Queen Mary
University of London

Group 14
Michael Capaldi - 210641428
Daniel Crake - 210744246
Philip Wurzner - 210930632

Queen Mary University of London

Mini project overview

1. Create a dynamically generated REST API.
2. Makes use of an external Owen Wilson WOW! API.
3. API set of services include ”GET”, ”POST”, ”PUT” and ”DELETE”.
4. The application uses cloud database (GCP) to store and access information.
5. Makes use of two unique databases.
6. Implementing a hash-based authentication.
7. Allowing users to create accounts.

Queen Mary
University of London

Web App Wow!

Web App Wow!

The app has been setup to include two databases. The first facilitates the login and contains username and password information. This has been setup to ensure the username is unique within this database. Once an account has been setup the username and password information is added to the user database. If username and password requirements are met, you will be able to continue with the login and be directed to the user_home.

Each user_home is unique and associated with the username used to login. The second database is used for storing the ratings made by that user. It contains 6 columns, these include: list_id, username, movie_name, director, year and review_score. The users unique movie review list is used by querying the second database with the username.

RESTful Services

1. Web App Wow allows users to create (“POST”) and log in (“POST”) to their account.
2. A button will randomly select (”GET”) a movie from an external database.
3. User can rate the movie (“POST”).
4. Each movie rating is stored on the users account to keep record.
5. Editing a rating (“PUT”) is another option for the user.
6. Finally deleting a review (“DELETE”).

Queen Mary University of London

Database

```
class User(db.Model):

  u_id = db.Column(db.Integer,
                   primary_key=True)

  username = db.Column(db.String(64),
                       nullable=False)

  password = db.Column(db.String(64),
                       nullable=True)
  @classmethod
  def find_user(cls, username):
    return cls.query.filter_by(username = username).first()

  def generate_pass_hash(self, password):
    self.password = generate_password_hash(password)

  def check_pass(self, _password):
    return check_password_hash(self.password, _password)
```

User Database Setup

- Here the database is created with three columns (u_id, username, password).
- Three functions are defined for the user class.
  - 1. find_user searches the database to see if the user exists.
  - 2. generate_pass_hash which hashes the user's password.
  - 3 .check_pass which verifies the user's hashed password

Queen Mary University of London

Database

```python
class Movie_review(db.Model):

    l_id = db.Column(db.Integer,
                     primary_key=True)

    username = db.Column(db.String(64))

    movie = db.Column(db.String(64))

    director = db.Column(db.String(64))

    year = db.Column(db.Integer)

    review = db.Column(db.Float(10))

    @classmethod
    def find_review(cls, username, movie):
        return cls.query.filter_by(username = username, movie = movie).first()

    @classmethod
    def show_list(cls, username):
        return cls.query.filter_by(username = username)
```

Movie Review Database Setup

This defines the movie_review database . It consists of 6 unique columns. Two functions are also defined.

- find_review - this checks the movie review database to see if a specific user has reviewed a particular movie
- show_list - queries the movie database and returns a full list of every movie a user has reviewed alongside the movie details

Queen Mary University of London

User Accounts

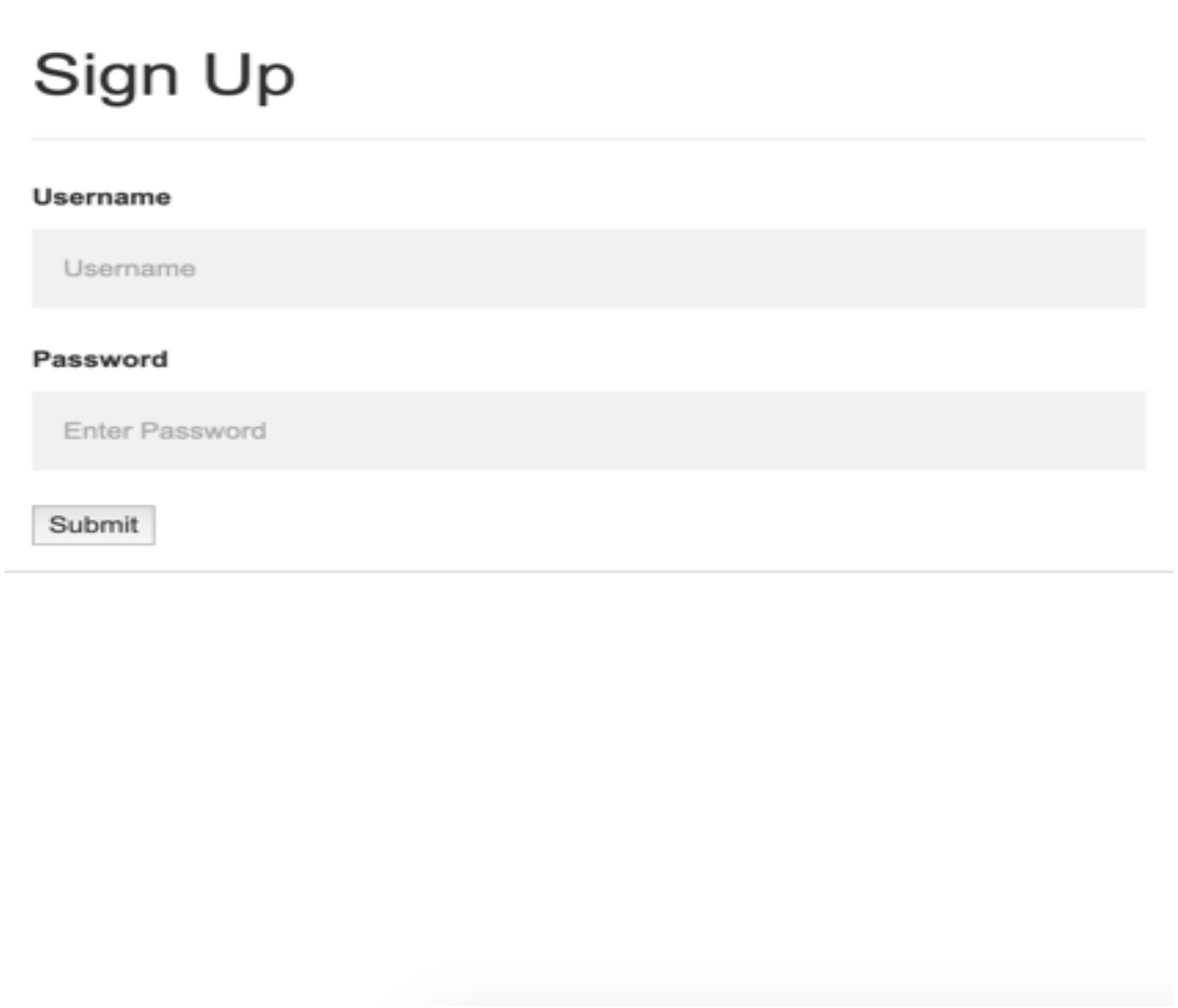
```python
@app.route("/signup/", methods=["GET","POST"])
def signup():
    if request.method == "POST":
        username = request.form['username'].strip()
        password = request.form['password'].strip()

        if User.find_user(username = username) or not (username and password):
            flash("Username already Exists")
        else:
          new_user = User(username = username)
          new_user.generate_pass_hash(password = password)
          db.session.add(new_user)
          db.session.commit()
          flash("User added")
          return redirect(url_for("login"))

    return render_template("signup.html")
```

Signup POST operation

- One the signup page a POST operation is conducted to collect the new_users username and password.
- The signup function checks whether the username exists in the user database and if it is not present it hashes the password and appends the user to the database

Queen Mary University of London

User Accounts




```
<form id="signup" action="{{ url_for('signup')}}" method="post" style="border:1px solid #ccc"
  <div class="container">
    <h1>Sign Up</h1>
    <hr>

    <label for="username"><b>Username</b></label>
    <input type="text" placeholder="Username" name="username" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="password" required>

    <div class="clearfix">
      <input type="submit" id="submit" text="Signup" />
    </div>
  </div>
</form>
```

Queen Mary University of London

Method="POST"

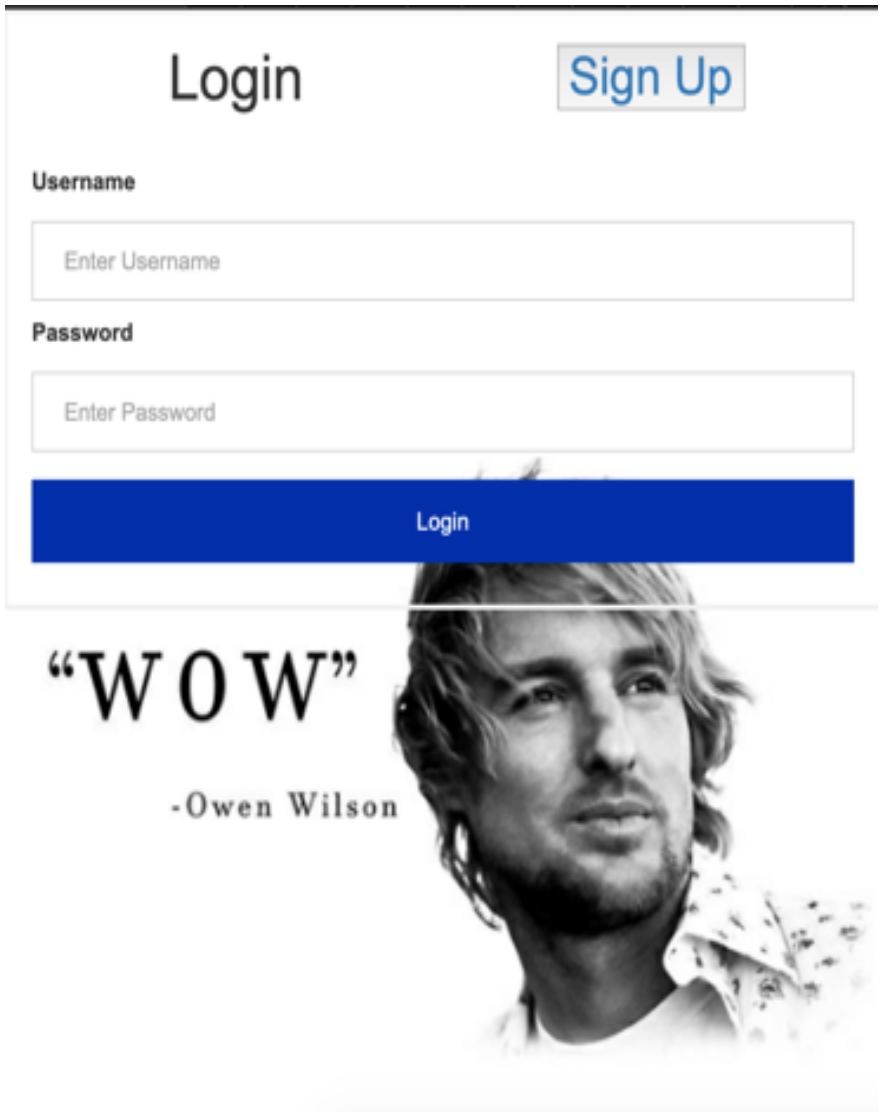
```
@app.route("/", methods=["POST"])
@app.route("/login/", methods=["GET", "POST"])
def login():

    if request.method == "POST":
        username = request.form['username'].strip()
        password = request.form['password'].strip()

        login_user = User.find_user(username = username)

        if login_user and login_user.check_pass(_password = password):
            session[username] = True
            return redirect(url_for("user_home", username = username))
        else:
            flash("Invalid login")

    return render_template("login_form.html")
```

User log in POST Example

- On the login page an operation similar to the signup function is preformed.
- If a POST method is detected from the login page, the username and password are stripped from the request form.
- The user is associated with an entry in the user database, and if the user exists and the password is verified, the user is allowed to login

Queen Mary University of London

Method="POST"




```
<form id="login" action="{{ url_for('login')}}" method="post">
    <div class="container">
      <label for="uname"><b>Username</b></label>
      <input type="text" id="username" placeholder="Enter Username" name="username" />

      <label for="psw"><b>Password</b></label>
      <input type="password" id="password" placeholder="Enter Password" name="password" />
      <button type="submit" value="Log In">Login</a></button>

    </div>
</form>
```

Queen Mary University of London

Method="GET"

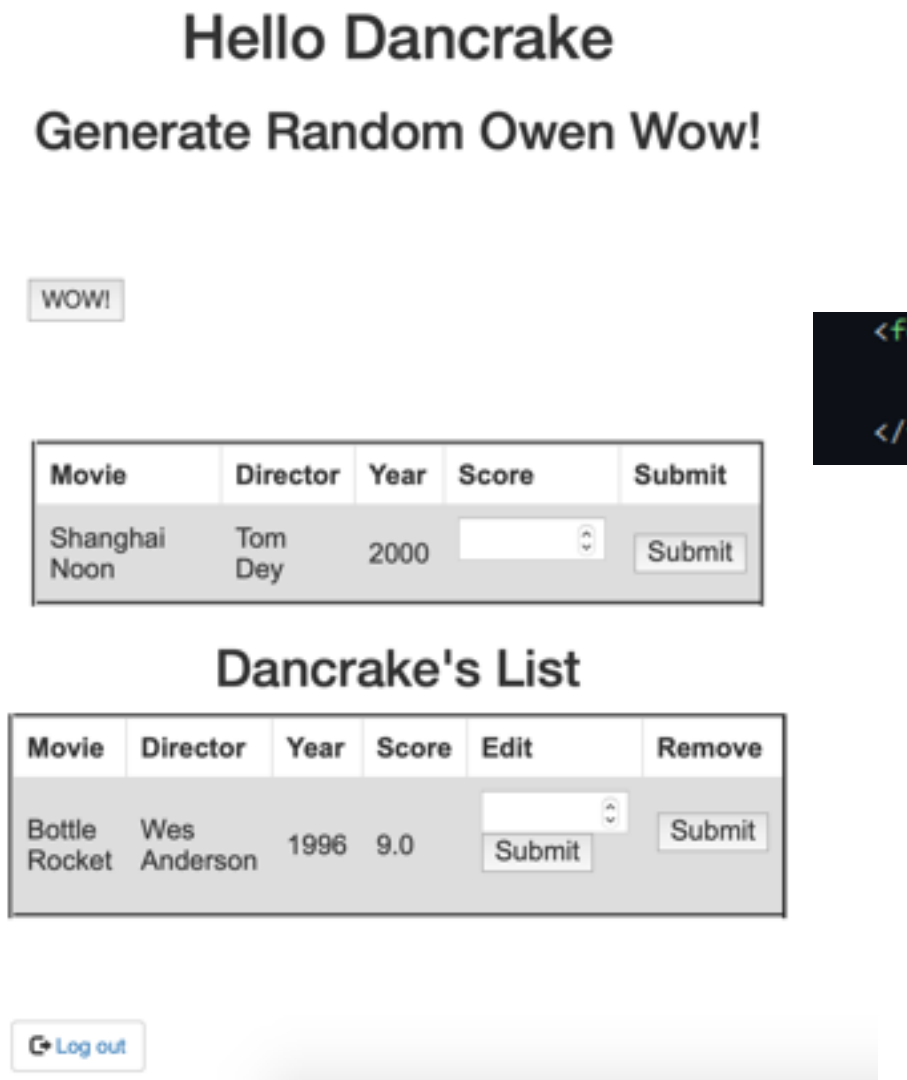
```
@app.route("/user/<username>/", methods = ["GET", "POST"])
def user_home(username):

    user_list = Movie_review.show_list(username = username)

    if request.method == "GET":
        url = "https://owen-wilson-wow-api.herokuapp.com/wows/random"
        resp = requests.get(url)
        response = resp.json()
        movie = response[0].get('movie')
        director = response[0].get('director')
        year = response[0].get('year')
```

Homepage GET Movie

- At the user's homepage, a random movie is generated for them to rate.
- The GET request is used to collect movie information.
- GET request connects to the API via the URL and extracts details of a single movie.
- Details are then displayed to the user in table format.

Queen Mary University of London

Method="GET"




```
<form class=random action="{{url_for('user_home',username = username)}}" method="get">
      <input type="submit" value="WOW!" />
</form>
```

Queen Mary University of London

Method="POST"

```
if request.method == "POST" and "rating" in request.form:

    movie = request.form['movie']
    director = request.form['director']
    year = request.form['year']
    score = request.form['rating']

    review = Movie_review(username = username,
                          movie = movie,
                          director = director,
                          year = year,
                          review = score)

    if Movie_review.find_review(username=username, movie=movie):
        flash("you have already reviewed this movie")

    else:
      db.session.add(review)
      db.session.commit()
      return redirect(url_for("user_home", username = username))
```

Movie rating POST Example

- Once generated, users can rate the movie out of 10.
- The POST request takes the user rating and records the following information.
- The database is examined to check the user has rated the movie previously.
- If unreviewed, the database is updated with the relevant information and stored.

Queen Mary University of London

Method="PUT"

```
if request.method == "POST" and "new_rating" in request.form:

    new_score = request.form['new_rating']
    movie = request.form['movie']
    movie_review = Movie_review.find_review(username = username,
                                            movie = movie)
    movie_review.review = new_score
    db.session.commit()
    return redirect(url_for("user_home", username = username))
```

PUT Example

- Here a user can edit their score via the PUT method.
- If 'new_rating' is present in the request form, the PUT operation is executed.
- The new rating is stored, and the movie review database is queried to select the correct data to be replaced.
- New value is recorded, and database updated.

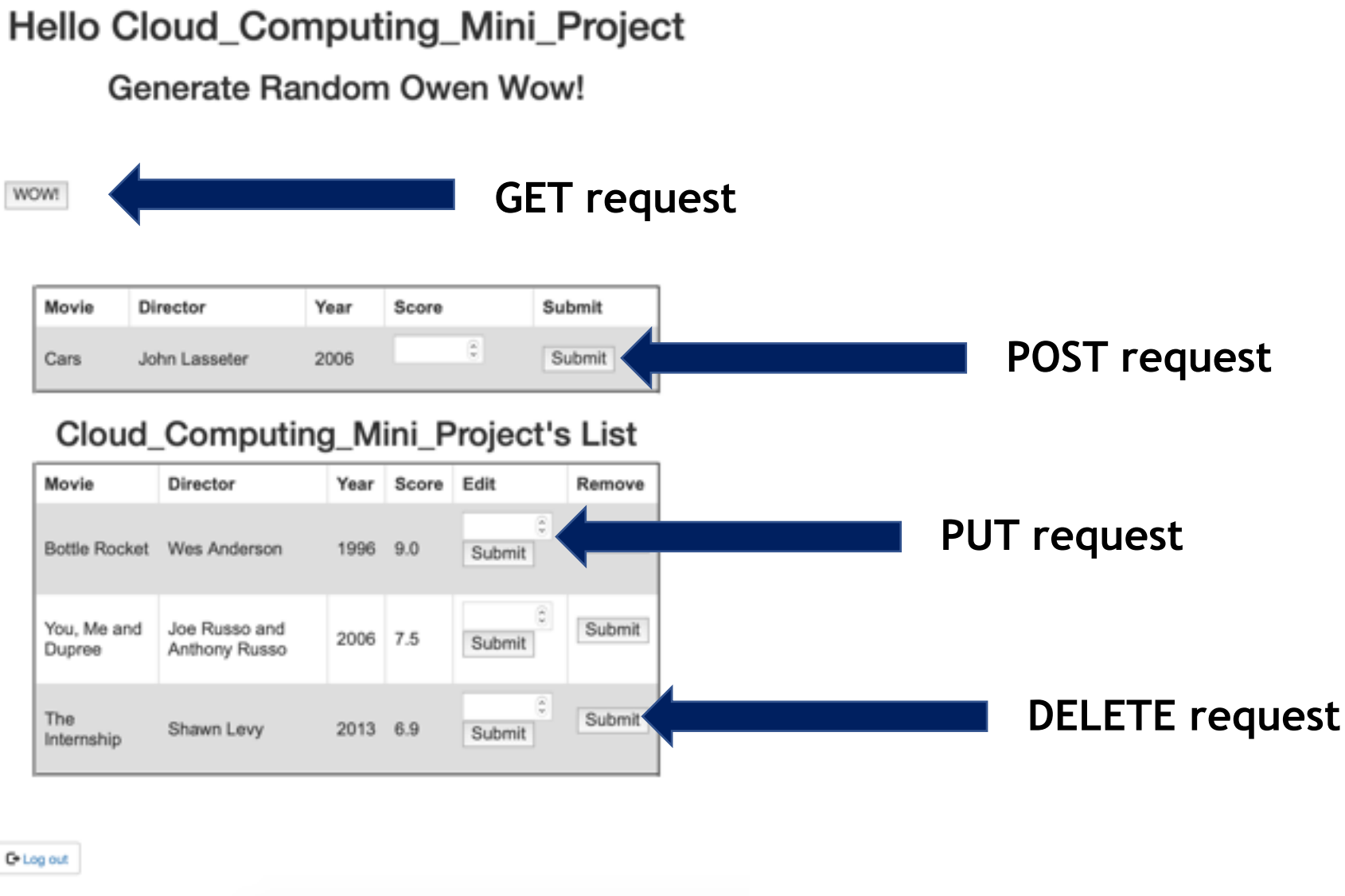
Queen Mary University of London

Method="DELETE"

```
if request.method == "POST" and "delete_request" in request.form:


    delete_request = request.form['delete_request']
    delete_review = Movie_review.find_review(username = username,
                                             movie = delete_request)
    db.session.delete(delete_review)
    db.session.commit()
    return redirect(url_for("user_home", username = username))
```

DELETE Example

- If a user wishes to delete their review all together, they can with the following method.
- If 'delete_request' is present in the request form, the DELETE operation is executed.
- Once a movie review delete button is submitted, the corresponding data is removed from the database.


Hello Cloud_Computing_Mini_Project
Generate Random Owen Wow!
WOW!
GET request
Movie
Director
Year
Score
Submit
Cars
John Lasseter
2006
Submit
POST request
Cloud_Computing_Mini_Project's List
Movie
Director
Year
Score
Edit
Remove
Bottle Rocket
Wes Anderson
1996
9.0
Submit
PUT request
You, Me and Dupree
Joe Russo and Anthony Russo
2006
7.5
Submit
Submit
The Internship
Shawn Levy
2013
6.9
Submit
Submit
DELETE request
Log out

WEB APP WOW!
DEMONSTRATION