```
# importing all packages used with the program
import os

from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin, LoginManager, login_required, current_user, login_user,
logout_user
from flask import Flask, render_template, request, redirect, session, flash

from produce import *

# initialises the web application
app = Flask(__name__)
app.secret_key = 'xyz'

# establishes connection for database
basedir = os.path.abspath(os.path.dirname(__file__))
app.config['SQLALCHEMY_DATABASE_URI'] = \
    'sqlite:///' + os.path.join(basedir, 'database.db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# initialise the login packages
login = LoginManager()
login.login_view = 'login'
login.init_app(app)



# creates a user class used to create a database table
class User(UserMixin, db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(80), unique=True)
    username = db.Column(db.String(100), nullable=False)
    password_hash = db.Column(db.String(), nullable=False)

    diary = db.relationship('Diary', backref='user', lazy=True, passive_deletes=True)
    age = db.Column(db.Integer)
    weight = db.Column(db.Integer)
    height = db.Column(db.Integer)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)


# creates a diary class used to create a database table
class Diary(UserMixin, db.Model):
    entryID = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(50))
    description = db.Column(db.String(100))
    step_count = db.Column(db.Integer)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
                        nullable=False)


# migrate script would be copied in here


@login.user_loader
```

```
def load_user(id):
    return User.query.get(int(id))



# base page users instantly land on
@app.route('/')
def index():
    return render_template('index.html')



# about page
@app.route('/about')
def about():
    return render_template('about.html')



# login page
@app.route('/login', methods=['POST', 'GET'])
def login():
    """encase users are already logged in we don't want them accessing this page,
     so they will instantly be routed back to the home page"""
    if current_user.is_authenticated:
        return redirect('/')
    # when the submit button is pressed this script will run
    if request.method == 'POST':
        # takes the data from the page
        email = request.form['email']
        user = User.query.filter_by(email=email).first()
        # checks the password is correct using the method created earlier
        if user is not None and user.check_password(request.form['password']):
            login_user(user)
```

```
            return redirect('/')
        else:
            flash('Invalid password provided', 'error')

    return render_template('login.html')



# register page
@app.route('/register', methods=['POST', 'GET'])
def register():
    """encase users are already logged in we don't want them accessing this page,
     so they will instantly be routed back to the home page"""
    if current_user.is_authenticated:
        return redirect('/')
    # when the submit button is pressed this script will run
    if request.method == 'POST':
        email = request.form['email']
        username = request.form['username']
        password = request.form['password']
        if username == '':
            flash('username is blank', 'error')
            return redirect('/register')
        if password == '':
            flash('password is blank', 'error')
            return redirect('/register')


        val = password_checker(password)


        if not val:
            flash('password is needs to be longer than 6 characters, 1 upper case, 1 lower case and should
have at '
```

```
                'least one of the symbols $@#', 'error')
            return redirect('/register')

        """creates a new user object
        which is then used to submit a new user to the database"""
        user = User(email=email, username=username, age=None, weight=None, height=None)
        if (User.query.filter_by(email=email).first()) is not None:
            flash('Email already Present', 'error')
            return redirect('/register')
        user.set_password(password)
        db.session.add(user)
        db.session.commit()
        return redirect('/login')
    return render_template('register.html')



# logout page
@app.route('/logout')
def logout():
    # function from flask-login package
    logout_user()
    return redirect('/')



# diary page
@app.route('/diary', methods=['POST', 'GET'])
@login_required
def diary():
    user_id = current_user.id
    # takes all the entries from the database
    entries = Diary.query.all()
```

```
    return render_template('diary.html', entries=entries)



# new entry page
@app.route('/diary/new_entry', methods=['POST', 'GET'])
@login_required
def new_entry():
    # passes data from the page to the route
    if request.method == 'POST':
        title = request.form['title']
        description = request.form['description']
        steps = request.form['steps']
        if title == '' or description == '' or steps == '':
            flash('please make sure all fields are filled in', 'error')
            return redirect('/diary/new_entry')

        uid = current_user.id
        # creates a diary object which is then submitted to the database
        entry = Diary(title=title, description=description, step_count=steps, user_id=uid)

        db.session.add(entry)
        db.session.commit()
        return redirect('/diary')
    return render_template('new_entry.html')



# dashboard-location page
@app.route('/dashboard-loc', methods=['POST', 'GET'])
@login_required
def dashboard_loc():
    if request.method == 'POST':
```

```
        # creates the location as a session object so that it can easily be used by other functions
        session["location"] = request.form['locations']
        if session["location"] == '':
            flash('Invalid location provided', 'error')
        else:
            return redirect('/dashboard')
    return render_template('dashboard_loc.html')



# dashboard page
@app.route('/dashboard')
@login_required
def dashboard():
    # takes the location session
    location = session["location"]
    latitude, longitude = convert_lat(location)

    # API used to get the temperature
    temp_api = f'https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}'
\
            f'&hourly=temperature_2m,rain'

    # API used to get different air quality details
    air_api = f'https://air-quality-api.open-meteo.com/v1/air-
quality?latitude={latitude}&longitude={longitude}' \

f'&hourly=pm10,pm2_5,alder_pollen,birch_pollen,grass_pollen,mugwort_pollen,olive_pollen,ragwe
ed_pollen'

    # procedures to pass info from APIs to the template
    avg_temp = weather_api(temp_api, location, True)
    pollen_names, pollen_averages = air_quality_api(air_api, location)
```

```
    return render_template('dashboard.html', location=location, avg_temp=avg_temp,
                           pollen_avgs=zip(pollen_names, pollen_averages))


# donate page
@app.route('/donate')
def donate():
    return render_template('donate.html')


# allergens page
@app.route('/allergens')
def allergens():
    return render_template('allergens.html')


# pollen page
@app.route('/allergens/pollen')
def pollen():
    return render_template('pollen.html')


# dust page
@app.route('/allergens/dust')
def dust():
    return render_template('dust.html')


# pets page
@app.route('/allergens/pets')
```

```
def pets():
    return render_template('pets.html')


# smoke page
@app.route('/allergens/smoke')
def smoke():
    return render_template('smoke.html')


# asthma page
@app.route('/allergens/asthma')
def asthma():
    return render_template('asthma.html')


# weather page
@app.route('/weather')
def weather():
    return render_template('weather.html')


# script to delete a specified diary entry
@app.route('/delete_entry/int:<entryID>')
def delete_entry(entryID):
    # finds the specified entry
    entry_to_delete = Diary.query.get_or_404(entryID)
    db.session.delete(entry_to_delete)
    db.session.commit()
    return redirect('/diary')
```

```
# delete account page
@app.route('/delete_account')
def delete_account():
    return render_template('delete_account.html')


@app.route('/del_account/int:<id>')
def del_account(id):
    # finds the specified account
    account_to_delete = User.query.get_or_404(id)
    db.session.delete(account_to_delete)
    db.session.commit()
    return redirect('/')


# error 404 page
def page_not_found(e):
    return render_template('404.html'), 404


# assigns the page to be used when a 404 error occurs
app.register_error_handler(404, page_not_found)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```