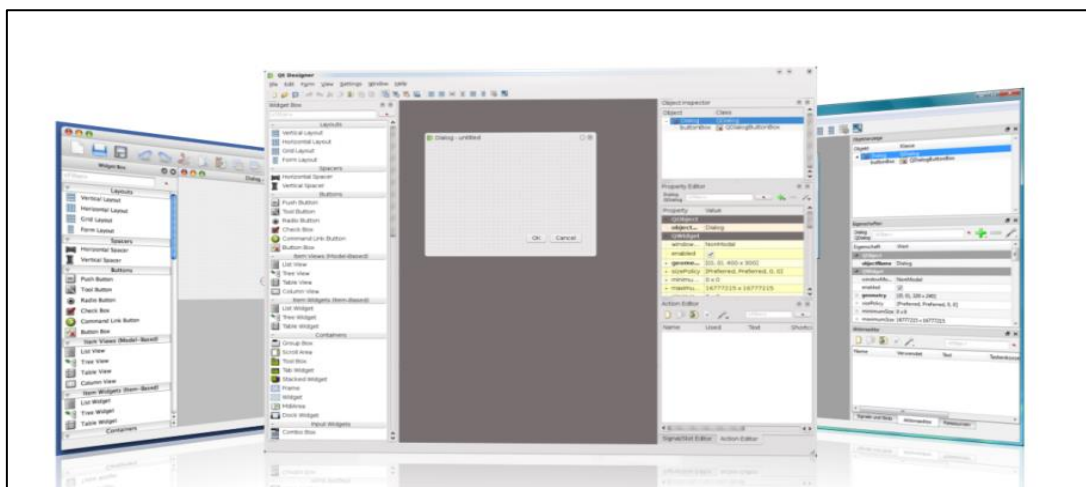


Fig 2.8: QT Designer

2.3.3) MySQL Command Line:

MySQL is a simple SQL shell with input line editing capabilities. It supports interactive and no interactive use. When used interactively, query results are presented in an ASCII-table format.

```
MySQL 8.0 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.15 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Fig 2.9: A look at MySQL Command Line

CHAPTER III– DEMONSTRATION OF TECHNOLOGY THROUGH PROJECT

3.1) Problem Statement

To create an employee management system for employee verification with added security features for the security room of the Air India Building.

3.2) System Requirement Specification (SRS)

3.2.1) Introduction

Purpose

The purpose of this document is to create a system to manage employees entering the Air India office.

Project Purpose

Big companies hire various Security Services in order to keep privacy and many other factors. But what about the small companies and business? They don't have such a large amount of capital to invest on security.

In many companies or offices registers are maintained to keep track of the employee as well as any person entering the building. This causes the following problems:

- **Consumption of paper:** A lot of paper is wasted for the process which is not good for environment. According to recent studies wood products like paper are responsible for the 10% deforestation of the total deforestation.
- **Backup Problems:** In case the information is lost. There is a very little possibility that it can be recovered. As well as it is difficult to backup and maintain the record as it will require more paperwork and storage in the office.

The purpose of the project is to tackle with these problems in a convenient and easy way.

3.2.2) Overall description

Product Perspective

A verification and security management system consist of the following:

- a) **Employee information:** It consist of all the information about the employee like Name, address, phone, signature etc. identified by a unique employee number.
- b) **Database users:** A way to manage the product users for security and information leaks.

Product Features

The major features of employee database system as shown in below entity–relationship model (ER model):

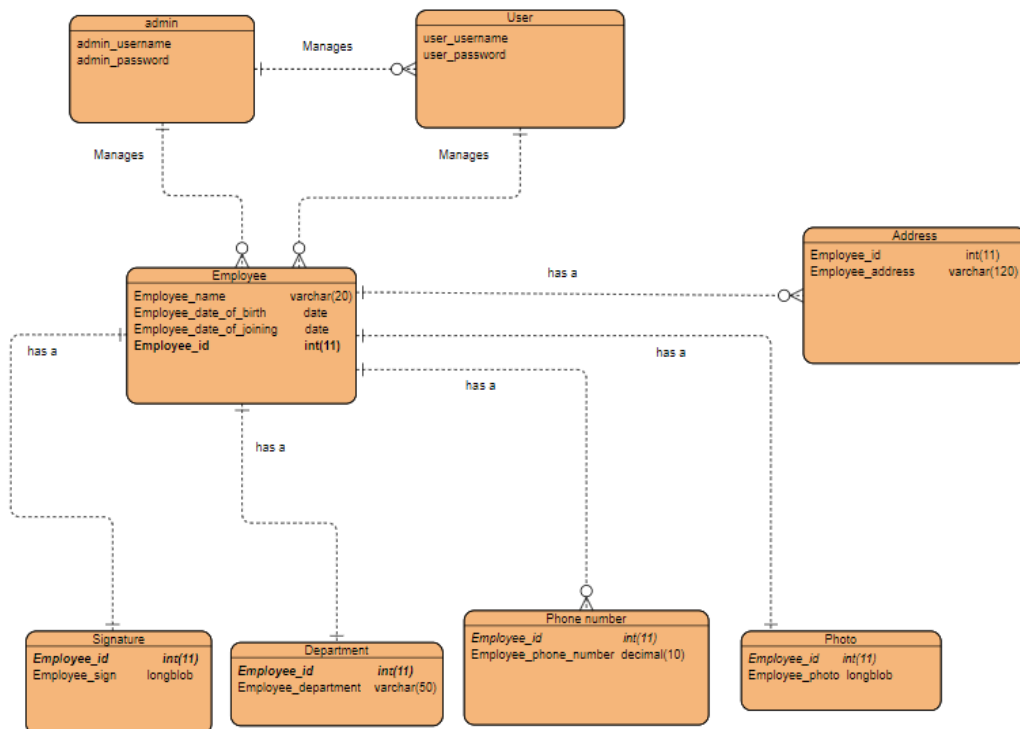


Fig 3.1 Entity relationship model of the project

The ER-MODEL consist of three major entities **Admin**, **User** and **Employee**. The **Admin** manages the **User** as well as the **Employee**. The **User** entity manages the **Employee**.

The **Employee** is further divided into five parts:

- Employee Address
- Employee Signature
- Employee Department
- Employee Phone Number
- Employee Photo

User class and characteristics

Users of the system should be able to retrieve information about the employee and update and delete information if present with the certain access. The User are sub categorized into two parts:

User with read Access:

- Can retrieve information about the employee

User with write Access:

- Retrieve information about the employee from the database
- Add a new employee
- Update the employee's information
- Delete an employee's record

Administrative privileges:

- Retrieve information about the employee
- Add an employee
- Update an employee's info
- Delete an employee's record
- Add a new user for the product or the application
- Update the user's access rights

OPERATING ENVIRONMENT

Operating environment for the airline management system is as listed below.

- client/server system
- Operating system: Windows.
- database: MySQL database
- platform: python

3.3) Data Flow Diagram (DFD)

3.3.1) DFD level 0

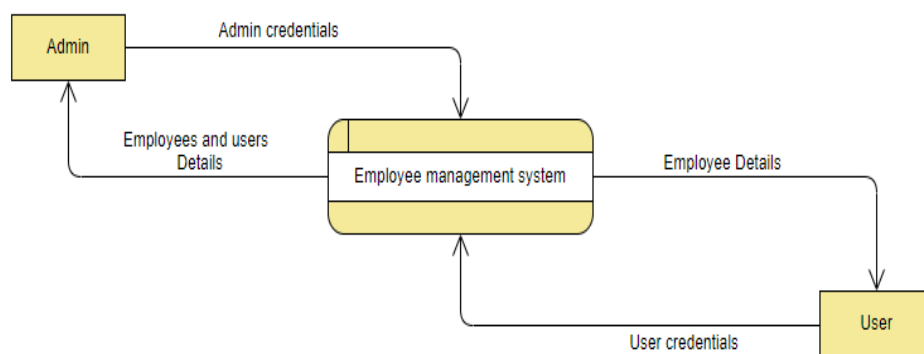


Fig 3.2 Data flow diagram (Level 0)

The **Admin** and **User** enters their credentials before accessing the employee management system. The **Admin** can retrieve both employee and users details whereas the **User** can only retrieve Employee details.

3.3.2) DFD level 1

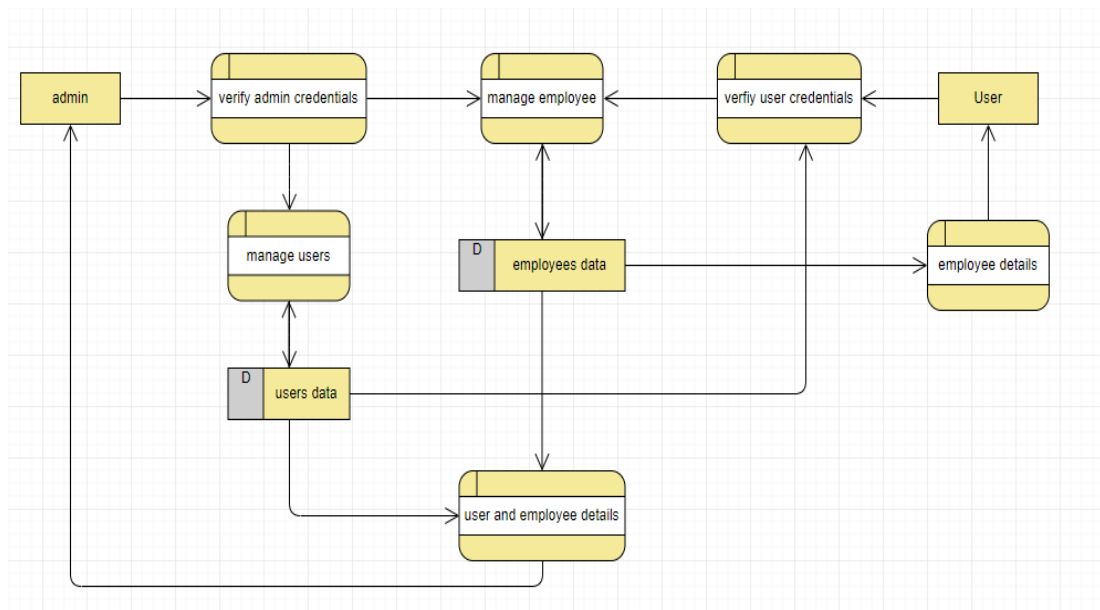


Fig 3.3 Data flow diagram (Level 1)

The **DFD** is further categorized into sub-processes. First of all **Admin** and **User** credentials are verified before they can access the system. The **Admin** can manage users as well as manage employee.

The manage users process connects the admin to the users data store and the manage employee process connects it with employee data store. Both the data stores go through the user and employee details process before the information is send to the admin.

So is for the **User**, in this case the user can only retrieve the employee data from the employee data store by the employee details process.

3.3.3) DFD Level 2

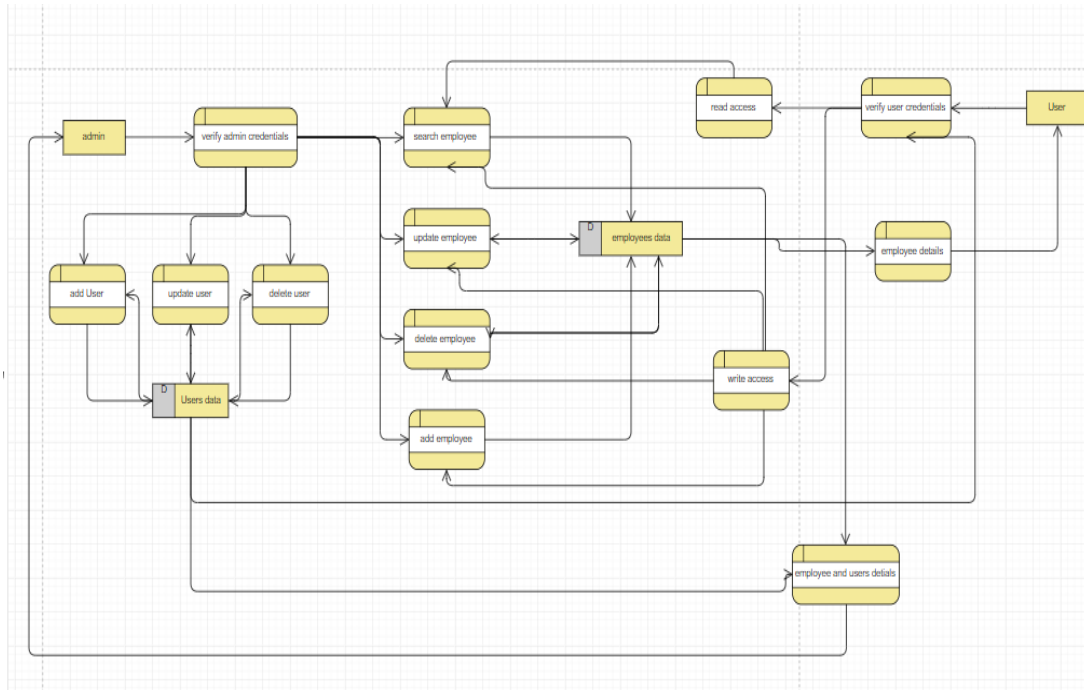


Fig 3.4 Data flow diagram (Level 2)

In DFD level 2 the manage users and manage employee process are divided into sub processes.

The manage users process is sub divided into:

- add user
- update user
- delete user

These sub processes can retrieve and send information from the Users data store.

The Manage employees process is sub divided into:

- Search employee
- Update employee
- Delete employee
- Add employee

These sub processes can retrieve and send information from the Employee data store.

Verify user credentials further passes to read access and write access processes which checks the access type of the user before he or she can access the database.

3.4) Project Details

3.4.1) Tree Structure

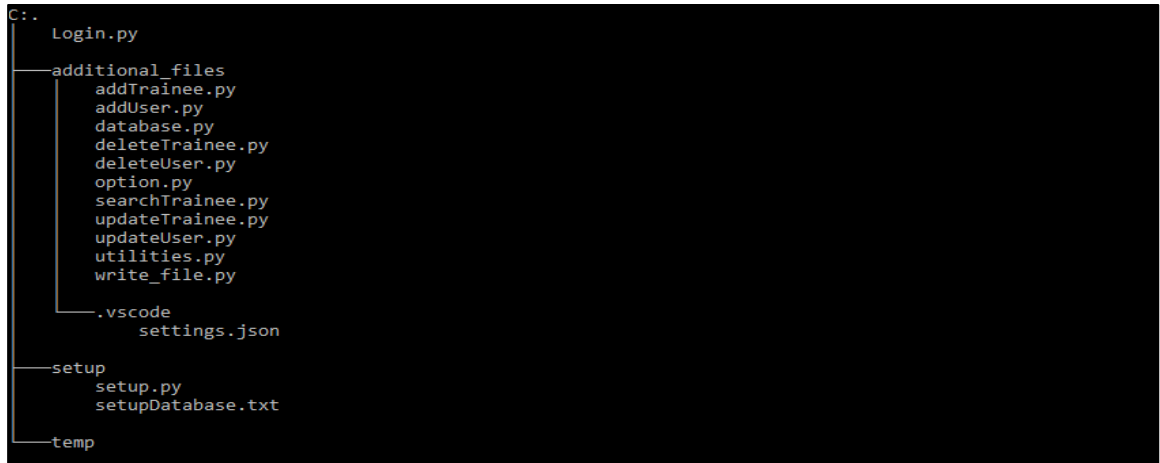


Fig 3.5: Tree structure

The tree structure basically consist of three main folders:

- Additional_files
- Setup
- Temp

Additional_files

Additional_files contain all the python files required for the application to run. These will be discussed later on in this chapter.

Setup

Setup consist of a setup.py file for setting up the database and admin user. The “setupDatabase.txt” consist of the SQL (Structured Query Language) code for creating the database.

Temp

Temp folder will be used to contain all the temporary data required for the application.

***Login.py is the entry point of the application**

3.4.2) Taking a look at GUI

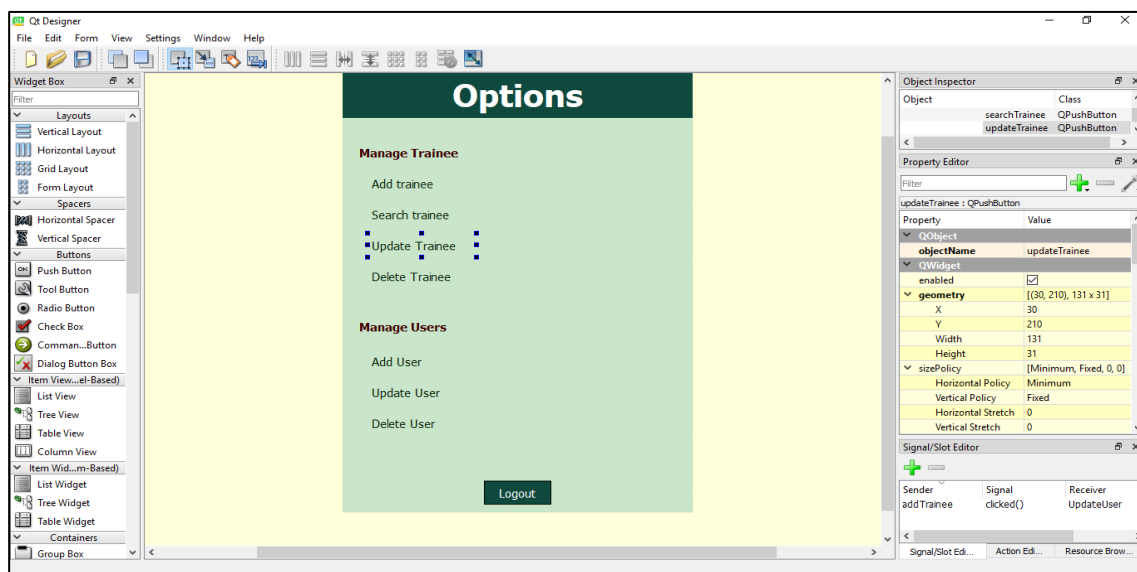


Fig 3.6: A look at Qt designer

Qt Designer stores file with the extension **.UI**. The **.UI** files are converted to python file using the command `pyuic5`.

```
C:\Users\prate\Desktop\juk\ui>pyuic5 -x option.ui -o option.py
```

-X: takes the UI file to be executed.

-O: saves the output with the filename provided

```

class Ui_optionPage(object):
    def __init__(self,db):
        self.db=db

    #GUI
    def setupUi(self, optionPage):
        optionPage.setObjectName("optionPage")
        optionPage.resize(ctypes.windll.user32.GetSystemMetrics(0), ctypes.windll.user32.GetSystemMetrics(1))
        self.centralwidget = QtWidgets.QWidget(optionPage)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        self.frame = QtWidgets.QFrame(self.centralwidget)
        self.frame.setStyleSheet("background:rgb(255, 255, 217);\n"
        """
        self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame.setObjectName("frame")
        self.frame_2 = QtWidgets.QFrame(self.frame)
        self.frame_2.setGeometry(QtCore.QRect(480, 40, 421, 571))

```

The python file created is in form of a class. Every UI file converted contains a **setupUI** function which consist of all the elements, labels, frames created.

```

def retranslateUi(self, optionPage):
    _translate = QtCore.QCoreApplication.translate
    optionPage.setWindowTitle(_translate("optionPage", "Manage"))
    self.addTrainee.setText(_translate("optionPage", "Add employee"))
    self.heading.setText(_translate("optionPage", "Options"))
    self.manageTrainee.setText(_translate("optionPage", "Manage employees"))
    self.manageUsers.setText(_translate("optionPage", "Manage Users"))
    self.searchTrainee.setText(_translate("optionPage", "Search employee"))
    self.updateTrainee.setText(_translate("optionPage", "Update employee"))
    self.deleteTrainee.setText(_translate("optionPage", "Delete employee"))
    self.addUser.setText(_translate("optionPage", "Add User"))
    self.UpdateUser.setText(_translate("optionPage", "Update User"))
    self.deleteUser.setText(_translate("optionPage", "Delete User"))
    self.logout.setText(_translate("optionPage", "Logout"))

```

The **retranslateUi** function sets name of all the labels and buttons set during creation of GUI. The code mentioned below creates the window with the help of the **QtWidgets** class of pyqt5 package of python.

```

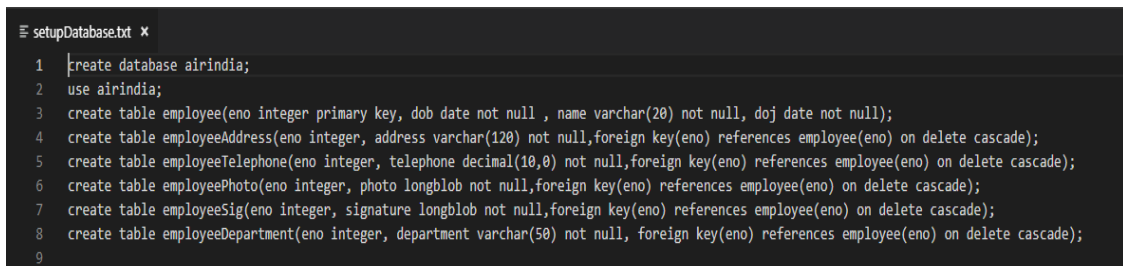
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    optionPage = QtWidgets.QMainWindow()
    ui=Ui_optionPage()
    ui.setupUi(optionPage)
    optionPage.show()
    sys.exit(app.exec_())

```

3.4.3) CreateDatabase.txt (The text file for the creation of database)

Before we start working with the application it is necessary to create a database which is going to be used to store and retrieve the employee's information.

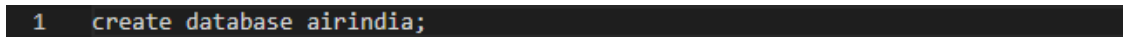
In this project the database is created using a text file named “**createDatabase.txt**”.



```
1 create database airindia;
2 use airindia;
3 create table employee(eno integer primary key, dob date not null, name varchar(20) not null, doj date not null);
4 create table employeeAddress(eno integer, address varchar(120) not null, foreign key(eno) references employee(eno) on delete cascade);
5 create table employeeTelephone(eno integer, telephone decimal(10,0) not null, foreign key(eno) references employee(eno) on delete cascade);
6 create table employeePhoto(eno integer, photo longblob not null, foreign key(eno) references employee(eno) on delete cascade);
7 create table employeeSig(eno integer, signature longblob not null, foreign key(eno) references employee(eno) on delete cascade);
8 create table employeeDepartment(eno integer, department varchar(50) not null, foreign key(eno) references employee(eno) on delete cascade);
9
```

Fig 3.7: A look at setupDatabase.txt

Initially a database named “**airindia**” is created using the command:



```
1 create database airindia;
```

The database contains six tables:

Employee:

The employee table is created using the “**create table**” command. It consist of:

- **Eno** – to store the employee unique id. It is a primary key since it will be unique for every employee.
- **Dob** – to store the date of birth of the employee.
- **Doj** – to store the date of joining of the employee.
- **Name** – to store the name of the employee.

EmployeeAddress

The employeeAddress table consist of:

- **Eno** – to store employee ID. It is a foreign key which refers to the eno in employee table.
- **Address** – to store employee's address

EmployeeTelephone

The employeeTelephone table consist of:

- **Eno** – to store employee ID. It is a foreign key which refers to the eno in employee table.
- **Telephone** – to store employee's phone numbers.

EmployeePhoto

The employeePhoto table consist of:

- **Eno** – to store employee ID. It is a foreign key which refers to the eno in employee table.
- **Photo** – to store employee's photograph. It is of datatype LONGBLOB.

EmployeeSig

The employeeSig table consist of:

- **Eno** – to store employee ID. It is a foreign key which refers to the eno in employee table.
- **Signature** – to store employee's signature. It is of datatype LONGBLOB.

EmployeeDepartment

The employeeSig table consist of:

- **Eno** – to store employee ID. It is a foreign key which refers to the eno in employee table.
- **Department** – to store employee's department.

3.4.4) Setup.py (For setting up the application's database)

3.4.4.1) Creating the database

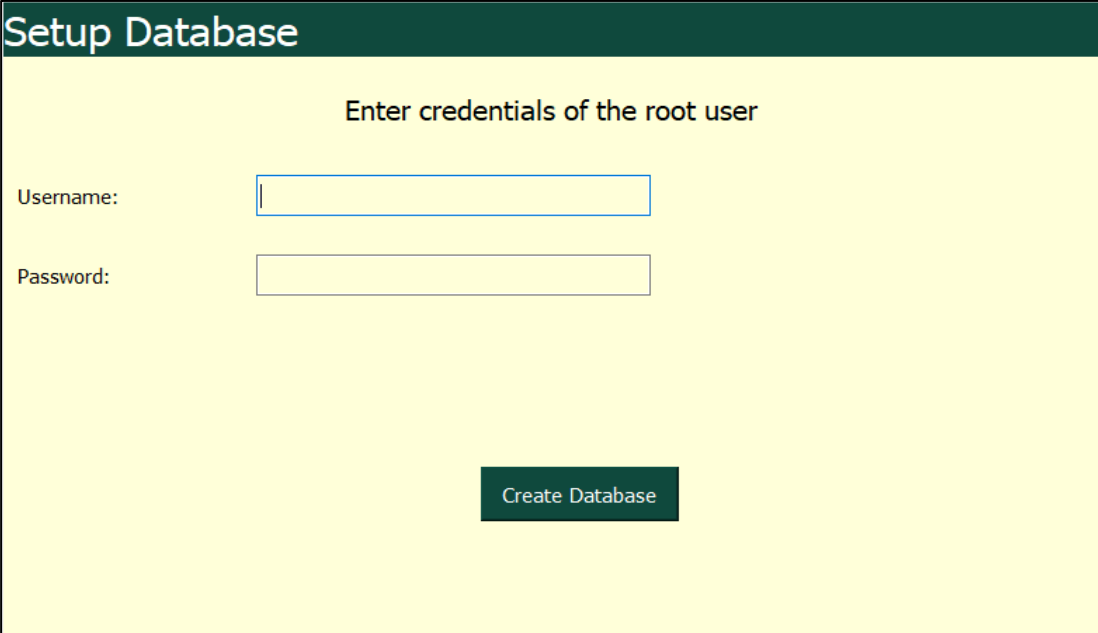


Fig 3.8: Opening page of setup.py

Firstly, Setup.py verifies the root user of the MySQL server before creating a database. The username and password are passed to **MySQLConnection** class of a python package called **mysql.connector**.

MySQL.connector is a python package used to establish a connection between the python file and the MySQL Server. It can be imported as shown below:

```
from mysql.connector import MySQLConnection , Error
```

The connection is established by the following set of code:

```
#creating the database
def onSubmit(self):
    if(self.usernameEdit.text()!='' and self.passEdit.text()!=''):
        try:
            self.db=MySQLConnection(host="localhost",user=self.usernameEdit.text(),password=self.passEdit.text())
            cur=self.db.cursor()

            with open('setupDatabase.txt','r') as f:
                for line in f:
                    cur.execute(line)

            cur.close()
```

Once we are successfully connected to MySQL server, a cursor is created which is used to perform all the MySQL actions.

The cursor helps to create the database by executing each line of the **setupDatabase.txt** file. The **Error** class of **mysql.connector** helps to catch any errors while connecting to the **MySQL Server**.

```
except Error as e:
    self.status.show()
    self.status.setStyleSheet('background:rgb(212,115,70)')

    if e.errno==1045:
        self.status.setText("wrong credentials")

    elif e.errno==1044:
        self.status.setText("Access denied to create a database")

    elif e.errno == 2006:
        self.status.setText('MySql server has gone')

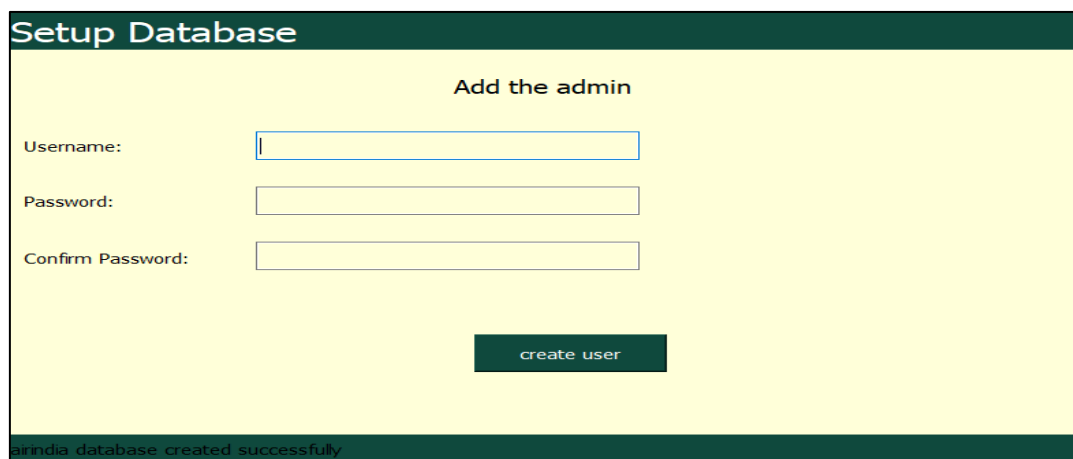
    elif e.errno == 2002:
        self.status.setText('Can\'t connect to local MySQL server through socket ')

    else:
        self.status.setText(str(e))
```

3.4.4.2) Creating the admin user of the application

Since we have created the database, now it's time to create an admin user that will have all privileges on the **"airindia"** database.

Once the database is successfully created the page is redirected to create an admin user. It consist of a username, password and password confirmation before submitting the details.



The screenshot shows a web application titled "Setup Database". Inside, there's a section "Add the admin" with three input fields labeled "Username:", "Password:", and "Confirm Password:". A green button labeled "create user" is positioned below the fields. At the bottom of the application window, a green status bar contains the text "airindia database created successfully".

Fig 3.9: Adding the admin user for the database

*The admin user is created by using the **CREATE USER** command:

```
cur.execute('create user "{}"@"localhost" identified by "{}"'.format(self.usernameEdit.text(),self.passEdit.text()))
```

*All the privileges are granted to the admin by the **GRANT** command:

```
cur.execute('grant all privileges on *.* to "{}"@"localhost" with grant option'.format(self.usernameEdit.text()))
```

Once the admin user is created the window is redirected to final page.

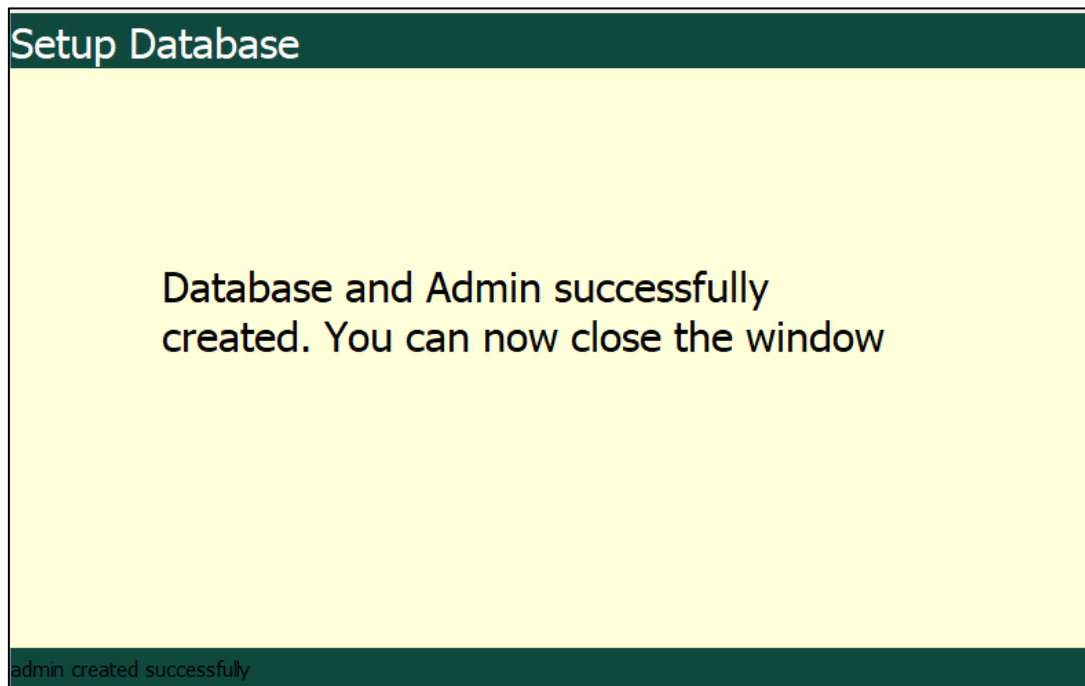


Fig 3.10: Successful creation of database

3.4.5 Login.py (entry point)

The application starts with the login page where the authenticated user enter his/her credentials in order to access the features of the application. For connecting to the MySQL server, the login page uses the class **Database** defined in **database.py** which acts as a medium to connect to connect to MySQL.

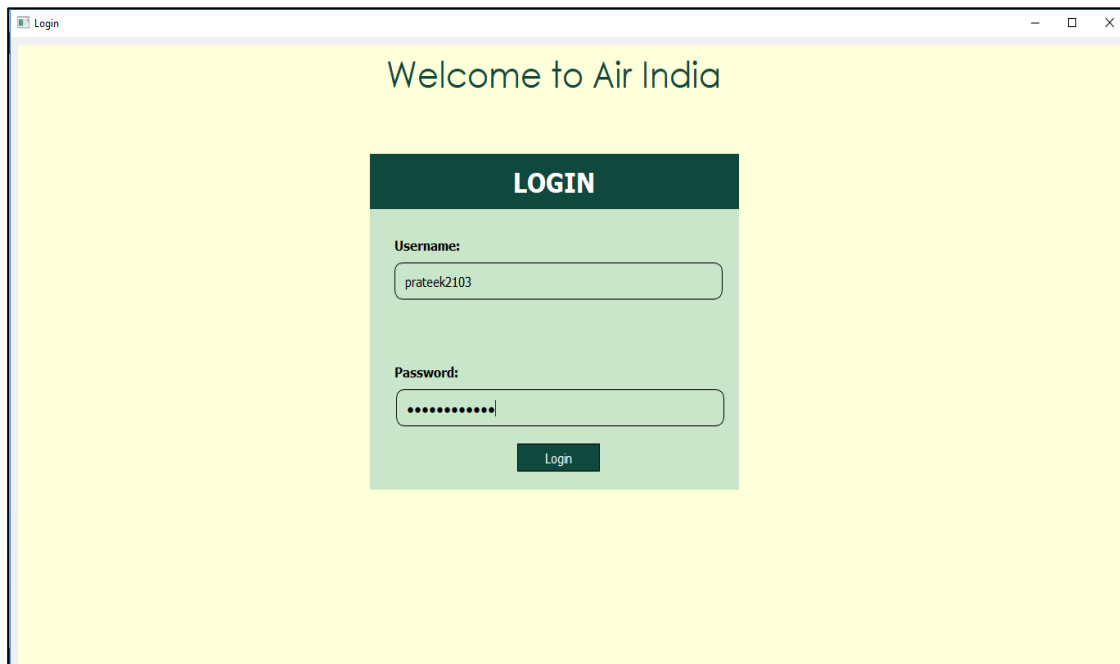


Fig 3.11: Login Window

Database.py imports the `mysql.connector` package to establish a connection to the MySQL server. Class `database` basically consist of two functions.

First, the `__init__()` function which acts as a constructor for the class and consist of two parameters **username** and **password**.

Secondly, the `callDatabase()` function which creates the connection with the MySQL server using the class **MySQLConnection** of `mysql.connector` package. **MySQLConnection** is passed with four parameters:

- **Host** – the host will be localhost as the MySQL server is installed in the machine.
- **User** – the user of the MySQL server
- **Password** – password of the user
- **Database** – the database to be connected to once the connection is established which “airindia” is in this case.

```

database.py x
1  from mysql.connector import MySQLConnection,Error
2
3
4  class Database():
5      def __init__(self,user,password):
6          self.user=user
7          self.password=password
8
9      def callDatabase(self):
10         try:
11             db=MySQLConnection(host="localhost",user=self.user,password=self.password,database="airindia")
12             return db
13
14         except Error as e:
15             return e
16
17

```

Fig 3.12: database.py

The whole process take place inside a try and except block which will catch any errors created during the connection. **MySQLConnection** returns an object which is an instance of the MySQL server. Now coming back to the login.py file.

The Login.py import the class Database as follows:

```

2  from additional_files import database

```

Once the Database class is imported, a new object named **db_obj** is created for that class.

```

#connecting to database
def onSubmit(self):
    if(self.userEdit.text()!='' and self.userEdit.text()!=''):
        db_obj=database.Database(self.userEdit.text(),self.passEdit.text())
        db_mysql=db_obj.callDatabase()

        if(db_mysql and isinstance(db_mysql,MySQLConnection)):
            self.optionWindow(db_mysql)

        elif(db_mysql and isinstance(db_mysql,Error)):
            if db_mysql.errno==1045:
                self.status.setText('wrong credentials')

            elif db_mysql.errno == 1049:
                self.status.setText('database does not exist')

            elif db_mysql.errno == 2006:
                self.status.setText('MySql server has gone')

            elif db_mysql.errno == 2002:
                self.status.setText('Can\'t connect to local MySQL server through socket ')

            else:
                self.status.setText(str(db_mysql))

```

The value of the class method `callDatabase ()` is stored in the `db_mysql`. To check whether `db_mysql` is an error object or the database object `isinstance ()` method of python is used. Errors are handled using the **Error** class of **mysql.connector**. If the object is the instance of the `MySQLConnection` then **optionPage** is opened with `db_mysql` as its parameter.

```
#connection to option window
def optionWindow(self,db):
    self.window=QtWidgets.QMainWindow()
    self.ui=option.Ui_optionPage(db)
    self.ui.setupUi(self.window)
    self.window.show()
    self.MainWindow.close()
```

Once the authentication is successful then the Login window is redirected to Option window using the following piece of code.

Block Diagram

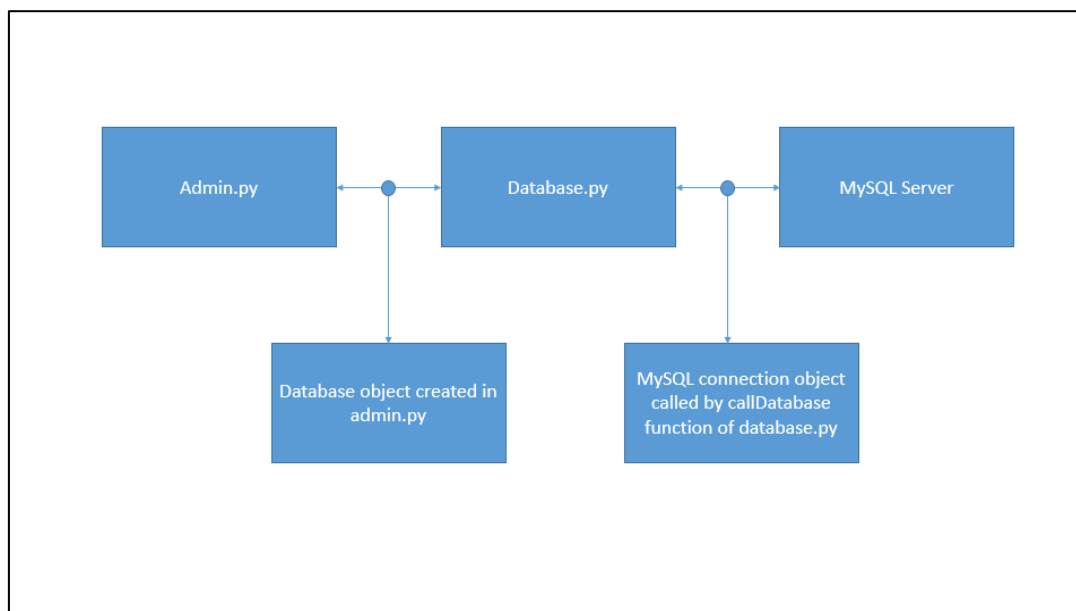


Fig 3.13: Block diagram for showing relation between login window and MySQL server

3.4.6) Option.py

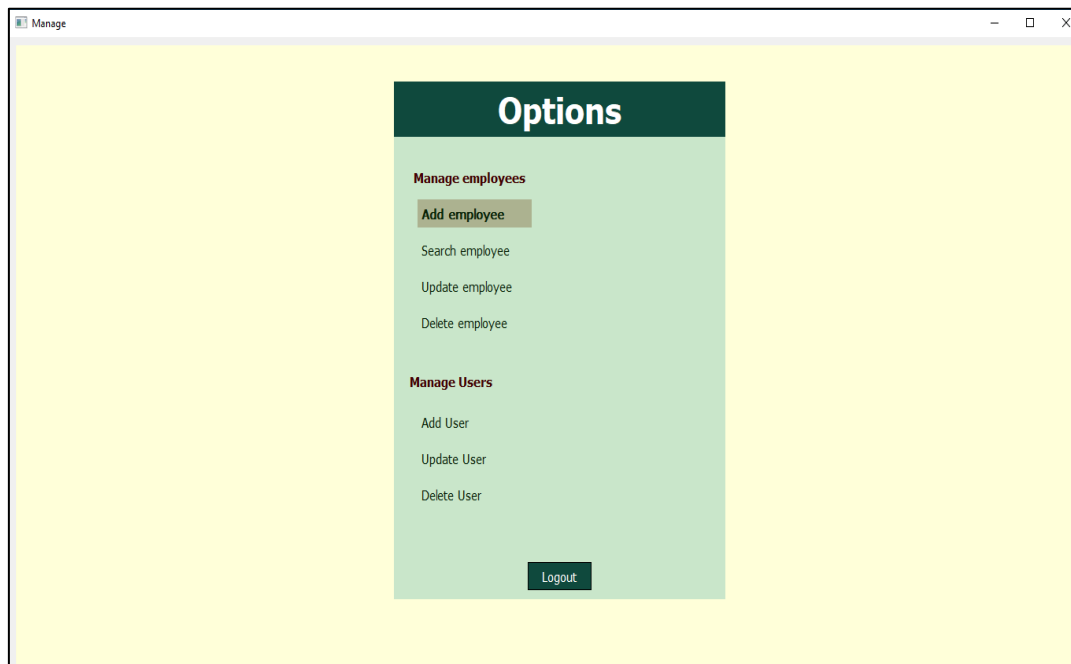


Fig 3.14: Option window

The option window is used to select various options to maintain employee and database users. It is basically divided into two parts:

- **Manage Employees** – These options are used to create, update, search and delete employee.
- **Manage Users** – These options are used to manage the users of the application.

Firstly, all the windows are imported to the option window in order to create a link with all the options present in the window.

```
#importing all the windows for option view
from additional_files import addTrainee as Ui_addTrainee #adding trainee window
from additional_files import searchTrainee as Ui_searchTrainee #search trainee window
from additional_files import updateTrainee as Ui_updateTrainee #update trainee window
from additional_files import deleteTrainee as Ui_deleteTrainee #delete trainee window
from additional_files import addUser as Ui_addUser #adding a new user to access database
from additional_files import updateUser as Ui_updateUser #updating the user rights
from additional_files import deleteUser as Ui_deleteUser #deleting the user
```


There are basically seven windows:

- **Ui_addTrainee** – For Add Employee
- **Ui_searchTrainee** – For Search Employee
- **Ui_updateTrainee** – For Update Employee
- **Ui_deleteTrainee** – For Delete Employee
- **Ui_addUser** – For Add User
- **Ui_updateUser** – For Update User
- **Ui_deleteUser** – For Delete User

```
#button actions(clicked)
self.addTrainee.clicked.connect(self.traineeWindow)
self.searchTrainee.clicked.connect(self.searchTraineeWindow)
self.updateTrainee.clicked.connect(self.updateTraineeWindow)
self.deleteTrainee.clicked.connect(self.deleteTraineeWindow)
self.addUser.clicked.connect(self.addUserWindow)
self.UpdateUser.clicked.connect(self.updateUserWindow)
self.deleteUser.clicked.connect(self.deleteUserWindow)
self.logout.clicked.connect(self.logoutWindow)
```

All windows are connected by the connect function of python. Each window is changed by the similar set of code. Every window is passed with the database object created during login. Most important is to check the user rights before letting any access to any option in the option page. This is done using the **checkUser()** function.

```
cur.execute('show grants')
grants=cur.fetchall()[0][0]

if not "GRANT OPTION" in grants:
    self.addUser.hide()
    self.UpdateUser.hide()
    self.deleteUser.hide()
    self.manageUsers.hide()

writeGrants=["INSERT", "UPDATE", "DELETE"]

for grant in writeGrants:
    if not grant in grants:
        self.addTrainee.hide()
        self.updateTrainee.hide()
        self.deleteTrainee.hide()
        self.searchTrainee.setGeometry(QtCore.QRect(30, 130, 131, 31))
        break
    else:
        continue
```

The **checkUser () function** uses the database object passed from the login window. The **cursor** is used to execute the **MySQL** queries.

User access are displayed using the **SHOW GRANTS** commands. If GRANT OPTION is not available in user's rights then manage user option is hidden or disabled. If write grants that are insert, update and delete are not available for the database user then Add employee, Update Employee and Delete Employee are also hidden.

The **Logout** button close the MySQL connection and delete all the temporary data. The window is redirected to Admin window.

```
def logoutWindow(self):
    os.chdir('temp')
    for file in os.listdir():
        os.remove(file)
    from Login import Ui_MainWindow as adminWindow
    self.db.close()
    self.window=QtWidgets.QMainWindow()
    self.ui=adminWindow()
    self.ui.setupUi(self.window)
    self.window.show()
    self.optionPage.close()
```

Python OS module is used to work with all operating system's functions.

3.4.7) AddTrainee.py (Adding the employee in the database)

The screenshot shows a web form for adding a trainee. It includes fields for Name, Date of Birth, Date of Joining, Address 1, Address 2, Phone No., and Select Department. There are also image upload sections for a photo and a signature. The form has a yellow background and a green Submit button.

Fig 3.15: Adding Employee Window

The adding employee window contains **Name**, **Address1**, **Address 2 (optional)**, **phone number**, **Date of Birth**, **Date of Joining**, **Photo**, **Signature** and **Department**.

3.4.7.1) STORING PHOTO AND SIGNATURE

Every other field is stored in text format but photo and signature are image files, they cannot be stored like that. For images we use the LONGBLOB datatype of MySQL. LONGBLOB is used to store large amount of data in binary format. Therefore before storing images we have to convert it into binary. This is done with the help of the following function:

```
1 def read_file(filename):
2     with open(filename, 'rb') as f:
3         photo = f.read()
4     return photo
5
```

The **read_file** function takes the path of the image and read it in binary format using the **open** function of python. That binary data is then stored into a variable named **photo** and is returned.

3.4.7.2) SQL QUERIES

All the data from the adding employee window is stored in the database using SQL queries.

```
#queries for inserting data
employeeQuery="insert into employee values(%s,%s,%s,%s)"
employeeAddressQuery="insert into employeeAddress values(%s,%s)"
employeeTelephoneQuery="insert into employeeTelephone values(%s,%s)"
employeePhotoQuery = "Insert into employeePhoto values(%s,%s)"
employeeSigquery = "Insert into employeeSig values(%s,%s)"
employeeDepartmentQuery="insert into employeeDepartment values(%s,%s)"
```

Before inserting the data a unique id is generated for each user by the **generateUniqueId** function as follows:

```
def generateUniqueId(self,size):
    #generates a unique id
    id=uuid.uuid1()

    #converting id into int and resizing it to given size
    id=id.int % (10**(size))

    return id
```

Data is inserted into the respective tables using these queries.

```
#adding name, date of birth , date of joining
try:
    cur.execute(employeeQuery,(id,self.info['dob'],self.info['name'],self.info['doj']))
    self.db.commit()

    #adding addresses
    for addr in self.info['address']:
        if(addr!=''):
            cur.execute(employeeAddressQuery,(id,addr))
            self.db.commit()

    #adding phone numbers
    for num in self.info['phone']:
        if(num!=''):
            cur.execute(employeeTelephoneQuery,(id,int(num)))
            self.db.commit()

    #adding department
    cur.execute(employeeDepartmentQuery,(id,self.info['department']))
    self.db.commit()

    #adding photo
    cur.execute(employeePhotoQuery,(id,self.info['photo']))
    self.db.commit()

    #adding signature
    cur.execute(employeeSigquery,(id,self.info['sig']))
    self.db.commit()
```

After every execution statement the database object is updated by using **self.db.commit** () to reflect the changes in the database. Errors are handled using the **Error** class of **mysql.connector**.

```
except Error as e:
    self.status.setStyleSheet('background:rgb(212,115,70)')

    if e.errno==1062:
        self.status.setText('Duplicate Id error : Try Submitting again')

    if e.errno==1146:
        self.status.setText('Problem with the Database (table does not exist)')

    elif e.errno==1054:
        self.status.setText('Problem with the Database (column does not exist)')
```

3.4.7.3) VALIDATION OF FIELDS

Before storing the data in the database it is necessary to validate each field to maintain consistency and integrity of data. It is done with the help **QRegExp** class of **PYQT5** tools.

PYQT5 tools is a python package used for designing GUI of the application. **QT Designer** is a part of PYQT5 tools.

```
2 from PyQt5.QtGui import QRegExpValidator
3 from PyQt5.QtCore import QRegExp
```

Validation of Name

```
#validation for name
def validation_name(self):
    regName=QRegExp("\w{3,}\s?\w*")
    input_validator = QRegExpValidator(regName, self.NameEdit)
    result=input_validator.validate(self.NameEdit.text(),0)
    self.NameEdit.setValidator(input_validator)
    self.validation_color(result,self.NameEdit)
    if(result[0]==2):
        self.flags['name']=True
    else:
        self.flags['name']=False
```

Name is validated by the regular expression created by **QRegExp**. It should be at least 3 characters long.

Validation of Address

```
#validation for addresses
def validation_address1(self):
    #address1edit
    regAddress=QRegExp("[\\w\\s-.,:]{15,120}")
    input_validator = QRegExpValidator(regAddress, self.address1Edit)
    result=input_validator.validate(self.address1Edit.text(),0)
    self.address1Edit.setValidator(input_validator)
```

```
def validation_address2(self):
    #address2Edit
    regAddress=QRegExp("([\\w\\s-.,:]{15,120}|\\w{0})")
    input_validator=QRegExpValidator(regAddress,self.address2Edit)
    result=input_validator.validate(self.address2Edit.text(),0)
    self.address2Edit.setValidator(input_validator)
```

Address1 should be at least 15 characters long and has a maximum length of 120 characters including spaces, dash (-), dot (.), colon (:).

Address2 is optional but if present it should follow the same validation as Address1.

Validation of Phone number

```
#validation for phones
def validation_phone1(self):
    #Phone
    regPhone=QRegExp("\\d{10}")
    input_validator=QRegExpValidator(regPhone,self.phone1Edit)
    result=input_validator.validate(self.phone1Edit.text(),0)
    self.phone1Edit.setValidator(input_validator)
    self.validation_color(result,self.phone1Edit)
```

First phone number is necessary and should of exact length of 10 digits.

```
def validation_phone2(self):
    regPhone=QRegExp('(\\d{10}|\\d{0})')
    input_validator=QRegExpValidator(regPhone,self.phone2Edit)
    result=input_validator.validate(self.phone2Edit.text(),0)
    self.phone2Edit.setValidator(input_validator)
    self.validation_color(result,self.phone2Edit)
```

Second Phone number is optional if present then should have a length of 10 digits.

```
def validation_phone3(self):
    regPhone=QRegExp('(\\d{10}|\\d{0})')
    input_validator=QRegExpValidator(regPhone,self.phone3Edit)
    result=input_validator.validate(self.phone3Edit.text(),0)
    self.phone3Edit.setValidator(input_validator)
    self.validation_color(result,self.phone3Edit)
```

Third phone number follows same rule as second phone number.

Validation of Department

Since validation is a select box it should contain any option except “Select Department”.

```
#validation for department
def validation_department(self):
    regDepartment=QRegExp("^(?!Select Department).)*$")
    input_validator=QRegExpValidator(regDepartment,self.DepartmentEdit)
    result=input_validator.validate(self.DepartmentEdit.currentText(),0)
    self.DepartmentEdit.setValidator(input_validator)
    self.validation_color(result,self.DepartmentEdit)
```

***Each validation is connected to a flag whose results is true if validation is successful. If all flags are true then only the information can be submitted.**

```
def onSubmit(self):
    #if all the flags are true then the form can be submitted
    if(self.photo!=''):
        self.flags['image']=True
    else:
        self.flags['image']=False

    if(self.sig!=''):
        self.flags['sig']=True
    else:
        self.flags['sig']=False

    count=0
    for flag in self.flags:
        if(self.flags[flag]==True):
            count=count+1

    if count==9:
        self.shouldSubmit=True
        self.onChecking()

    else:
        self.shouldSubmit=False
```

3.4.8) SearchTrainee.py (Window for searching employee)

The screenshot shows a window titled "SearchTrainee.py" with a search bar at the top. Below the search bar, a "Details" section displays the following information:

ID :	134307384
Name:	ram kumar
Date of Birth:	03 Apr 1999
Date of Joining:	21 May 2019
Address 1:	a-145, new ashok nagar , delhi - 96
Address 2:	Not Available
Phone Numbers :	9229299292 Not Available Not Available
Department:	Information Technology

On the right side of the details section, there is a photo of a man with glasses and a blue shirt, and a signature below it. At the bottom left, a status bar says "Found a Employee".

Fig 3.16: Search Window

SQL QUERIES

```
#getting the trainee id before executing other queries
cur.execute('select eno from employee where eno={}'.format(self.idEdit.text()))
id=cur.fetchone()

if(id):
    #queries
    id=id[0] #extracting id from tuple
    employeeQuery="select name , dob, doj from employee where eno = %s "
    employeeAddress="select address from employeeAddress where eno= %s"
    employeeTelephone="select telephone from employeeTelephone where eno= %s"
    employeeDepartment="select department from employeeDepartment where eno= %s"
    employeePhoto ='select photo from employeePhoto where eno = %s'
    employeeSig='select signature from employeeSig where eno = %s'
```

Each Query is then executed by the cursor and stored in respective variables which help in displaying data. Since photo and signature were stored in binary format, so to display them **write_file** function is used which converts the binary data into image.


```

#execution
try:
    cur.execute(employeeQuery,(id,)) #employeeTable
    row1=cur.fetchone()

    cur.execute(employeeAddress,(id,)) #address table
    row2=cur.fetchall()

    cur.execute(employeeTelephone,(id,)) #telephone table
    row3=cur.fetchall()

    cur.execute(employeeDepartment,(id,)) #department table
    row4=cur.fetchone()

    cur.execute(employeePhoto,(id,)) #photo table
    photo=cur.fetchone()[0]

    cur.execute(employeeSig,(id,)) #signature table
    sig=cur.fetchone()[0]

```

After the photo and signature are fetched they are passed to write_file function which converts it into image.

```

6 def write_file(data,filename):
7     with open(filename, 'wb') as f:
8         f.write(data)

```

All the images are stored in a folder temporarily in the **TEMP** folder.

3.4.9) UpdateTrainee.py (Update employee details)

Id: Search

Details

ID : Date of Birth:

Name: Date of Joining:

Address 1:

Address 2:

Phone Numbers:

Department:

Found a employee

Fig 3.17: Update window

SQL QUERIES

```
#queries for updating data
employeeQuery="update employee set dob = %s, name= %s , doj = %s where eno = %s"
employeeAddressQuery="insert into employeeAddress values (%s,%s);"
employeeTelephoneQuery="insert into employeeTelephone values (%s,%s);"
employeePhotoQuery = "update employeePhoto set photo = %s where eno = %s;"
employeeSigquery = "update employeeSig set signature= %s where eno= %s;"
employeeDepartmentQuery="update employeeDepartment set department =%s where eno=%s;"
```

The queries are then executed using the cursor. Photo and Signature are converted to binary format before storing in the database using the **read_file function**. Old Addresses and phone numbers are deleted and new ones are added in the database instead of updating them because of multiple values. All the execution takes place inside Try and except block and errors are caught by the **Error** class of **mysql.connector**

Updating the data:

```
#adding name, date of birth , date of joining
try:
    cur.execute(employeeQuery,(info['dob'],info['name'],info['doj'],info['id']))
    self.db.commit()

    #adding addresses
    cur.execute("delete from employeeAddress where eno=%s",(info['id'],))
    self.db.commit()

    for addr in info['address']:
        if(addr!=''):
            cur.execute(employeeAddressQuery,(info['id'],addr))
            self.db.commit()

    #adding phone numbers
    cur.execute('delete from employeeTelephone where eno = %s',(info['id'],))
    for num in info['phone']:
        if(num!=''):
            cur.execute(employeeTelephoneQuery,(info['id'],int(num)))
            self.db.commit()

    #adding department
    cur.execute(employeeDepartmentQuery,(info['department'],info['id']))
    self.db.commit()

    #adding photo
    cur.execute(employeePhotoquery,(info['photo'],info['id']))
    self.db.commit()

    #adding signature
    cur.execute(employeeSigquery,(info['sig'],info['id']))
    self.db.commit()
```

Error handling:

```
except Error as e:
    self.Status.setStyleSheet('background:rgb(212,115,70)')

    if e.errno==1146:
        self.Status.setText('Problem with the Database (table does not exist)')

    elif e.errno==1054:
        self.Status.setText('Problem with the Database (column does not exist)')
    else:
        self.Status.setText(str(e))
```

***Validation of fields is same as Add employee's window since all the fields are same.**

3.4.10) DeleteTrainee.py (Delete information about employee)

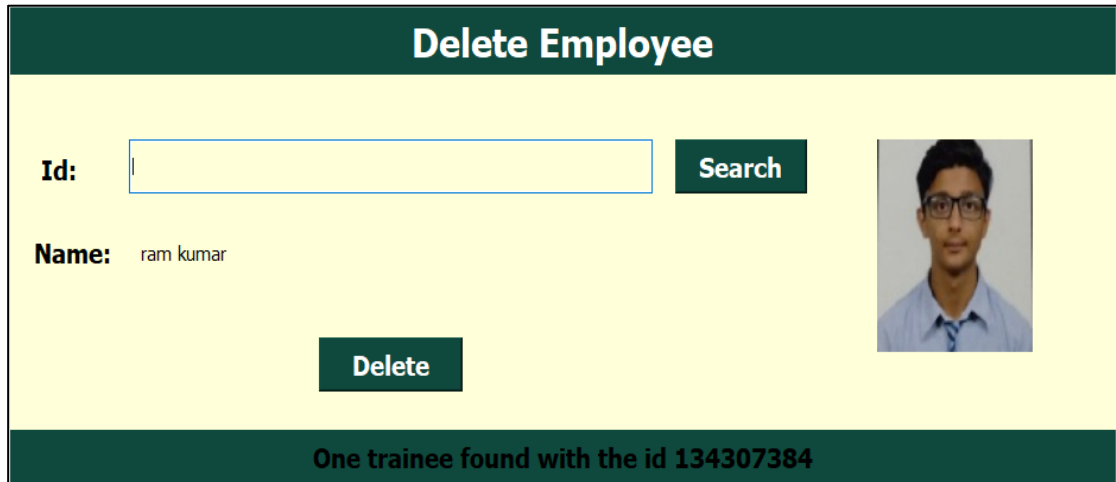


Fig 3.18: Delete employee Window

SQL QUERIES

Searching the employee:

```
try:
    cur.execute('select eno , name from employee where eno="{}".format(self.idEdit.text())')
    trainee=cur.fetchone()
    print(trainee)
    if(trainee):
        cur.execute('select photo from employeePhoto where eno="{}".format(trainee[0])')
        photo=cur.fetchone()[0]
        u.write_file(photo,"temp/deletedTrainee.jpg")
        self.currentId=trainee[0]
```

Deleting the employee:

```
def onDelete(self):
    cur=self.db.cursor()
    if(self.currentId!=''):
        try:
            cur.execute('delete from employee where eno="{}".format(self.currentId)')
            self.db.commit()
```

The existing employee is deleted by the **DELETE** command of MYSQL

3.4.11) AddUser.py

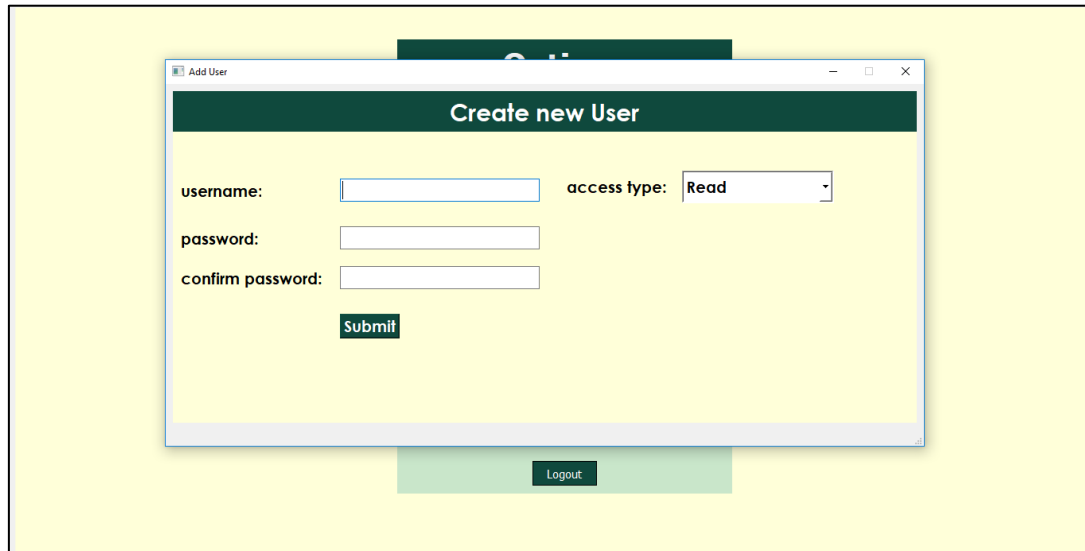


Fig 3.19: Create user window

Create User window basically contains of four fields:

- Username of the user
- Password for the user
- Confirmation of password
- Access rights (read or write)

3.4.11.1) Checking User existence before adding

Before adding any new user in the MySQL server it is necessary to check if the user with that name already exist or not. This is done by the **checkUser ()** function.

```
#checking if the user doesn't already exists
def checkUser(self,user):
    cur=self.db.cursor()
    cur.execute('select user from mysql.user where user="{0}"'.format(user))
    data=cur.fetchall()
    if(not data):
        return True
    else:
        return False
```

The **checkUser** function checks the existence of user from the **MySQL.USER** database. If no data is returned or no user is present with that name, then **true** is returned else **false**.

3.4.11.2) Adding user to the database

```
#creating a user
cur=self.db.cursor()
cur.execute('create user "{}"@"localhost" identified by "{}"'.format(self.usernameEdit.text(),self.passwordEdit.text()))
self.db.commit()

#checking access type for the user
if(self.accessEdit.currentText()=='Read'):
    cur.execute('grant select on *.* to "{}"@"localhost"'.format(self.usernameEdit.text()))

elif(self.accessEdit.currentText()=="Write"):
    cur.execute('grant select,update,insert,delete on *.* to "{}"@"localhost"'.format(self.usernameEdit.text()))

#status GUI
self.status.show()
self.status.setStyleSheet('color:white;background:rgb(15,73,61)')
self.status.setText('{} added successfully with {} access'.format(self.usernameEdit.text(),self.accessEdit.currentText()))
self.clearData()
```

The User is created by using the **CREATE USER** command of MySQL. After adding the user it's time to grant his/her access rights. The GUI checks whether read or write is selected and rights are granted according to that value.

The user is granted access by the **GRANT** command of MySQL.

If access type is **READ** then user is only granted **SELECT** on database.

Else if the access type is **WRITE** then user is granted **INSERT, UPDATE, DELETE, SELECT** on database.

3.4.12) UpdateUser.py

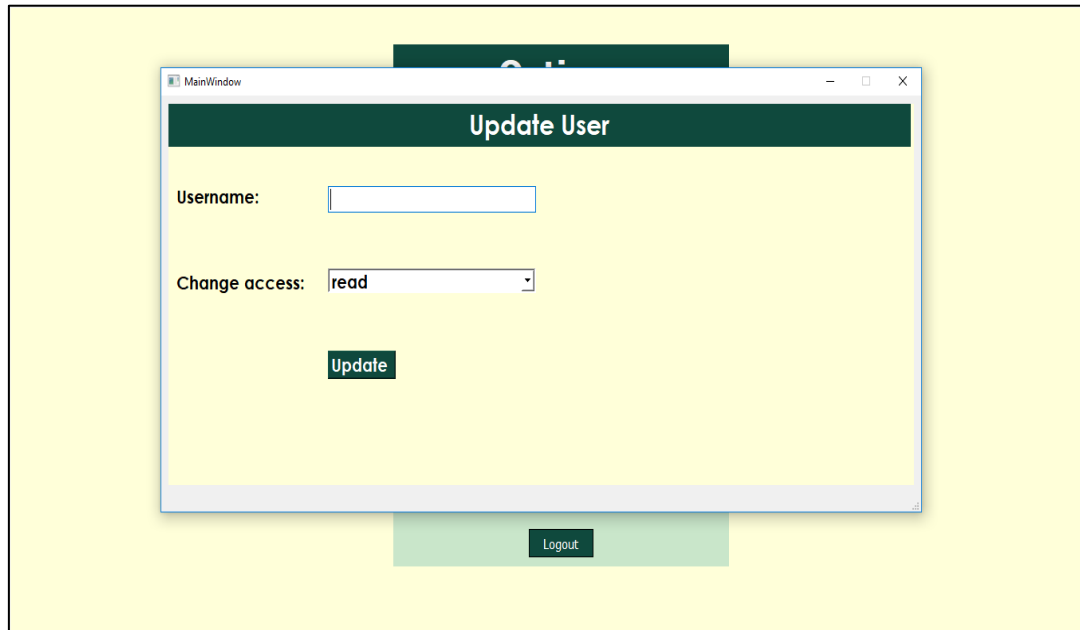


Fig 3.20: Update user window

The Update user window contains two fields:

- Username whose access rights are need to updated.
- Change access to change user grants

```
107
108 def updateDetails(self):
109     if(self.usernameEdit.text()!=''):
110         cur=self.db.cursor()
111         cur.execute('select user from mysql.user where user="{format(self.usernameEdit.text())}'')
112         user=cur.fetchone()
113
```

Before updating the user it is necessary to check If the user exists by the username which is done by **LINE 111**. It checks for the user in the **MYSQL.USER** database and returns the user.

```
114
115     if user:
116         user=user[0]
117         #revoking privileges from the current user
118         cur.execute('revoke all privileges on *.* from "{format(user)}@"localhost"')
119         self.db.commit()
```

If the user is present all its access rights are revoked by the **REVOKE** command of MySQL(Line 117).

When all privileges are revoked from the user, new rights are granted to the user on the basis of the access type chosen. If the access type is **READ** then **SELECT** is granted on the database. Else if the access type is **WRITE** then **INSERT**, **UPDATE**, **DELETE** and **SELECT** are granted on the database.

```
#checking the access type and reassigning the privileges
if(self.accessEdit.currentText()=="read"):
    cur.execute('grant select on *.* to "{}"@localhost".format(user))
    self.db.commit()

elif(self.accessEdit.currentText()=="write"):
    cur.execute('grant select,update,insert,delete on *.* to "{}"@localhost".format(user))
    self.db.commit()
```


3.4.13) DeleteUser.py

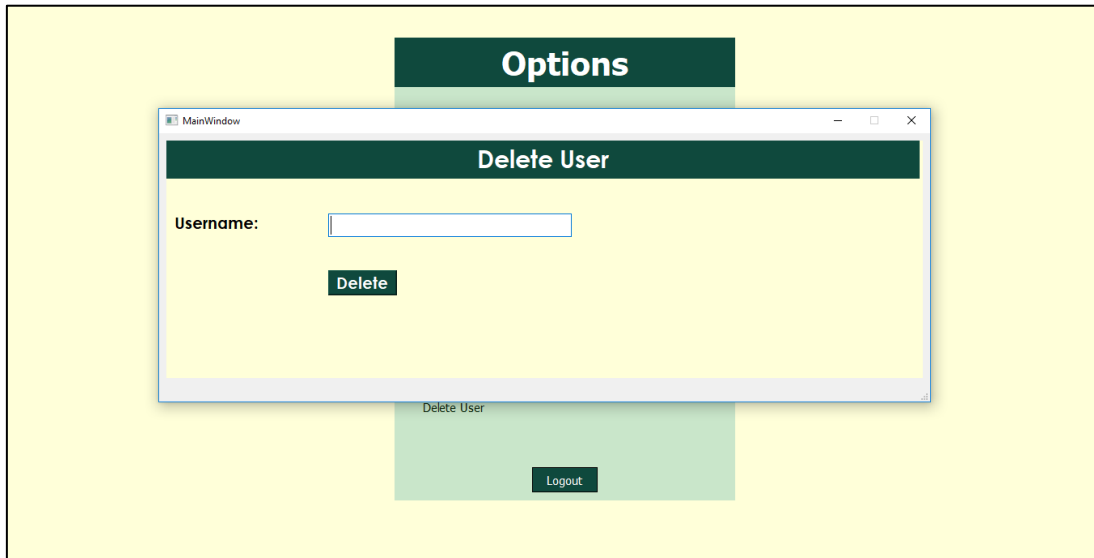


Fig 3.21: Delete user window

Before deleting the user, it is necessary to check if the user exists which is done by:

```
cur=self.db.cursor()
cur.execute('select user from mysql.user where user="{0}"'.format(self.usernameEdit.text()))
user=cur.fetchone()
```

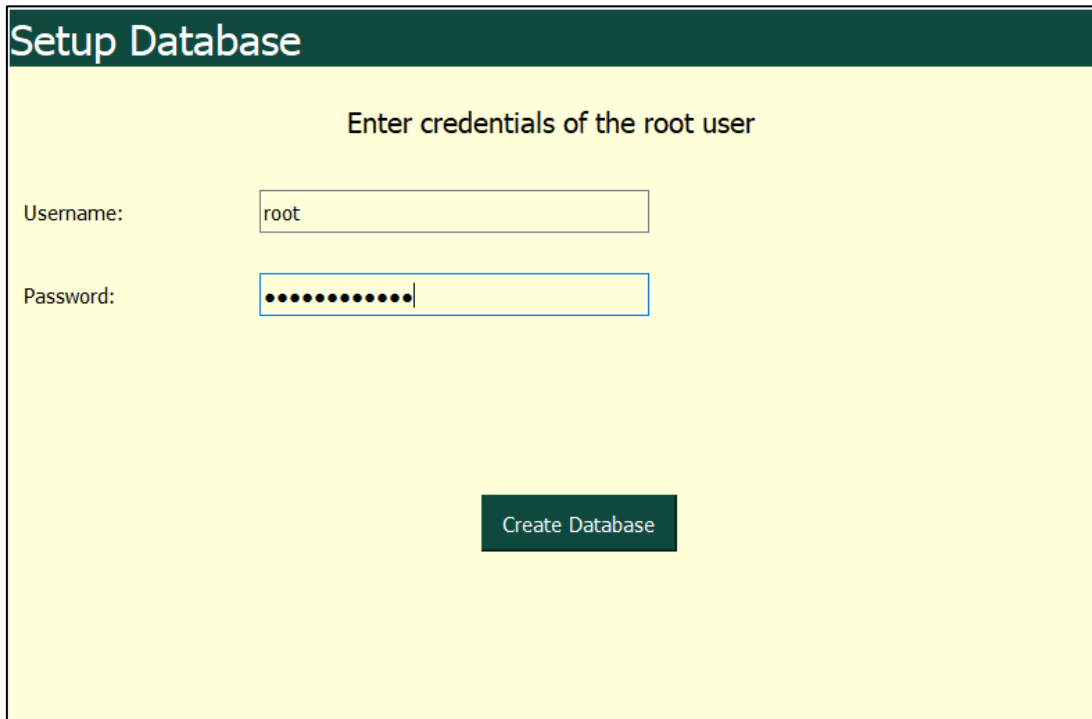
Once the user is found, it is deleted by the **DROP USER** command of MySQL.

```
if user:
    user=user[0]
    cur.execute('drop user "{0}"@"localhost"'.format(user))
    self.db.commit()
```

Errors are handled by the **ERROR** class of **mysql.connector** package of python.

```
except Error as e:
    self.Status.show()
    self.Status.setStyleSheet("background:rgb(212,115,70)")
    self.Status.setText(e)
```

Chapter IV: SCREENSHOTS OF THE PROJECT



The screenshot shows a web form titled "Setup Database" with a dark green header. The main content area has a light yellow background. The heading "Enter credentials of the root user" is centered. Below it, there are two input fields: "Username:" with the value "root" and "Password:" with masked characters. A dark green "Create Database" button is positioned at the bottom center.

Setup Database

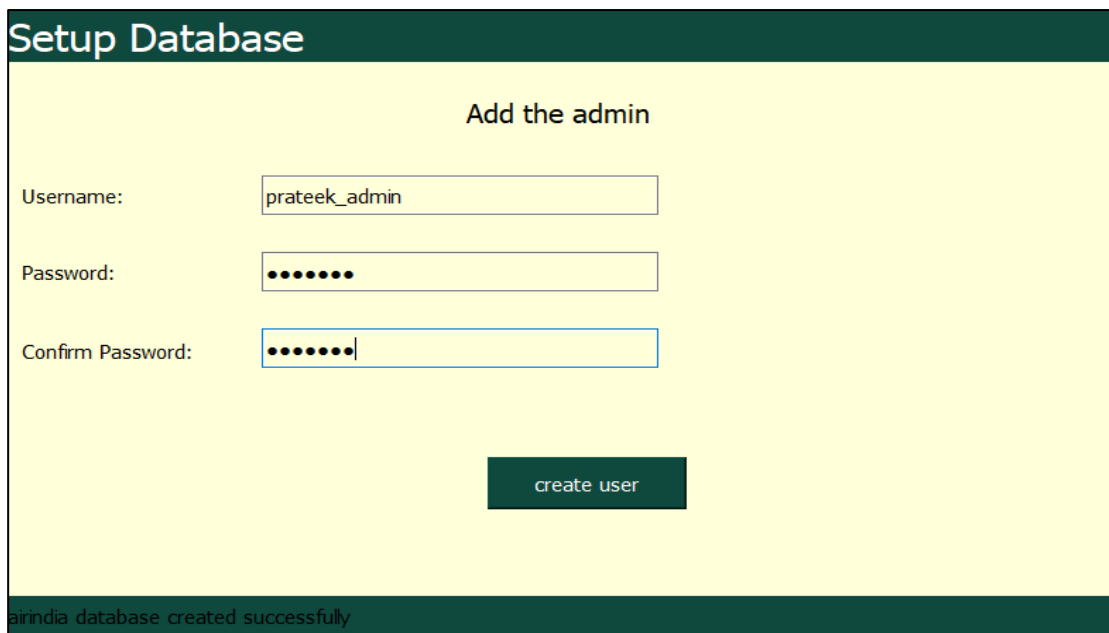
Enter credentials of the root user

Username:

Password:

Create Database

Fig 4.1: Setting up database (snapshots)



The screenshot shows the same "Setup Database" form, but now with the heading "Add the admin". It includes three input fields: "Username:" with the value "prateek_admin", "Password:" with masked characters, and "Confirm Password:" with masked characters. A dark green "create user" button is at the bottom center. A dark green footer bar at the bottom contains the text "airindia database created successfully".

Setup Database

Add the admin

Username:

Password:

Confirm Password:

create user

airindia database created successfully

Fig 4.2: Setting up admin User (snapshots)

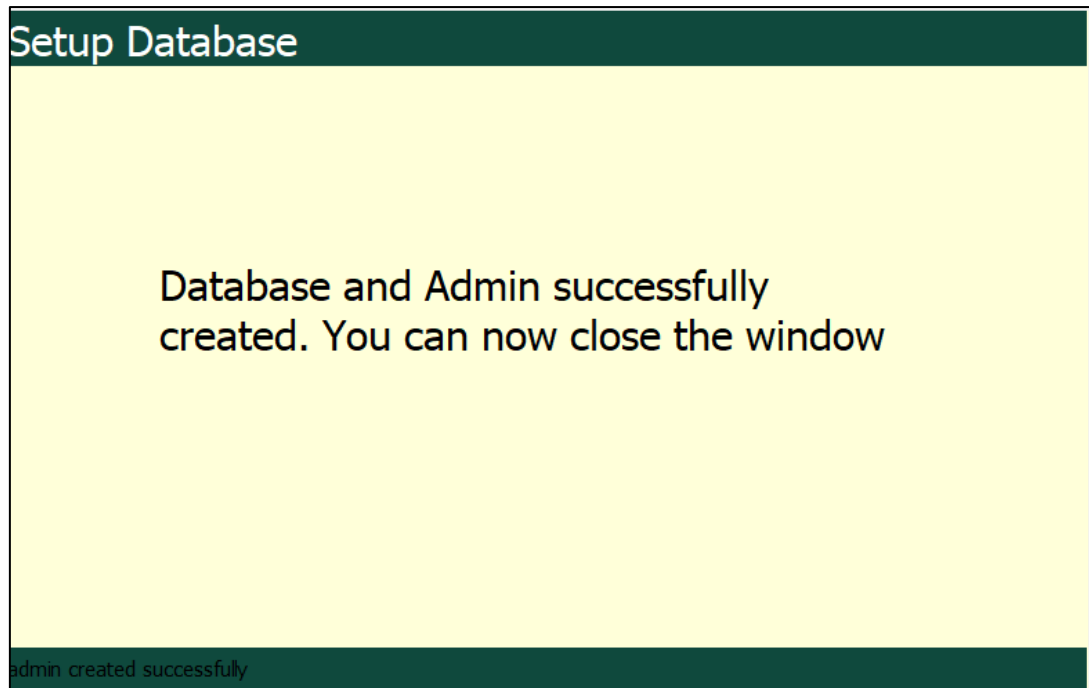


Fig 4.3: successfully created database (snapshots)

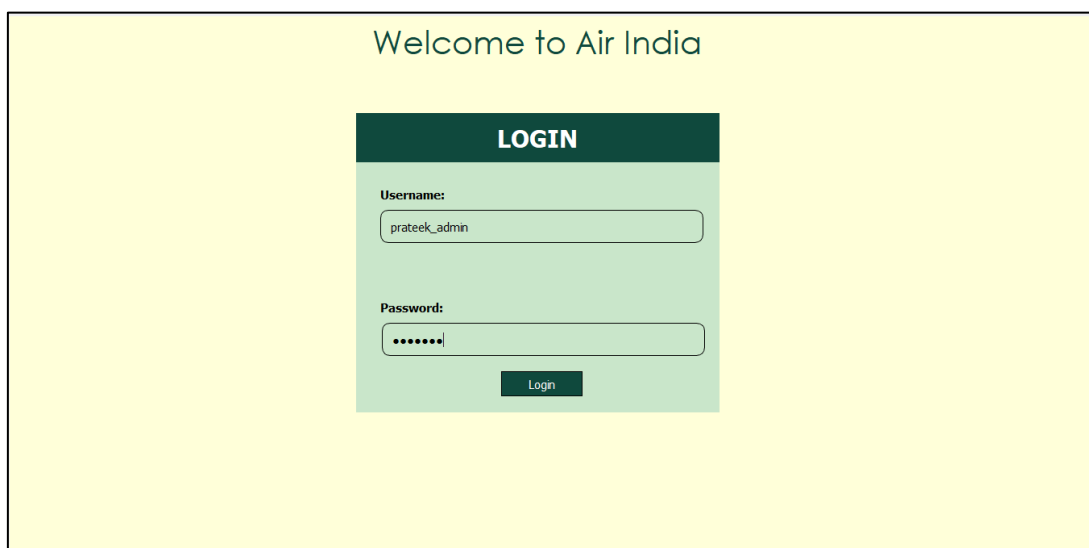


Fig 4.4: Login Page (snapshots)

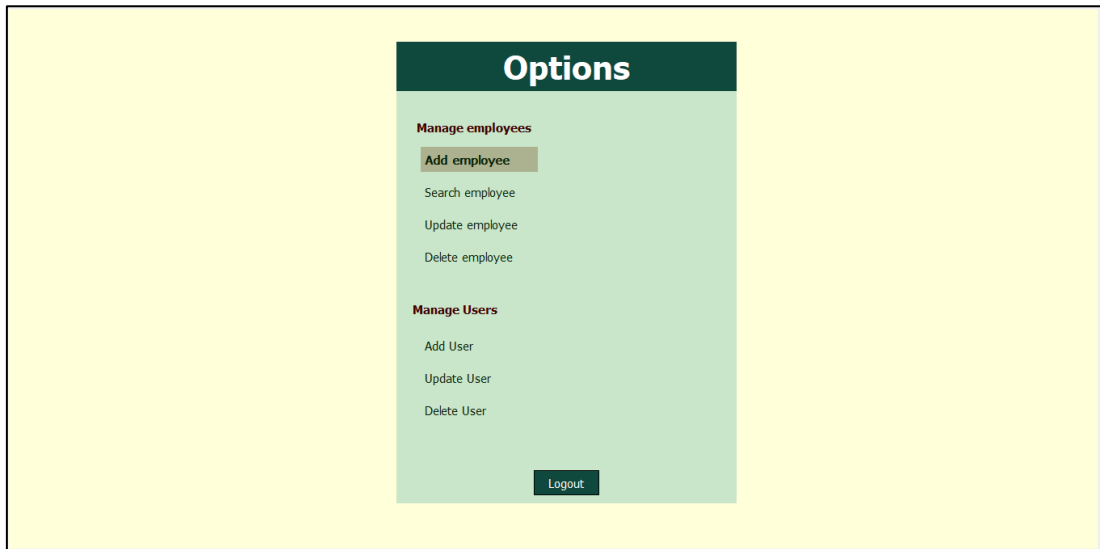


Fig 4.5: Option Page (snapshots)


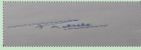
The screenshot shows a web application interface for adding a new employee. The form has a yellow background and includes the following fields and controls:

- Name ***: Text input field containing "prateek". Below it, a red error message says "Name should be atleast 3 characters long".
- Date of Birth ***: Date picker showing "1999-03-21".
- Date of Joining***: Date picker showing "2019-04-22".
- Address 1 ***: Text input field containing "a-156, new ashok nagar, delhi-96". Below it, a red error message says "Address should be atleast 15 characters long".
- Address 2**: Text input field containing "Add address 2 (optional)".
- Phone No. ***: Three text input fields. The first contains "9971930070". The second and third are labeled "(optional)". Below them, a red error message says "A ten-digit phone number is required".
- Select Department ***: Dropdown menu showing "Information Technology".
- Image and Signature**: Two image upload areas. The first shows a photo of a man with a red "Required" label and an "Add Image" button. The second shows a signature with a red "Required" label and an "Add Signature" button.
- Submitted**: A green button at the bottom center.
- Modal Dialog**: A small "Info" dialog box is open, displaying "Userid : 502556216 (Note for future references)" and an "OK" button.
- Footer**: A green bar at the bottom with the text "New Employee added successfully".

Fig 4.6 Add Employee window (snapshots)

Id:

Details



ID :	502556216			
Name:	prateek			
Date of Birth:	21 Mar 1999			
Date of Joining:	22 Apr 2019			
Address 1:	a-156, new ashok nagar, delhi-96			
Address 2:	Not Available			
Phone Numbers :	9971930070	Not Available	Not Available	
Department:	Information Technology			

Found a Employee

Fig 4.7 Search Window (snapshots)

Id:

Details

ID :	<input type="text" value="502556216"/>	Date of Birth:	<input type="text" value="1999-03-21"/>	 change image	
Name:	<input type="text" value="prateek"/>	Date of Joining:	<input type="text" value="2019-04-22"/>		
Address 1:	<input type="text" value="a-156, new ashok nagar, delhi-96"/>				
Address 2:	<input type="text"/>				
Phone Numbers :	<input type="text" value="9971930070"/>	<input type="text" value="8130927793"/>			 change signature
Department:	<input type="text" value="Mechanical Department"/>				

Update

Found a employee


Fig 4.8 Update window (snapshots)

Delete Employee

Id:

Name: prateek

Search



Delete

One trainee found with the id 502556216

Fig 4.9 Delete Window (snapshots)

Create new User

username:

access type:

Write

password:

confirm password:

Submit

Fig 4.10 Add User window (snapshots)

Update User

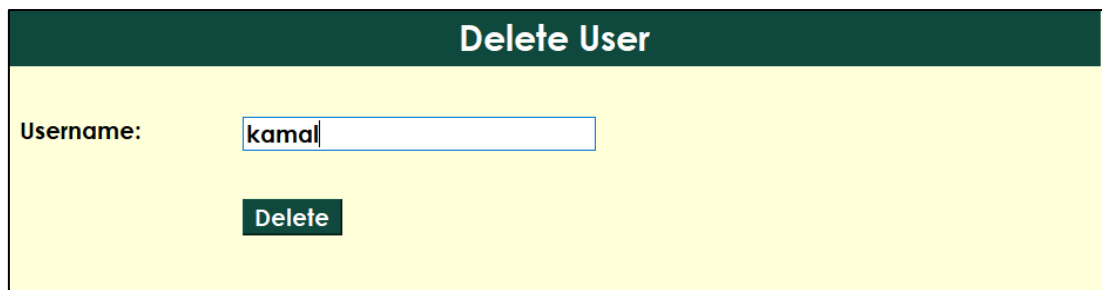
Username:

Change access:

read

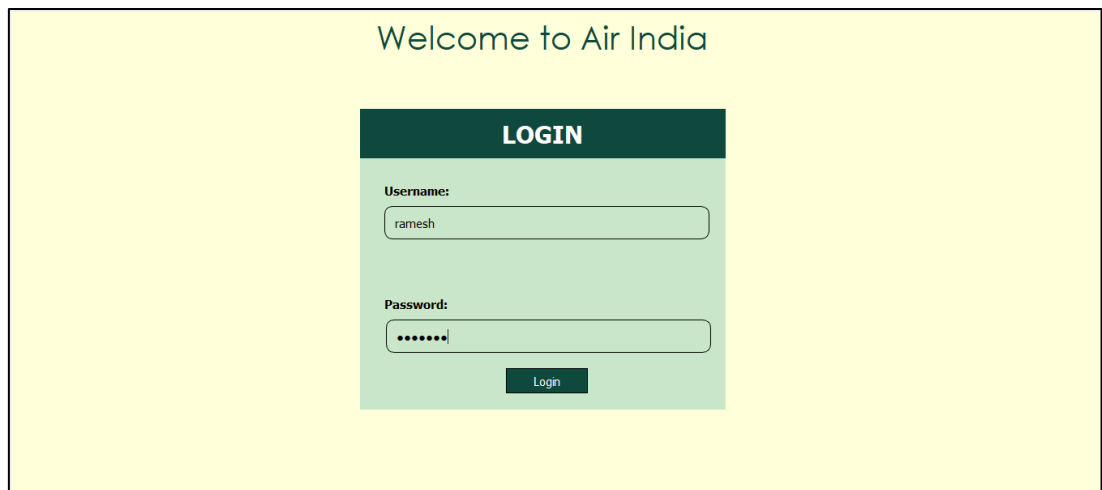
Update

Fig 4.11 Update User window (snapshots)



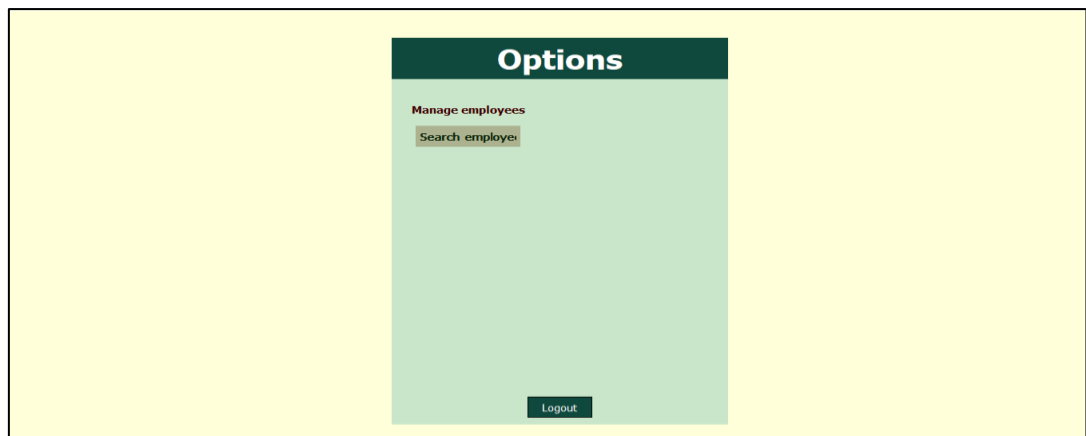
The 'Delete User' window features a dark green header with the title 'Delete User' in white. The main area has a light yellow background. It contains a 'Username:' label, a text input field with 'kama|', and a dark green 'Delete' button.

Fig 4.12 Delete User window (snapshots)



The login window is titled 'Welcome to Air India' in dark green. It contains a 'LOGIN' sub-window with a dark green header. Inside, there are 'Username:' and 'Password:' labels, input fields (the first containing 'ramesh' and the second with masked characters), and a dark green 'Login' button.

Fig 4.13 Login window with read access user (snapshots)



The 'Options' window has a dark green header with the title 'Options' in white. The main area is light yellow. It contains a 'Manage employees' section with a 'Search employee:' label and a text input field. At the bottom right is a dark green 'Logout' button.

Fig 4.14 Option window for user with read rights (snapshots)

REFERENCES

- 1) <https://www.google.com> – Google**
- 2) <https://stackoverflow.com> - Stack Overflow**
- 3) <https://www.python.org> - Python Documentation**
- 4) <https://doc.qt.io> – Qt Designer Documentation**
- 5) <https://www.w3schools.com> – W3Schools**
- 6) <https://dev.mysql.com/doc> - MySQL Documentation**

BOOKS USED:

- Python Crash Course by Matthes Eric