1 app.py

Listing 1: app.py

```python
from flask import Flask, render_template
from api.common.register_routes import CommonBPRegister
from api.web.register_routes import WebBPRegister
from flask_jwt_extended import JWTManager
from flask_jwt_extended.exceptions import
    NoAuthorizationError, InvalidHeaderError, JWTDecodeError
from datetime import timedelta
from database.db import init_db

app = Flask(__name__)

# Register JWT
app.config['JWT_SECRET_KEY'] = 's3cr3t_k3y_f0r_jwt'
app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(days=30)
jwt = JWTManager(app)

# Handle JWT exceptions
@app.errorhandler(NoAuthorizationError)
def handle_auth_error(e):
    return jsonify({"msg": "Missing authorization header"}),
        401

@app.errorhandler(InvalidHeaderError)
def handle_invalid_header_error(e):
    return jsonify({"msg": "Invalid authorization header"}),
        401

@app.errorhandler(JWTDecodeError)
def handle_expired_signature_error(e):
    return jsonify({"msg": "Token has expired"}), 401

# Register blueprints
CommonBPRegister(app)
WebBPRegister(app)

# Init db
app.config['SQLALCHEMY_DATABASE_URI'] =
    f'postgresql://root:root@postgres:5432/postgres'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
init_db(app)

# DONT forget to remove
@app.route('/')
def welcome():
    return render_template('welcome.html')

```

```python
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

2 database/db.py

Listing 2: database/db.py

```python
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

def init_db(app):
    db.init_app(app)

class DBStatus:
    OK = "OK"
    ERROR = "ERROR"

"""Base model class providing common database operations"""
class BaseModel:

    # Create a new record in the database
    def create(self):
        try:
            db.session.add(self)
            db.session.commit()
            return DBStatus.OK
        except Exception as e:
            db.session.rollback()
            print(f"Error creating record: {e}")
            return DBStatus.ERROR

    # Find a record by its ID
    @classmethod
    def find_by_id(cls, id):
        try:
            return cls.query.filter_by(id=id).first(),
                DBStatus.OK
        except Exception as e:
            print(f"Error finding record by ID: {e}")
            return None, DBStatus.ERROR

    # Update an existing record
    def update(self, **kwargs):
        try:
            for key, value in kwargs.items():
                setattr(self, key, value)
            db.session.commit()
            return DBStatus.OK
        except Exception as e:
```

```
            db.session.rollback()
            print(f"Error updating record: {e}")
            return DBStatus.ERROR

    # Delete an existing record
    def delete(self):
        try:
            db.session.delete(self)
            db.session.commit()
            return DBStatus.OK
        except Exception as e:
            db.session.rollback()
            print(f"Error deleting record: {e}")
            return DBStatus.ERROR

    # Return all records of this model
    @classmethod
    def return_all(cls):
        try:
            return cls.query.all(), DBStatus.OK
        except Exception as e:
            print(f"Error retrieving all records: {e}")
            return None, DBStatus.ERROR

"""Model representing users in the database."""
class users(db.Model, BaseModel):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    full_name = db.Column(db.String(100))
    gender = db.Column(db.String(10))
    age = db.Column(db.Integer)
    weight = db.Column(db.Float)
    date_of_birth = db.Column(db.Date)

"""Model representing records in the database."""
class records(db.Model, BaseModel):
    __tablename__ = 'records'

    id = db.Column(db.Integer, primary_key=True)
    reading_date = db.Column(db.Date, nullable=False)
    time_interval = db.Column(db.Interval, nullable=False)
    readings_data = db.Column(db.JSON)

"""Model representing authentication credentials in the
    database."""
class auth(db.Model, BaseModel):
    __tablename__ = 'auth'

    id = db.Column(db.Integer, primary_key=True)
```

```
92      login = db.Column(db.String(120), unique=True,
            nullable=False)
93      password = db.Column(db.String(120), nullable=False)
94      email = db.Column(db.String(120), unique=True,
            nullable=False)
95      user_id = db.Column(db.Integer, db.ForeignKey('users.id'),
            nullable=False)
96
97      user = db.relationship('users', backref=db.backref('auth',
            lazy=True))
98
99      # Find an authentication record by login
100     @classmethod
101     def find_by_login(cls, login):
102         try:
103             return cls.query.filter_by(login=login).first(),
                    DBStatus.OK
104         except Exception as e:
105             print(f"Error finding record by login: {e}")
106             return None, DBStatus.ERROR
107
108     # Find an authentication record by email
109     @classmethod
110     def find_by_email(cls, email):
111         try:
112             return cls.query.filter_by(email=email).first(),
                    DBStatus.OK
113         except Exception as e:
114             print(f"Error finding record by email: {e}")
115             return None, DBStatus.ERROR
```

3 test/basic_post.py

Listing 3: test/basic_post.py

```
1  import requests
2
3  url = 'http://127.0.0.1:5000/api/register'
4
5  data = {
6      'login': 'va',
7      'password': 'bd',
8      'email': "bch09@rambler.com"
9  }
10
11 response = requests.post(url, json=data)
12
13 if response.status_code ==  200:
14     print('POST request was successful.')
15     print('Content\n')
```

```
    print(response.json())
else:
    print(f'POST request failed with status code
        {response.status_code}.')

print(response.text)
```

4 api/web/register$_{r}$outes.py

Listing 4: api/web/register_routes.py

```
from api.web.users.routes import web_users_bp

common_prefix='/api/web'

class WebBPRegister:
    def __init__(self, app):
        self.app = app
        self.register_blueprints()

    def register_blueprints(self):
        self.app.register_blueprint(web_users_bp,
            url_prefix=common_prefix)
```

5 api/web/users/routes.py

Listing 5: api/web/users/routes.py

```
from flask import Blueprint

web_users_bp = Blueprint('users_info', __name__)
```

6 api/common/register$_{r}$outes.py

Listing 6: api/common/register_routes.py

```
from api.common.auth.routes import common_auth_bp

common_prefix='/api'

class CommonBPRegister:
    def __init__(self, app):
        self.app = app
        self.register_blueprints()

    def register_blueprints(self):
        self.app.register_blueprint(common_auth_bp,
            url_prefix=common_prefix)
```

7 api/common/auth/routes.py

Listing 7: api/common/auth/routes.py

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import create_access_token,
    create_refresh_token, jwt_required
from database.db import auth, DBStatus, users

common_auth_bp = Blueprint('auth', __name__)

@common_auth_bp.route('/login', methods=['POST'])
def login_post():
    if not request.is_json:
        return jsonify({"msg": "Missing JSON in request"}), 400

    login = request.json.get('login')
    password = request.json.get('password')

    if not login or not password:
        return jsonify({"msg": "Missing login or password"}),
            400

    auth_record, status = auth.find_by_login(login)

    if status == DBStatus.ERROR or auth_record is None:
        return jsonify({"msg": "Invalid login credentials"}),
            401

    if auth_record.password != password:
        return jsonify({"msg": "Invalid login credentials"}),
            401

    access_token = create_access_token(identity=login)
    refresh_token = create_refresh_token(identity=login)
    return jsonify({"msg": "Login successful", "access_token":
        access_token, "refresh_token": refresh_token}), 200

@common_auth_bp.route('/refresh', methods=['POST'])
@jwt_required(refresh=True)
def refresh():
    current_user = get_jwt_identity()
    access_token = create_access_token(identity=current_user)
    return jsonify({"access_token": access_token}), 200

@common_auth_bp.route('/register', methods=['POST'])
def register_post():
    if not request.is_json:
        return jsonify({"msg": "Missing JSON in request"}), 400

```

```
42      login = request.json.get('login')
43      password = request.json.get('password')
44      email = request.json.get('email')
45
46      if not login or not password or not email:
47          return jsonify({"msg": "Missing required fields"}), 400
48
49      existing_auth_record, status = auth.find_by_login(login)
50
51      if status == DBStatus.OK and existing_auth_record:
52          return jsonify({"msg": "Login already exists"}), 400
53
54      existing_email_record, status = auth.find_by_email(email)
55
56      if status == DBStatus.OK and existing_email_record:
57          return jsonify({"msg": "Email already registered"}),
                400
58
59      new_user_record = users()
60      status_user = new_user_record.create()
61
62      if status_user != DBStatus.OK:
63          return jsonify({"msg": "Failed to register user"}), 500
64
65      new_auth_record = auth(login=login, password=password,
            email=email, user_id=new_user_record.id)
66      status = new_auth_record.create()
67
68      if status == DBStatus.OK:
69          return jsonify({"msg": "Registration successful"}), 200
70      else:
71          return jsonify({"msg": "Failed to register user"}), 500
72
73  """ REMOVE """
74  @common_auth_bp.route('/auths', methods=['GET'])
75  def get_all_auths():
76      from database.db import auth
77      auths, status = auth.return_all()
78
79      if status == DBStatus.OK:
80          user_data = [{"id": auth.id, "login": auth.login,
                "email": auth.email, "id_user": auth.user_id} for
                auth in auths]
81          return jsonify({"auths": user_data}), 200
82      else:
83          return jsonify({"msg": "Failed to fetch users"}), 500
84
85  """ REMOVE """
86  @common_auth_bp.route('/users', methods=['GET'])
87  def get_all_users():
```

```
88      from database.db import users
89      users, status = users.return_all()
90
91      if status == DBStatus.OK:
92          user_data = [{"id": users.id} for users in users]
93          return jsonify({"users": user_data}), 200
94      else:
95          return jsonify({"msg": "Failed to fetch users"}), 500
```