```
"""Restful Server-Side API"""

from flask import request, jsonify
from flask_bcrypt import Bcrypt
bcrypt = Bcrypt()
from helpers import db_query_to_dict, db_user_to_dict, dict_query_to_db, db_story_to_dict, dict_story_to_db
from app import app

# sqlalchemy imports
from sqlalchemy import exc
from psycopg2.errors import UniqueViolation
from models import db, User, Story, Query


"""USERS"""

@app.route('/users', methods=["GET"])
def get_all_users():
    """Gets all users"""
    users = User.query.all()
    dict_list = [db_user_to_dict(user) for user in users]
    return jsonify(users = dict_list)

@app.route('/users/delete', methods=["DELETE"])
def delete_all_users():
    """Deletes all users"""
    try:
        User.query.delete()
        db.session.commit()
    except exc.SQLAlchemyError as e:
```

```
        response = {"Unable to delete all users": f"{e.origin}"}
        return response
    return {"message":"Success! All users were deleted."}

@app.route('/users/<int:user_id>', methods=["GET"])
def get_user(user_id):
    """Gets a user by user_id"""
    user = User.query.get_or_404(user_id)
    dict = db_user_to_dict(user)
    return jsonify(user = dict)

@app.route('/users/<int:user_id>', methods=["DELETE"])
def delete_user(user_id):
    """Deletes a user by user_id"""
    try:
        User.query.filter_by(id=user_id).delete()
        db.session.commit()
        return {"message":"Success! User was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete user": f"{e.origin}"}
        return response

@app.route('/users/new', methods=["POST"])
def new_user():
        """Creates a new user"""
        data = request.json['user']
        new_user = User.register(
            data['username'], data['password'], data['email'], data['first_name'], data['last_name'])
        try:
            db.session.add(new_user)
            db.session.commit()
```

```
            response = {"message": f"User {data['username']} added successfully."}
        except exc.SQLAlchemyError as e:
            if isinstance(e.orig, UniqueViolation):
                response = {"message": f"{e.origin}"}
            else:
                response = {"message": "Sorry, something went wrong. Unable to add new user to database."}
        return response


@app.route('/users/<int:user_id>/edit', methods=["PUT"])
def edit_user(user_id):
    """Edits a user's information by user_id"""
    user = User.query.get(user_id)
    data = request.json['user']
    user.username = data['username']
    hashed = bcrypt.generate_password_hash(data['password'])
    hashed_utf8 = hashed.decode("utf8")
    user.password = hashed_utf8
    user.email = data['email']
    user.first_name = data['first_name']
    user.last_name = data['last_name']
    try:
        db.session.add(user)
        db.session.commit()
        dict = db_user_to_dict(user)
        return jsonify(updated_user = dict)
    except exc.SQLAlchemyError as e:
        if isinstance(e.orig, UniqueViolation):
            response = {"message": f"{e.origin}"}
        else:
            response = {"message": "Sorry, something went wrong. Unable to update user."}
```

```
            return response


"""USER QUERIES"""


@app.route('/users/<int:user_id>/queries/new', methods=["POST"])
def new_query(user_id):
    """Creates a new query and associates it with user_id"""
    data = request.json['query']
    try:
        query = dict_query_to_db(user_id, data)
        db.session.add(query)
        db.session.commit()

    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response

    dict = db_query_to_dict(query)
    return jsonify(new_query = dict)


@app.route('/users/<int:user_id>/queries/', methods=["GET"])
def get_all_queries_by_user(user_id):
    """Gets all queries matching user_id"""
    user = User.query.get_or_404(user_id)
    queries = user.queries
    dict_list = [db_query_to_dict(query) for query in queries]
    return jsonify(queries = dict_list)


@app.route('/users/<int:user_id>/queries/<int:query_id>/edit', methods=["PUT"])
def edit_query(user_id, query_id):
    """Edits a User Query by User and Query ids"""
```

```
    data = request.json['query']
    query = Query.query.get(query_id)
    try:
        if user_id != query.user_id:
            raise ValueError("The user id is not associated with the query id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the query id."}
        return response
    try:
        query = dict_query_to_db(user_id, data)
        db.session.add(query)
        db.session.commit()
    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response
    dict = db_query_to_dict(query)
    return jsonify(updated_query = dict)


@app.route('/users/<int:user_id>/queries/<int:query_id>/delete', methods=["DELETE"])
def delete_query(user_id, query_id):
    """Deletes a User Query by User and Query ids"""
    query = Query.query.get(query_id)
    try:
        if user_id != query.user_id:
            raise ValueError("The user id is not associated with the query id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the query id."}
        return response
    try:
        Query.query.filter_by(id=query_id).delete()
        db.session.commit()
```

```
        return {"message":"Success! Query was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete query": f"{e.origin}"}
        return response

"""USER STORIES"""

@app.route('/users/<int:user_id>/stories/new', methods=["POST"])
def new_story(user_id):
    """Creates a new story and associates it with user_id"""
    data = request.json['story']
    try:
        story = dict_story_to_db(user_id, data)
        db.session.add(story)
        db.session.commit()

    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response

    dict = db_query_to_dict(story)
    return jsonify(new_story = dict)

@app.route('/users/<int:user_id>/stories/', methods=["GET"])
def get_all_stories_by_user(user_id):
    """Gets all queries matching user_id"""
    user = User.query.get_or_404(user_id)
    stories = user.stories
    dict_list = [db_story_to_dict(story) for story in stories]
    return jsonify(stories = dict_list)
```

```
@app.route('/users/<int:user_id>/stories/<int:story_id>/edit', methods=["PUT"])
def edit_story(user_id, story_id):
    """Edits a User Story by User and Query ids"""
    data = request.json['query']
    story = Story.query.get_or_404(story_id)
    try:
        if user_id != story.user_id:
            raise ValueError("The user id is not associated with the story id.")
    except ValueError:
        response = {"Value Error": "The user id is not associated with the story id."}
        return response

    try:
        story = dict_story_to_db(user_id, data)
        db.session.add(story)
        db.session.commit()
        dict = db_query_to_dict(story)
        return jsonify(updated_story = dict)

    except exc.SQLAlchemyError as e:
        response = {"message": f"{e.origin}"}
        return response

@app.route('/users/<int:user_id>/queries/<int:story_id>/delete', methods=["DELETE"])
def delete_story(user_id, story_id):
    """Deletes a User Story by User and Query ids"""
    story = Query.query.get_or_404(story_id)
    try:
        if user_id != story.user_id:
            raise ValueError("The user id is not associated with the story id.")
    except ValueError:
```

```
        response = {"Value Error": "The user id is not associated with the story id."}
        return response

    try:
        Query.query.filter_by(id=story_id).delete()
        db.session.commit()
        return {"message":"Success! Story was deleted."}
    except exc.SQLAlchemyError as e:
        response = {"Unable to delete story": f"{e.origin}"}
        return response

"""QUERIES"""

@app.route('/queries', methods=["GET"])
def get_all_queries():
    """Gets all queries"""
    queries = Query.query.all()
    dict_list = [db_query_to_dict(query) for query in queries]
    return jsonify(queries = dict_list)

@app.route('/queries/<int:query_id>', methods=["GET"])
def get_query(query_id):
    """Gets a query by query_id"""
    query = Query.query.get_or_404(query_id)
    dict = db_query_to_dict(query)
    return jsonify(query= dict)


"""STORIES"""

@app.route('/stories', methods=["GET"])
```

```
def get_all_stories():
    """Gets all stories"""
    stories = Story.query.all()
    dict_list = [db_story_to_dict(story) for story in stories]
    return jsonify(stories = dict_list)


@app.route('/stories/<story_id>', methods=["GET"]) #story ids are non-numerical as they are inherited from sessions randomly generated uuid() ids
def get_story(story_id):
    """Gets story by story id"""
    story = Story.query.get_or_404(story_id)
    dict = db_story_to_dict(story)
    return jsonify(story = dict)




#sample query:
#{"query": {"name":"test222", "source":"fox news", "quantity":10, "date_from":"", "date_to":"", "language":"en", "sort_by":"subjectivity", "sa":"", "type":"Detailed Search"}}
#sample user:
# {"user": {"username":"coolguy41", "password":"hmmm", "email":"eee@mail.com", "first_name":"first", "last_name":"last"}}
```