Computer Science

Shubh Mittal

Roll No

—

Project Work

—

Ms. Monica Jain

CERTIFICATE

This is to certify that the Project entitled Library Management System is a Bonafede work done by _____ of class XII Session 2020-21 in partial fulfillment of CBSE's AISSCE Examination 2021 and has been carried out under my direct supervision and guidance. This report or a similar report on the topic has not been submitted for any other examination and does not form a part of any other course undergone by the candidate.

Sign of student Sign of Teacher

Acknowledgement

I wish to express my deep gratitude and sincere thanks to the principal, Dr Sheetal Mann, Modern Convent School for her encouragement and for all facilities she provided for this project work. I sincerely appreciate this magnanimity of hers which she has shown by taking me into her fold, for which I shall remain indebted to her.
I extend my hearty thanks to Ms. Monika Jain (Computer Science teacher at Modern Convent School) who guided me to the successful completion of this project. I take this opportunity to express my deep sense of gratitude for her invaluable guidance, constant encouragement, immense motivation which has sustained my efforts at all stages of my project work.

I cannot forget to offer my profound gratitude to my parents and my classmates who helped me carry out this project work successfully. I would also thank them for their invaluable advice and support which was showered upon me in the time of need.

Prodigy

TOMORROW IS IMAGINARY

BACKGROUND

Based on the MySQL Database and Python

In simple language database is a collection of rows and columns which are related to each other, MySQL being one of them, This project was not possible with the powerful tools provided by python 3.8.

An important aspect of this project is database designing, the way in which the database is designed for this project provides the backbone for the same. It can be found in the later section.

Objective

Prodigy is an app made with the help of kivymd (based on kivy) for python. The main objective of the app is to help students who are stuck to social media to add reminders of their day to day tasks in a simple yet innovative manner.

The main focus of the app is to be simple and easy to navigate, the server can be hosted on any machine that supports python and has a screen, users who added reminders will receive them instantly on their smartphones through the means of social media.

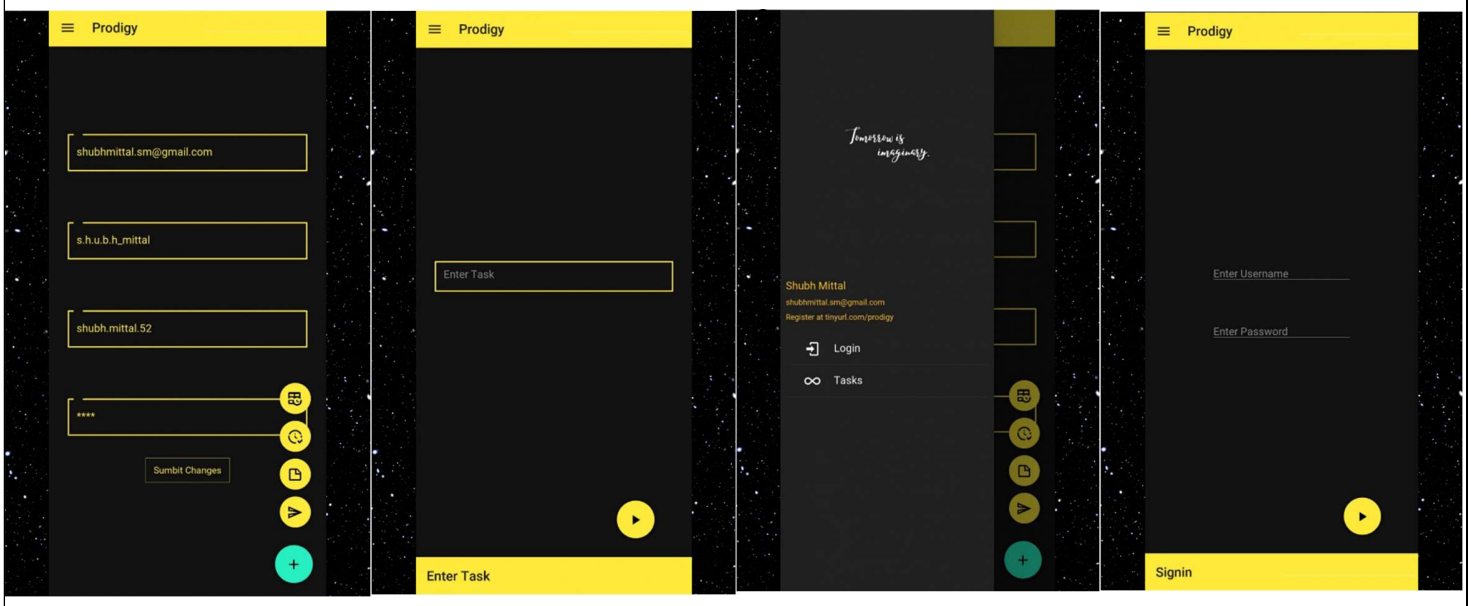
Database and Python information along with screenshots.

- The project consists of two tables one containing name of users and other containing the tasks entered by each user.

```
mysql> describe users;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| name   | varchar(100) | NO   | PRI | NULL    |       |
| fb     | varchar(100) | YES  |     | NULL    |       |
| insta  | varchar(100) | YES  |     | NULL    |       |
| email  | varchar(100) | YES  |     | NULL    |       |
| passwd | varchar(100) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

```
+-------+----------------+------+-----+---------+-------+
| Field | Type           | Null | Key | Default | Extra |
+-------+----------------+------+-----+---------+-------+
| name  | varchar(100)   | YES  | MUL | NULL    |       |
| task  | varchar(8000)  | YES  |     | NULL    |       |
| date  | date           | YES  |     | NULL    |       |
| time  | time           | YES  |     | NULL    |       |
+-------+----------------+------+-----+---------+-------+
```

- The ‘name’ column is the primary key for both the tables.
- The screen-shots of the app are below.



- The requirements are as follows:
    - Python3
    - Selenium
    - Chromedriver
    - Mysql.connector
    - Pandas
    - Kivy=1.11.1
    - Kivymd=Master
    - Yagmail
    - Buildozer
- The python scripts are as follows:

1>Main Server

```
# importing libraries
import datetime
import csv
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
import multiprocessing
import threading
import time
import pandas as pd
import pickle
import mysql.connector as ms
import yagmail

# loading cookies

cookie2 = []
cookie1 = []
cookies_ = pickle.load(open("fb.pkl", "rb"))
for cookie in cookies_:
    cookie2.append(cookie)
cookies_ = pickle.load(open("insta.pkl", "rb"))
for cookie in cookies_:
    cookie1.append(cookie)
curr_data = []
active_data = []
pending_task = []


# Function for sending email
```

```python
def emai(active_data):
    yag = yagmail.SMTP(user="prodigy.tii@gmail.com", password="omnmh2003shubh")
    # sending the email
    for i in active_data:
        try:
            yag.send(to=i["email"], subject="Prodigy Reminder", contents=i["task"])
            print("Email sent successfully")
        except:
            print("Email Error")
            pass


# function for facebook
def facebook(active_data, cookie):

    chrome_options = Options()

    # initializing the selenium browser
    chrome_options.add_argument("--disable-gpu")
    chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(
        executable_path="chromedriver.exe", options=chrome_options
    )
    print("Getting Link!")
    driver.get("https://www.facebook.com")
    m = [driver.add_cookie(i) for i in cookie]
    # print("Sending message!")
    # Iterating over given data
    for i in active_data:
        driver.get(f"https://www.facebook.com/messages/t/{i['fb']}")

        element = WebDriverWait(driver, 100).until(
            EC.element_to_be_clickable(
                (
                    By.XPATH,
                    '//div[@data-contents="true"]//div[@data-block="true"]',
                )
            )
        )
        element.send_keys(i["task"])
        element.send_keys(Keys.ENTER)
        # print("Done")
    driver.close()
    return


# function for instagram
```

```python
def instagram(active_data, cookie):
    chrome_options = Options()
    chrome_options.add_argument("--disable-gpu")
    chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(
        executable_path="chromedriver.exe", options=chrome_options
    )
    # print("Initiating")
    driver.get("https://instagram.com")
    m = [driver.add_cookie(i) for i in cookie]
    # print("Login Successfull")
    # iterating over given data
    for i in active_data:
        driver.get(f"https://www.instagram.com/{i['insta']}")
        print("Sending Message")
        element = WebDriverWait(driver, 100).until(
            EC.element_to_be_clickable(
                (By.XPATH, '//button[@type="button"][text()="Message"]')
            )
        )
        element.click()
        element = WebDriverWait(driver, 100).until(
            EC.element_to_be_clickable(
                (By.XPATH, '//textarea[@placeholder="Message..."]')
            )
        )
        element.send_keys(i["task"])
        element.send_keys(Keys.ENTER)
        print("Completed!")
    # Waiting for message to be sent
    while True:
        try:
            element = WebDriverWait(driver, 5).until(
                EC.elements_to_be_clickable(
                    (By.XPATH, '//div[@class="FeN85  xTVtN aQWyo"]')
                )
            )
        except:
            break
            driver.close()
        return


def connector(curr_data):
    global fields, pending_task, active_data, db, cursor, cookie1, cookie2, instagram, facebo
ok
    temp_list = []
```

```
    all_done = []

    while True:
        # Connecting to database
        try:
            db = ms.connect(
                host="logs.c7xtjtjv8ph3.ap-south-1.rds.amazonaws.com",
                port=3306,
                user="shubh",
                passwd="shubh2003",
                db="tasker",
            )
            cursor = db.cursor()
            cursor.execute(
                "select users.*,task.date,task.time,task.task from users,task where DATE(task
.date)=DATE(NOW()) and task.name=users.name order by task.time;"
            )
            b = cursor.fetchall()
            if len(b) > len(curr_data):
                cursor.execute(
                    "select users.*,task.date,task.time,task.task from users,task where DATE(
task.date)=DATE(NOW()) and task.name=users.name order by task.time;"
                )
                fields = cursor.column_names
                temp = []
                while True:
                    try:
                        temp.append(dict(zip(cursor.column_names, cursor.fetchone())))
                    except:
                        break
                for j in temp:
                    for k in fields:
                        if j[k] == None:
                            j[k] = ""
                        else:
                            j[k] = str(j[k])
                for i in temp:
                    if i not in curr_data:
                        curr_data.append(i)
                print(
                    f"Statistics:\nAll tasks for the day:\n{pd.DataFrame.from_dict(curr_data)
}"
                )
                active_task = [
                    i
                    for i in curr_data
                    if i["time"] == datetime.datetime.now().strftime("%H:%M:00")
```

```
            ]
            active_task = [i for i in active_task if i not in all_done]
            for i in active_task:
                all_done.append(i)
            if len(active_task) > 0:
                print("Starting Tasks:")
                active_task = active_task[::-1]
                print(f"{pd.DataFrame.from_dict(active_task)}")
                a = active_task
                insta_task = multiprocessing.Process(
                    target=instagram, args=(active_task, cookie1)
                )
                fab_task = multiprocessing.Process(
                    target=facebook, args=(active_task, cookie2)
                )
                ema_task = multiprocessing.Process(target=emai, args=(active_task,))
                insta_task.daemon = True
                fab_task.daemon = True
                ema_task.daemon = True
                insta_task.start()
                fab_task.start()
                ema_task.start()
                active_task.clear()
                print("startttt")
                continue
        else:
            active_task = [
                i
                for i in curr_data
                if i["time"] == datetime.datetime.now().strftime("%H:%M:00")
            ]
            active_task = [i for i in active_task if i not in all_done]
            for i in active_task:
                all_done.append(i)
            if len(active_task) > 0:
                print("Starting Tasks:")
                active_task = active_task[::-1]
                print(f"{pd.DataFrame.from_dict(active_task)}")
                a = active_task
                insta_task = multiprocessing.Process(
                    target=instagram, args=(active_task, cookie1)
                )
                fab_task = multiprocessing.Process(
                    target=facebook, args=(active_task, cookie2)
                )
                insta_task.daemon = True
                fab_task.daemon = True
```

```
                    insta_task.start()
                    fab_task.start()
                    active_task.clear()
                    print("startttt")
                    continue
        except Exception as e:
            with open("error_log.txt", "a+") as f:
                f.write(str(e))
                f.write("\n")
            print("Exception")
            continue


def time_checker(curr_data, active_data, pending_task):
    print("Started 1")
    global fields
    temp = [
        i
        for i in curr_data
        if i["time"] == datetime.datetime.now().strftime("%H:%M:00")
    ]
    for i in temp:
        if i not in pending_task:
            pending_task.append(i)
            active_data.append(i)
    starter(active_data, curr_data)


def startert(insta, fb, active_data, curr_data):
    global cookie1, cookie2
    insta_task = multiprocessing.Process(target=insta, args=(active_data, cookie1))
    fab_task = multiprocessing.Process(target=fb, args=(active_data, cookie2))
    insta_task.start()
    fab_task.start()


if __name__ == "__main__":
    connector(curr_data)
```

2>Registration Server

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
import multiprocessing
import mysql.connector as ms
import pickle


def tasko(usero):
    chrome_options = Options()
    chrome_options.add_argument("--disable-gpu")
    chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(
        executable_path="chromedriver.exe", options=chrome_options
    )
    driver.get("https://instagram.com")
    # pickle.dump(driver.get_cookies(), open("insta.pkl", "wb"))
    cookies = pickle.load(open("insta.pkl", "rb"))

    for cookie in cookies:
        driver.add_cookie(cookie)
    for user in usero:
        driver.get(f"https://instagram.com/{user}")
        try:
            element = WebDriverWait(driver, 100).until(
                EC.element_to_be_clickable((By.XPATH, '//button[text()="Follow"]'))
            )
            element.click()
        except:
            pass


def connector():
    while True:
        db = ms.connect(
            host="logs.c7xtjtjv8ph3.ap-south-1.rds.amazonaws.com",
            port=3306,
            user="shubh",
            passwd="shubh2003",
            db="tasker",
        )
```

```
        cursor = db.cursor()
        cursor.execute("select insta from users;")
        b = cursor.fetchall()
        users = [i[0] for i in b]
        # print(users)
        with open("insta_users.txt", "r+") as f:
            curr_users = f.readlines()
            curr_users = [i.replace("\n", "") for i in curr_users]
        new_users = [i for i in users if i not in curr_users]
        if len(new_users) == 0:
            pass
        else:
            print("FOLLOWING THE FOLLOWING ACCOUNT xD:")
            print(new_users)
            with open("insta_users.txt", "a+") as f:
                for i in new_users:
                    f.write(i)
                    f.write("\n")
            taske = multiprocessing.Process(target=tasko, args=(new_users,))
            taske.daemon = True
            taske.start()
            new_users.clear()
            continue


if __name__ == "__main__":
    connector()
```

3>Client App

```
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import ObjectProperty
from kivy.uix.screenmanager import ScreenManager, Screen
from kivymd.app import MDApp
from kivymd.uix.picker import MDTimePicker, MDDatePicker

# from kivy.core.window import Window
from kivymd.uix.datatables import MDDataTable
from kivy.metrics import dp
import webbrowser
import mysql.connector as ms
from datetime import datetime
import datetime as dt

try:
    from android.permissions import request_permissions, Permission

    request_permissions(
        [
            Permission.INTERNET,
            Permission.READ_EXTERNAL_STORAGE,
            Permission.WRITE_EXTERNAL_STORAGE,
        ]
    )
except:
    pass
# Window.size = (300, 500)
task = ""
time_ = ""
user = ""
passwd_ = ""
date_ = ""


class Mainscreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def login(self):
        global user, passwd_
        self.ids.bar.start()
        user_ = self.ids.username
        pass_ = self.ids.passw
```

```
        user = user_.text
        passw = pass_.text
        if user == "":
            return
        # print(user,passw)
        try:
            # self.ids.bar.stop()
            self.db = ms.connect(
                host="logs.c7xtjtjv8ph3.ap-south-1.rds.amazonaws.com",
                port=3306,
                user="shubh",
                passwd="shubh2003",
                db="tasker",
            )
            self.db.autocommit = True
            self.cursor = self.db.cursor()
            self.cursor.execute(
                f'select passwd,email,insta,fb from users where name="{user}";'
            )
            a = self.cursor.fetchall()
        except:
            self.ids.bar.start()
            return
        # db.close()
        # print(len(a))
        for i in a:
            for k in i:
                if k == None:
                    k = ""
        if len(a) == 0:
            user_.focus = True
            user_.error = True
            user_.text = ""
            # print(type(a))
        elif passw != a[0][0]:
            user_.error = False
            pass_.focus = True
            pass_.error = True
            pass_.text = ""
        else:
            user_.error = False
            pass_.error = False
            self.ids.passwd_.text = a[0][0]

            self.ids.email.text = a[0][1]
            self.ids.insta.text = a[0][2]
            self.ids.fb.text = a[0][3]
```

```
        self.ids.fb.text = a[0][3]
        self.ids.bar.stop()
        self.ids.screen_manager.current = "scr 2"
        self.table()

    def show_time_picker(self):
        global user
        if user == "":
            pass
        time_dialog = MDTimePicker()
        time_dialog.bind(time=self.get_time)
        time_dialog.open()

    def get_time(self, instance, time):
        global time_
        """
        The method returns the set time.

        :type instance: <kivymd.uix.picker.MDTimePicker object>
        :type time: <class 'datetime.time'>
        """
        sel_time = time
        time_ = str(time)
        # date = datetime.datetime.strptime('2018-01-01', '%Y-%m-%d').date()
        curr_time = datetime.strptime(
            str(datetime.now().strftime("%H:%M:00")), "%H:%M:00"
        ).time()
        # print(sel_time,curr_time)
        if sel_time > curr_time:
            strd = str(dt.datetime.now().strftime("%Y:%m:%d"))
            self.show_date_picker(datetime.strptime(strd, "%Y:%m:%d").date())
        else:
            strd = str((dt.datetime.now() + dt.timedelta(days=1)).strftime("%Y:%m:%d"))
            # print(strd)
            self.show_date_picker(datetime.strptime(strd, "%Y:%m:%d").date())
        # return time

    def show_date_picker(self, min):
        min_date = min
        max_date = datetime.strptime("2050:12:12", "%Y:%m:%d").date()
        dats = str(min).split("-")
        # print(str(min))
        date_dialog = MDDatePicker(
            callback=self.get_date,
            min_date=min_date,
            max_date=max_date,
            year=int(dats[0]),
```

```
            month=int(dats[1]),
            day=int(dats[2]),
        )
        date_dialog.open()
        return

    def get_date(self, date):
        global date_
        """
        :type date: <class 'datetime.date'>
        """
        date_ = str(date)
        return

    def insert_task(self):
        global time_, date_, task, user
        if time_ == "" or date_ == "" or task == "" or user == "":
            print("eff")
        else:
            try:
                self.ids.bar2.stop()
                self.db.ping(reconnect=True, attempts=1)
                self.cursor.execute(
                    f"insert into task values('{user}','{task}','{date_}','{time_}')"
                )
                time_ = date_ = task = ""
                self.table()
                return
            except:
                self.ids.bar2.start()
                return

    def callback(self, instance):
        if instance.icon == "clock-check-outline":
            self.show_time_picker()
            return
        elif instance.icon == "note-outline":
            self.ids.screen_manager.current = "data entry"
            return
        elif instance.icon == "timetable":
            # print('hehehehe')
            self.show_table()
            return
        elif instance.icon == "send-outline":
            self.insert_task()
            return
```

```python
    def grabtask(self):
        global task
        task_ = self.ids.task
        task = str(task_.text)
        # print(task)
        self.ids.screen_manager.current = "scr 2"
        return

    def table(self):
        # rw=()
        row_ = []
        global user
        try:
            self.ids.bar2.stop()
            self.db.ping(reconnect=True, attempts=1)
            self.cursor.execute(
                f"select task.date,task.time,task.task from task where name='{user}';"
            )
            r = self.cursor.fetchall()
        except:
            self.ids.bar2.start()
            return
            # db.close()
        if len(r) == 0:
            return
        for i in r:
            rw = ()
            for k in i:
                rw += (str(k),)
            row_.append(rw)
        # return
        self.data_table = MDDataTable(
            size_hint=(0.9, 0.6),
            column_data=[
                ("Date", dp(20)),
                ("Time", dp(20)),
                ("Task", dp(30)),
            ],
            row_data=row_,
            use_pagination=False,
        )
        # print(row_)
        return

    def show_table(self):
        try:
            self.data_table.open()
```

```
            return
        except:
            return

    def change_data(self):
        global user
        if user == "":
            return
        ema = self.ids.email
        f = self.ids.fb
        pas = self.ids.passwd_
        ins = self.ids.insta
        email = ema.text
        paswd_ = pas.text
        fb = f.text
        insta = ins.text
        try:
            self.ids.bar2.stop()
            self.db.ping(reconnect=True, attempts=1)
            self.cursor.execute(
                f'update users set email="{email}",insta="{insta}",fb="{fb}",passwd="{paswd_}
" where name="{user}";'
            )
            return
        except:
            self.ids.bar2.start()
            # db.close()
            return


class ContentNavigationDrawer(BoxLayout):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def register(self):
        webbrowser.open("https://abhivyakti-evolve.com/prodigy.html")
        return


class ProdigyApp(MDApp):
    screen_manager = ObjectProperty()
    nav_drawer = ObjectProperty()
    data = {
        "timetable": "Current Tasks",
        "clock-check-outline": "Set Time",
        "note-outline": "Set Task",
        "send-outline": "Send",
```

```
    }

    def build(self):
        self.theme_cls.primary_palette = "Yellow"
        # self.theme_cls.primary_hue = "900"
        self.theme_cls.theme_style = "Dark"
        self.theme_cls.accent_palette = "Green"
        b = Mainscreen()
        return b


if __name__ == "__main__":
    a = ProdigyApp()
    a.run()
```

4>. kv File

```
<ContentNavigationDrawer>:
    orientation: 'vertical'
    padding: "8dp"
    spacing: "8dp"
    Image:
        id: avatar
        size_hint: (1,1)
        source: "mine.png"
    MDLabel:
        text: "Shubh Mittal"
        font_style: "Subtitle1"
        theme_text_color: "Custom"
        text_color: app.theme_cls.primary_dark
        # 0/256, 181/256, 204/256, 1
        size_hint_y: None
        height: self.texture_size[1]
    MDLabel:
        text: "shubhmittal.sm@gmail.com"
        size_hint_y: None
        theme_text_color: "Custom"
        text_color:app.theme_cls.primary_dark
        font_style: "Caption"
        height: self.texture_size[1]
    ScrollView:

        MDList:
            OneLineIconListItem:
                text: "Register"
                on_press:
                    root.register()
                    #root.screen_manager.current = "scr 2"
                IconLeftWidget:
                    icon:'account'
            OneLineIconListItem:
                text: "Login"
                on_press:
                    root.nav_drawer.set_state("close")
                    root.screen_manager.current = "login_screen"
                IconLeftWidget:
                    icon:'login'

            OneLineIconListItem:
                text: "Tasks"
                on_press:
```

```
                    root.nav_drawer.set_state("close")
                    root.screen_manager.current = "scr 2"
                IconLeftWidget:
                    icon:'infinity'


<Mainscreen>:
    size_hint_x:1
    size_hint_y:1
    MDToolbar:
        id: toolbar
        pos_hint: {"top": 1}
        title:'Prodigy'
        halign:'center'
        elevation: 10
        left_action_items: [["menu", lambda x: nav_drawer.set_state("open")]]
        MDProgressBar:
            id:bar2
            color:[1,0,0,1]
            pos_hint: {"center_x": .5, "center_y": .45}
            type: "indeterminate"

    NavigationLayout:
        x: toolbar.height

        ScreenManager:
            id: screen_manager

            Screen:
                name: "login_screen"
                id:sc1
                MDTextField:
                    id:username
                    pos_hint:{'center_x': 0.5, 'center_y': 0.55}
                    hint_text: "Enter Username"
                    helper_text:"Invalid Username"
                    hiny_text_color:[1,1,1,1]
                    required: True
                    # rgba(0, 230, 64, 1)
                    # mode:'rectangle'
                    size_hint_x:0.5
                    active_line:True
                    line_color_normal:[1,1,1,1]
                    # width:300
                    helper_text_mode:'on_error'
                    color_mode: 'custom'
                    on_text_validate:passw.focus=True
```

```
            line_color_focus: 0/256, 230/256, 64/256, 1
        MDTextField:
            id:passw
            pos_hint:{'center_x': 0.5, 'center_y': 0.45}
            hint_text: "Enter Password"
            helper_text:"Invalid Password"
            hiny_text_color:[1,1,1,1]
            required: True
            password:True
            on_text_validate:root.login()
            # rgba(0, 230, 64, 1)
            # mode:'rectangle'
            size_hint_x:0.5
            # active_line:True
            line_color_normal:[1,1,1,1]
            # width:300
            helper_text_mode:'on_error'
            color_mode: 'custom'
            line_color_focus: 0/256, 230/256, 64/256, 1
        MDBottomAppBar:
            MDToolbar:
                title: "Signin"
                elevation:10
                icon: "play"
                round:'50dp'
                type: "bottom"
                # icon_color:[256,0,0]
                mode:'free-end'
                # left_action_items: [["menu", lambda x: x]]
                on_action_button: root.login()
                # root.state = "stop" if root.state == "start" else "start"
                MDProgressBar:
                    id:bar
                    color:[1,0,0,1]
                    pos_hint: {"center_x": .5, "center_y": .45}
                    type: "indeterminate"

    Screen:
        name: "scr 2"
        id:sc2
        MDFloatingActionButtonSpeedDial:
            callback: root.callback
            data:app.data
            bg_hint_color: app.theme_cls.primary_darkest
            id:but
            label_text_color:[1,1,1,1]
            hint_animation: True
```

```
                bg_hint_color:[0,0,0,1]
                bg_color_root_button:[41/256,241/256,195/256,1]
            MDTextField:
                pos_hint:{'top':(0.85-toolbar.height/root.height),'center_x':0.5}
                size_hint_x:0.8
                id:email
                helper_text: "Your Email Id"
                helper_text_mode: "on_focus"
                required: True
                mode: "rectangle"
            MDTextField:
                pos_hint:{'top':(0.7-toolbar.height/root.height),'center_x':0.5}
                size_hint_x:0.8
                helper_text: "Your Insta Id"
                helper_text_mode: "on_focus"
                required: True
                id:insta
                mode: "rectangle"
            MDTextField:
                pos_hint:{'top':(0.55-toolbar.height/root.height),'center_x':0.5}
                size_hint_x:0.8
                helper_text: "Your Facebook Id"
                helper_text_mode: "on_focus"
                mode: "rectangle"
                required: True
                id:fb
            MDTextField:
                pos_hint:{'top':(0.4-toolbar.height/root.height),'center_x':0.5}
                size_hint_x:0.8
                helper_text: "Your Password"
                helper_text_mode: "on_focus"
                mode: "rectangle"
                password:True
                required: True
                id:passwd_
            MDRectangleFlatButton:
                pos_hint:{'top':(0.3-toolbar.height/root.height),'center_x':0.5}
                # size_hint_x:0.8
```

```
                text:'Submit Changes'
                on_release:root.change_data()
        Screen:
            name:'data entry'
            MDTextField:
                id:task
                pos_hint:{'center_x': 0.5, 'center_y': 0.55}
                hint_text: "Enter Task"
```

```
                helper_text:"Invalid Task"
                hiny_text_color:[1,1,1,1]
                helper_text_mode:'on_error'
                required: True
                icon:'note-outline'
                # rgba(0, 230, 64, 1)
                mode:'rectangle'
                on_text_validate:root.grabtask()
                # line_color_normal: [1,1,0,1]
                size_hint_x:0.8
                # on_text: root.set_list_md_icons(self.text, True)
            MDBottomAppBar:
                MDToolbar:
                    title: "Enter Task"
                    elevation:10
                    icon: "play"
                    round:'50dp'
                    type: "bottom"
                    # icon_color:[256,0,0]
                    mode:'free-end'
                    # left_action_items: [["menu", lambda x: x]]
                    on_action_button: root.grabtask()
                    # root.state = "stop" if root.state == "start" else "start"
    MDNavigationDrawer:
        id: nav_drawer

        ContentNavigationDrawer:
            screen_manager: screen_manager
            nav_drawer: nav_drawer
```

5> .html file for registration

```
<!DOCTYPE html>
<html>

<head>
    <title>Prodigy</title>

    <link rel="icon" href="Prodigy.ico" type="image/icon type">

    <style>
        body {
            background: #040404;
        }

        .tb {
            width: 25%;
            height: 35px;
            margin-left: 37.5%;
            text-align: center;
            margin-bottom: 15px;
            background: #040404;
            border-style: solid;
            border-color: #ffde59;
            border-radius: 5px;
            color: #ffde59;
            transition: 0.25s;
        }

        .btn {
            width: 25.75%;
            height: 40px;
            margin-left: 37.5%;
            text-align: center;
            background: #ffde59;
            color: #040404;
            border-style: solid;
            border-width: 1px;
            border-radius: 5px;
            transition: 0.25s;
        }

        .btn:hover {
            width: 26.25%;
            height: 42px;
            margin-left: 37.25%;
            color: #ffde59;
```

```
            background: #040404;
        }

        .form-container {
            margin-top: 12.5%;
        }

        .input_1:hover {
            width: 26%;
            margin-left: 37%;
            height: 37px;
        }

        .img {
            width: 10%;
            position: fixed;
            bottom: 10px;
            left: 89%;
        }
    </style>
</head>

<body>
    <div class="form-container">
        <form name="form" method="POST" action="./html-connect.php">
            <input type="text" name="tb_name" placeholder="Your Name" id="tb_name" class="tb
input_1" required>
            <input type="text" name="tb_email" placeholder="Your Email" id="tb_email" class="
tb input_1" required>
            <input type="text" name="tb_instagram" placeholder="Your Instagram Username" id="
tb_instagram"
                class="tb input_1" required>
            <input type="text" name="tb_facebook" placeholder="Your Facebook Username" id="tb
_facebook"
                class="tb input_1" required>
            <input type="password" name="password" placeholder="Your Password" id="password"
class="tb input_1"
                required>
            <br>
            <input type="submit" name="submit-btn" value="Submit" id="submit-
btn" class="btn">
        </form>
    </div>
    <img src="Prodigy.png" class="img">
</body>

</html>
```

6> .php file for html

```
<?php
// connecting to the database.
$con = mysqli_connect('logs.c7xtjtjv8ph3.ap-south-
1.rds.amazonaws.com', 'shubh', 'shubh2003','tasker');
// header("Location: ./index.html");
// Retrieving the records.
$name = $_POST['tb_name'];
$email = $_POST['tb_email'];
$instagram = $_POST['tb_instagram'];
$facebook = $_POST['tb_facebook'];
$password = $_POST['password'];

// database insert SQL code, change accordingly.
$sql = "INSERT INTO users VALUES ('$name','$facebook','$instagram','$email','$password');";

// getting the information from database and checking if available.
$check_name = mysqli_query($con, "SELECT name FROM users where name = '$name'");
if(mysqli_num_rows($check_name) != 0){
    echo'<script>alert("Name already exists");window.location.href="./Prodigy.html";</script>
';
}
else{
    // insert in database
    $rs = mysqli_query($con, $sql);
    if($rs){
        echo "<script>alert('***Registeration Successfull***Please Accept Instagram Follow Re
quest from _Prodigy_TIM***Cheer!***');
            window.location.href='./Prodigy.html';</script>";
    }
}
// exit()
?>
```