GAME STORE/LIBRARY REST API ENDPOINT

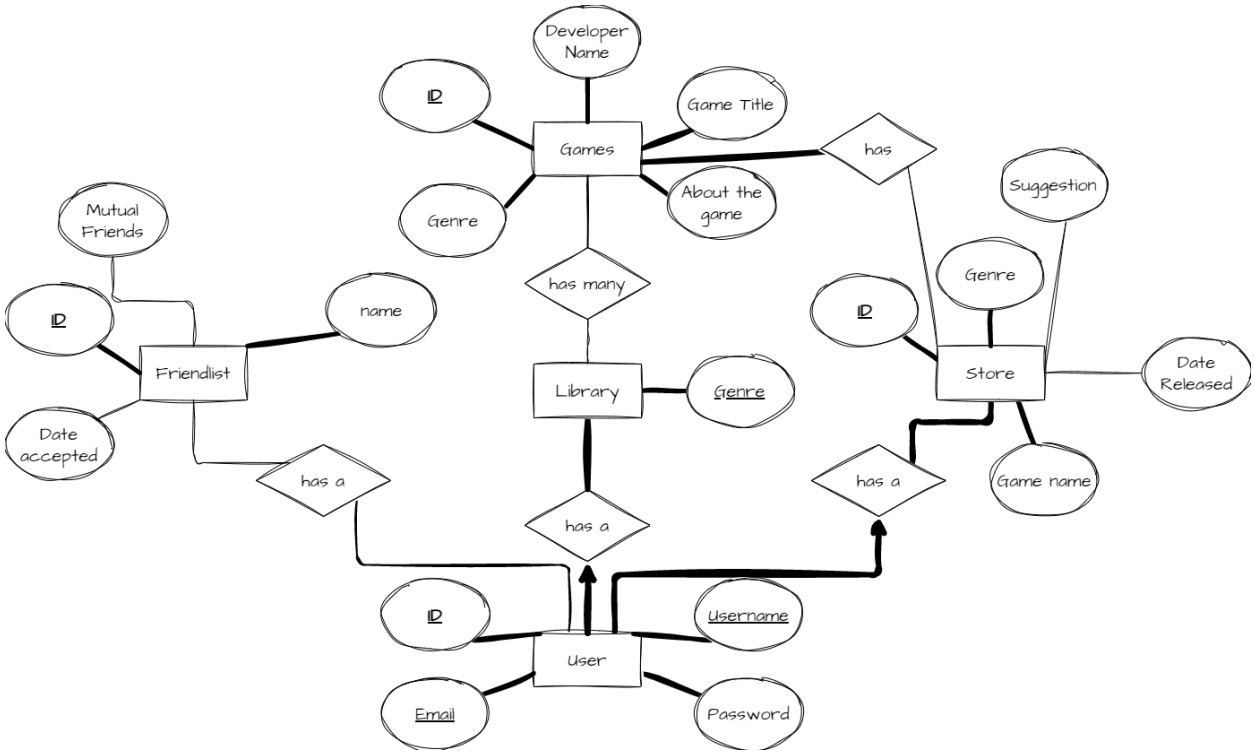
- Nicholas Patrick Suyat

This Portfolio Project of mine is about making a REST API ENDPOINT for 'Steam clone Game Launcher."

Notes: I am only showing the "users" entity of this document. There are 4 total entities that I used (users, library, games, store)

Things that I used in this Project: ER-DIAGRAM, Database (PostgreSQL Pgadmin), VScode, Python, psycopg2, alembic, Insomnia, and Flask.

ER-DIAGRAM



DATABASE MIGRATION- ALEMBIC

Step: In this Migration- Alembic. I used Raw SQL to migrate the data. It transferred the file from Vscode python file to the Pgadmin.

```python
def upgrade():
    op.execute(
        """
        CREATE TABLE users(
            id SERIAL PRIMARY KEY,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL,
            email TEXT NOT NULL UNIQUE,
            library_id INT,
            store_id INT

        );
        """
```

```python
def downgrade():
    op.execute(
        """
        DROP TABLE users;
        """
    )
```

The op.execute is how you execute the migration Raw SQL.

API ENDPOINT- 'GET' INDEX

Step: In this blocks of code, I am implementing a 'GET' method.

```python
bp = Blueprint("users", __name__, url_prefix="/users")


@bp.route('', methods=['GET'])
def index():
    cur = conn.cursor()
    cur.execute("SELECT * FROM users;")
    rows = cur.fetchall()
    result = []
    for row in rows:
        info = {'id': row[0], 'username': row[1], 'email': row[2],
                'library_id': row[3], 'store_id': row[4]}
        result.append(info)
    return jsonify(result)
```

The @bp.route is a code that tells the Insomnia "This is the endpoint for the GET method."

I also created a function called def index() that execute the 'GET' method by fetching all "cur.fetchall" the data from "users" and iterating it using the for loop method. After that, I used the return jsonify so it will transfer as a JSON file

API ENDPOINT- 'GET' get_id

Step: In this code, I implemented a get id method.

```python
@bp.route('/<int:id>', methods=['GET'])
def get_id(id):
    cur = conn.cursor()
    cur.execute("SELECT * FROM users WHERE id = %s;", (id,))
    rows = cur.fetchall()
    result = []
    for row in rows:
        info = {'id': row[0], 'username': row[1], 'email': row[2],
                'library_id': row[3], 'store_id': row[4]}
        result.append(info)
    return jsonify(result)
```

Whenever a client requests a specific id, it returns the information of that specific id. As you can see, the @bp.route('/<int:id>') is different than the index function. It is because a client can put any id they want on the URL.

API ENDPOINT- 'POST' create

```python
@bp.route('', methods=['POST'])
def create():
    data = request.get_json()

    if data is not None:
        if 'username' in data and 'password' in data and 'email' in data and library_id in data and store_id in data:
            username = data['username']
            password = data['password']
            email = data['email']
            library_id = data['library_id']
            store_id = data['store_id']

            try:
                cur = conn.cursor()
                cur.execute(
                    "INSERT INTO users (username, password, email, library_id, store_id) VALUES (%s, %s, %s, %s, %s)", (username, password, email, library_id, store_id))
                conn.commit()
                cur.close()

                return jsonify(success=True)
            except psycopg2.Error as e:
                conn.rollback()
                return abort(500)
        else:
            return abort(400)
    else:
        return abort(400)
```

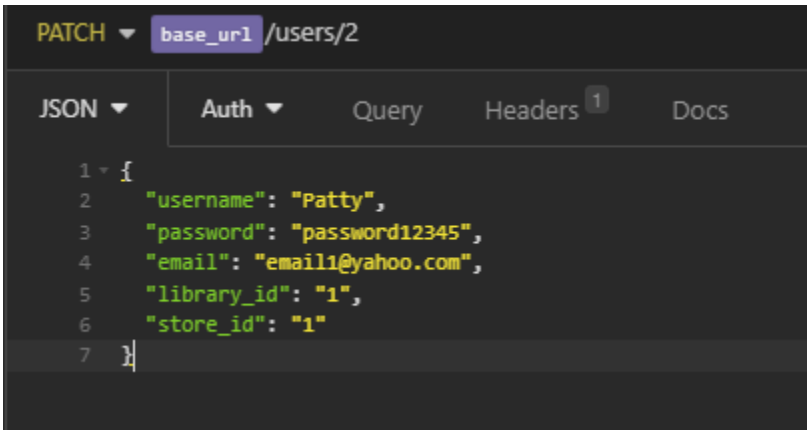
Step: In this code, I implemented a 'POST' method that creates a new data/value to the existing database. I declare a variable called data = request.get_json(). This variable reads what ever you put in the Insomnia JSON. I also made sure that what ever you put in the get_Json is not in the database already. If not, it will execute the INSERT function

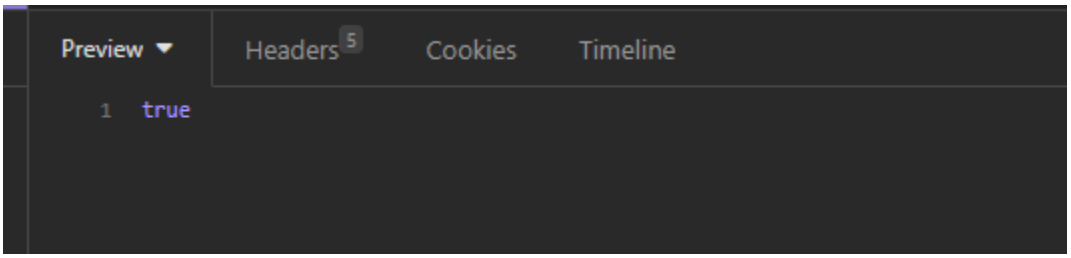
API ENDPOINT- ‘PATCH’ ‘PUT’ update

```python
@bp.route('/<int:id>', methods=['PATCH', 'PUT'])
def update(id):
    data = request.get_json()

    if 'username' in data and 'password' in data and 'email' in data and library_id in data and store_id in data:
        username = data['username']
        password = data['password']
        email = data['email']
        library_id = data['library_id']
        store_id = data['store_id']

        try:
            cur = conn.cursor()
            cur.execute(
                "UPDATE users SET username = %s, password = %s, email = %s, library_id = %s, store_id =%s WHERE id = %s ", (username, password, email, library_id, store_id, id))
            conn.commit()
            cur.close()

            return jsonify(True)
        except psycopg2.Error as er:
            conn.rollback()
            return abort(400)
    else:
        return abort(400)
```

Step: In this code, I implemented a ‘PATCH’ method that updates the existing values in the database. The code is the same as the create except the cur.execute() which updates the value to whatever you put in the JSON requests. As you can see, the @bp.route(‘/<int:id>’) has the id in it. It is because have to put which id data you want to change.

Insomnia: In Insomnia, the JSON requests looks like this:





API ENDPOINT- ‘DELETE’ delete

```python
@bp.route('/<int:id>', methods=['DELETE'])
def delete(id):
    try:
        cur = conn.cursor()
        cur.execute("DELETE FROM users WHERE id = %s", (id,))
        conn.commit()
        return jsonify(True)

    except:
        conn.rollback()
        return abort(400)
```

Step: In this code, I implemented a ‘DELETE’ method that deletes a specific that is requested in the URL. In the cur.execute() I input the raw SQL DELETE function to delete any data in it.

CONCLUSION

In this Document, I covered everything about making an API ENDPOINT and how to test it using Insomnia.

After working on this Portfolio Project, I learned a lot about using ER-DIAGRAM, Database (PostgreSQL Pgadmin), VScode, Python, psycopg2, alembic, Insomnia, and Flask. It teaches me how to migrate the data from Vscode to Pgadmin using Raw SQL. It also teaches me how to connect my code to the server or to the database.