API Design

Useful links:

FA24 Hack Challenge Description

November 25, 2024

Notes Model:

1. Id
2. Title
3. Text
4. Date
5. Public, private
6. Users who have access

Users Model:

1. Id
2. Name
3. Age
4. Notes

Users to Notes (one to many)

Routes

1. Add
2. Delete
3. Edit
4. Get all notes
5. Get all users
6. View access, edit access

Detailed API descriptions can be found in our repository.
https://github.com/nikobross/NotesApp-AppDev-Hack-Challenge/blob/main/api_spec.ipynb

Below are the database and server files for easy reference.

Database file

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(db.Model):
   __tablename__ = "users"
   id = db.Column(db.Integer, primary_key=True)
   username = db.Column(db.String, nullable=False)
   password = db.Column(db.String, nullable=False)
   notes = db.relationship('Note', backref='user', lazy=True)

   def serialize(self):
       return {
           "id": self.id,
           "username": self.username,
           "password": self.password,
           "notes": [note.serialize() for note in self.notes]
       }

class Note(db.Model):
   __tablename__ = "notes"
   id = db.Column(db.Integer, primary_key=True)
   title = db.Column(db.String, nullable=False)
   content = db.Column(db.String, nullable=False)
   date = db.Column(db.String, nullable=False)
   user_id = db.Column(db.Integer, db.ForeignKey("users.id"), nullable=False)

   def serialize(self):
       return {
           "id": self.id,
           "title": self.title,
           "content": self.content,
           "date": self.date,
           "user_id": self.user_id
       }
```

Server file (stubs show necessary routes)

```
from db import db, User, Note
from flask import Flask, request
import json

app = Flask(__name__)
db_filename = "notes.db"

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///%s" % db_filename
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
app.config["SQLALCHEMY_ECHO"] = False

db.init_app(app)
with app.app_context():
   db.create_all()


# generalized response formats
def success_response(data, code=200):
   return json.dumps(data), code


def failure_response(message, code=404):
   return json.dumps({"error": message}), code


# ----------- User routes -------------


@app.route("/api/users", methods=["GET"])
def get_all_users():
   pass


@app.route("/api/users/", methods=["POST"])
def create_user():
   pass


@app.route("/api/users/<int:user_id>/")
def get_specific_user(user_id):
   pass
```

```python
@app.route("/api/users/<int:user_id>/", methods=["DELETE"])
def delete_user(user_id):
   pass



# ---------- Notes Helpers ------------

def create_note_helper(title, content, user_id):
   note = Note(title=title, content=content, user_id=user_id)

   db.session.add(note)
   db.session.commit()

   return note.serialize()

def update_note_helper(note_id, title, content):
   note = Note.query.filter_by(id=note_id).first()

   if note is None:
       return failure_response("note not found, check note id")

   note.title = title
   note.content = content

   db.session.commit()

   return note.serialize()

def delete_note_helper(note_id):
   note = Note.query.filter_by(id=note_id).first()

   if note is None:
       return failure_response("note not found, check note id")

   res = note.serialize()

   db.session.delete(note)
   db.session.commit()
   return success_response(res)

# ----------- Notes routes ------------

# get all notes
@app.route("/api/notes/", methods=["GET"])
def get_all_notes():
   notes = Note.query.all()

   res = {"notes": [note.serialize() for note in notes]}

   return success_response(res)


# get specific note
@app.route("/api/notes/<int:note_id>/", methods=["GET"])
def get_note(note_id):
   note = Note.query.filter_by(id=note_id).first()

   if note is None:
       return failure_response("note not found, check id")

   res = note.serialize()

   return success_response(res)


# create note, take in title, content, and user_id
# should we allow empty notes? I am right now
@app.route("/api/notes/", methods=["POST"])
def create_note():

   body = json.loads(request.data)

   title = body.get("title")
   content = body.get("content")
   user_id = body.get("user_id")

   if user_id is None:
       return failure_response("user_id is required")

   user = User.query.filter_by(id=user_id).first()
```

```
    if user is None:
        return failure_response("user not found, check user_id")

    res = create_note_helper(title, content, user_id)

    return success_response(res, 201)


# update note, take in title and content
# (these will be updated no matter what so we need to pass in both even if only
one has been changed)
# this can easily be changed if that makes it easier
@app.route("/api/update-note/<int:note_id>/", methods=["POST"])
def update_note(note_id):

    body = json.loads(request.data)

    title = body.get("title")
    content = body.get("content")

    note = Note.query.filter_by(id=note_id).first()

    if note is None:
        return failure_response("note not found, check note id")

    res = update_note_helper(note_id, title, content)

    return success_response(res)


# delete note
@app.route("/api/notes/<int:note_id>/", methods=["DELETE"])
def delete_note(note_id):

    res = delete_note_helper(note_id)

    return res

# get all notes for a user
@app.route("/api/user/<int:user_id>/notes/", methods=["GET"])
def get_all_notes_for_user(user_id):
```

```python
   user = User.query.filter_by(id=user_id).first()

   if user is None:
       return failure_response("user not found, check user_id")

   notes = Note.query.filter_by(user_id=user_id).all()

   res = {"notes": [note.serialize() for note in notes]}

   return success_response(res)


if __name__ == "__main__":
   app.run(host="0.0.0.0", port=8000, debug=True)
```