

Welcome to itucsd1809's documentation!

Team: Intellect
Members:

- Hakan Eröztekin

Keep your favorite musics and movies in Intellect. Intellect keeps each users favorites and also community favorites. Each user can contribute to community lists, as in Wikipedia. Then user can pick their favorites in that list to shape his/her own list.

There are a different community lists for movies and musics. Users can add, update or delete elements in community lists and changes are reflected to the user lists. User can also shape his/her own list by adding or deleting items from his/her lists.

Contents:

- [User Guide](#)
- [Developer Guide](#)

User Guide

As a visitor of Intellect, you are welcomed with the homepage.



Authentication

You should sign up to get full benefit of Intellect. For that, you should visit the sign up page.

Sign-up to Intellect

Username

Email Address

Name

Surname

Age

Gender

New Password

Repeat Password

With a successful sign-up you are redirected to the sign-in page.

Log In To Intellect

Username

Password

☒ Remember Me

Movies & User Lists

After the sign-in now you can start using Intellect. Initially, your lists are empty as it is shown below.

[Intellect](#)

[Sign in](#)
[Sign up](#)
[Your Lists](#)

[My Lists](#)
[Community Movies](#)
[Community Musics](#)

Welcome John!

Movies

[Add](#)
[Update](#)
[Delete](#)

#	Title	Year	Duration	Director	Genre	Add to Your List
3	Venom	2018	112	Ruben Fleischer	Action	Add

We can add more movies,

Add Update Delete

Title

Year

Duration

Director

Genre

Our community list is growing,

Intellect [Sign in](#) [Sign up](#) [Your Lists](#)

[My Lists](#) [Community Movies](#) [Community Musics](#) Welcome John!

Movies

Add Update Delete

#	Title	Year	Duration	Director	Genre	Add to Your List
3	Venom	2018	112	Ruben Fleischer	Action	<input type="button" value="Add"/>
4	The Shawshank Redemption	1995	142	Frank Darabont	Drama	<input type="button" value="Add"/>
5	The Conjuring	2013	112	James Wan	Horror	<input type="button" value="Add"/>

We can update community movies,

Id of The Movie

Title

Year

Duration

Director

Genre

Here is the result,

Movies

Add **Update** **Delete**

#	Title	Year	Duration	Director	Genre	Add to Your List
3	Venom	2018	112	Ruben Fleischer	Action	<button>Add</button>
4	The Shawshank Redemption	1995	142	Frank Darabont	Drama	<button>Add</button>
5	The Conjuring	2013	112	James Wan	Horror	<button>Add</button>

We can delete movies too,

Id

Register

Here is our updated list,

Movies

Add **Update** **Delete**

#	Title	Year	Duration	Director	Genre	Add to Your List
3	Venom	2018	112	Ruben Fleischer	Action	<button>Add</button>
5	The Conjuring	2013	112	James Wan	Horror	<button>Add</button>

Now that we have existing movies in the Community Movies we can add it to our own Movies list. Just clicking the Add button is enough. Let's add Venom to our list,

Movies

Add **Update** **Delete**

#	Title	Year	Duration	Director	Genre	Add to Your List
3	Venom	2018	112	Ruben Fleischer	Action	<button>Add</button>
5	The Conjuring	2013	112	James Wan	Horror	<button>Add</button>

And here is our list;

YOUR LISTS

Movies

#	Title	Year	Duration	Director	Genre	
3	Venom	2018	112	Ruben Fleischer	Action	<button>Delete</button>

Musics

That's all for the usage of Intellect. If you want more information, you can find similar operations done in Musics list from now on, Also, you can delete your favorites by clicking the Delete button. In addition, you can delete from community lists which'll be cascaded to the user lists. These'll be shown in the next section.

Musics

Let's add couple of music;

Name <input type="text" value="Stolen Dance"/>	Name <input type="text" value="Crazy"/>	Name <input type="text" value="River"/>
Genre <input type="text" value="Indie"/>	Genre <input type="text" value="R&B/Soul"/>	Genre <input type="text" value="Rap"/>
Duration(seconds) <input type="text" value="313"/>	Duration(seconds) <input type="text" value="177"/>	Duration(seconds) <input type="text" value="221"/>
Singer <input type="text" value="Milky Chance"/>	Singer <input type="text" value="Gnas Barkley"/>	Singer <input type="text" value="Eminem"/>
Year <input type="text" value="2013"/>	Year <input type="text" value="2006"/>	Year <input type="text" value="2018"/>
<button>Add Music</button>	<button>Add Music</button>	<button>Add Music</button>

Here we go, this is the Community Musics now;

Intellect

Sign inSign upYour Lists

My ListsCommunity MoviesCommunity Musics

Welcome John!

AddUpdateDelete

#	Name	Genre	Duration(s)	Singer	Year	Add to Your List
1	Stolen Dance	Indie	313	Milky Chance	2013	<div>Add</div>
2	Crazy	R&B/Soul	177	Gnas Barkley	2006	<div>Add</div>
3	River	Rap	221	Eminem	2018	<div>Add</div>

We can also update/delete musics from the list;

/mylists/musics/update

Id of The Music

Name

Genre

Duration(seconds)

Singer

Year

/mylists/musics/delete

Id

Here is the result,

#	Name	Genre	Duration(s)	Singer	Year	Add to Your List
2	Crazy	R&B/Soul	177	Gnas Barkley	2006	<input type="button" value="Add"/>
3	River	Pop	216	Bishop Briggs	2018	<input type="button" value="Add"/>
4	Everything Black	Electronic	229	Unlike Pluto	2017	<input type="button" value="Add"/>
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<input type="button" value="Add"/>

We can “Add” couple of musics to our lists;

Intellect

[Sign in](#) [Sign up](#) [Your Lists](#)

[My Lists](#) [Community Movies](#) [Community Musics](#)

Welcome John!

YOUR LISTS

Movies

#	Title	Year	Duration	Director	Genre	
3	Venom	2018	112	Ruben Fleischer	Action	<input type="button" value="Delete"/>

Musics

#	Name	Genre	Duration	Singer	Year	
4	Everything Black	Electronic	229	Unlike Pluto	2017	<input type="button" value="Delete"/>
2	Crazy	R&B/Soul	177	Gnas Barkley	2006	<input type="button" value="Delete"/>
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<input type="button" value="Delete"/>

Developed by Hakan Eroztekin

We can even “Delete” musics from our lists. Let’s do it in “Everything Black”,

YOUR LISTS

Movies

#	Title	Year	Duration	Director	Genre	
3	Venom	2018	112	Ruben Fleischer	Action	<button>Delete</button>

Musics

#	Name	Genre	Duration	Singer	Year	
2	Crazy	R&B/Soul	177	Gnas Barkley	2006	<button>Delete</button>
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<button>Delete</button>

What happens to user lists when we delete the existing items from the community lists? Here we go. As a reminder, here is the last situation of our list;

YOUR LISTS

Movies

#	Title	Year	Duration	Director	Genre	
3	Venom	2018	112	Ruben Fleischer	Action	<button>Delete</button>

Musics

#	Name	Genre	Duration	Singer	Year	
2	Crazy	R&B/Soul	177	Gnas Barkley	2006	<button>Delete</button>
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<button>Delete</button>

Let's delete two music. One is existing in the user list, one is not,

Id

Register

Id

Register

#	Name	Genre	Duration(s)	Singer	Year	Add to Your List
4	Everything Black	Electronic	229	Unlike Pluto	2017	<button>Add</button>
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<button>Add</button>

Here is the final result:

YOUR LISTS

Movies

#	Title	Year	Duration	Director	Genre	
3	Venom	2018	112	Ruben Fleischer	Action	<button>Delete</button>

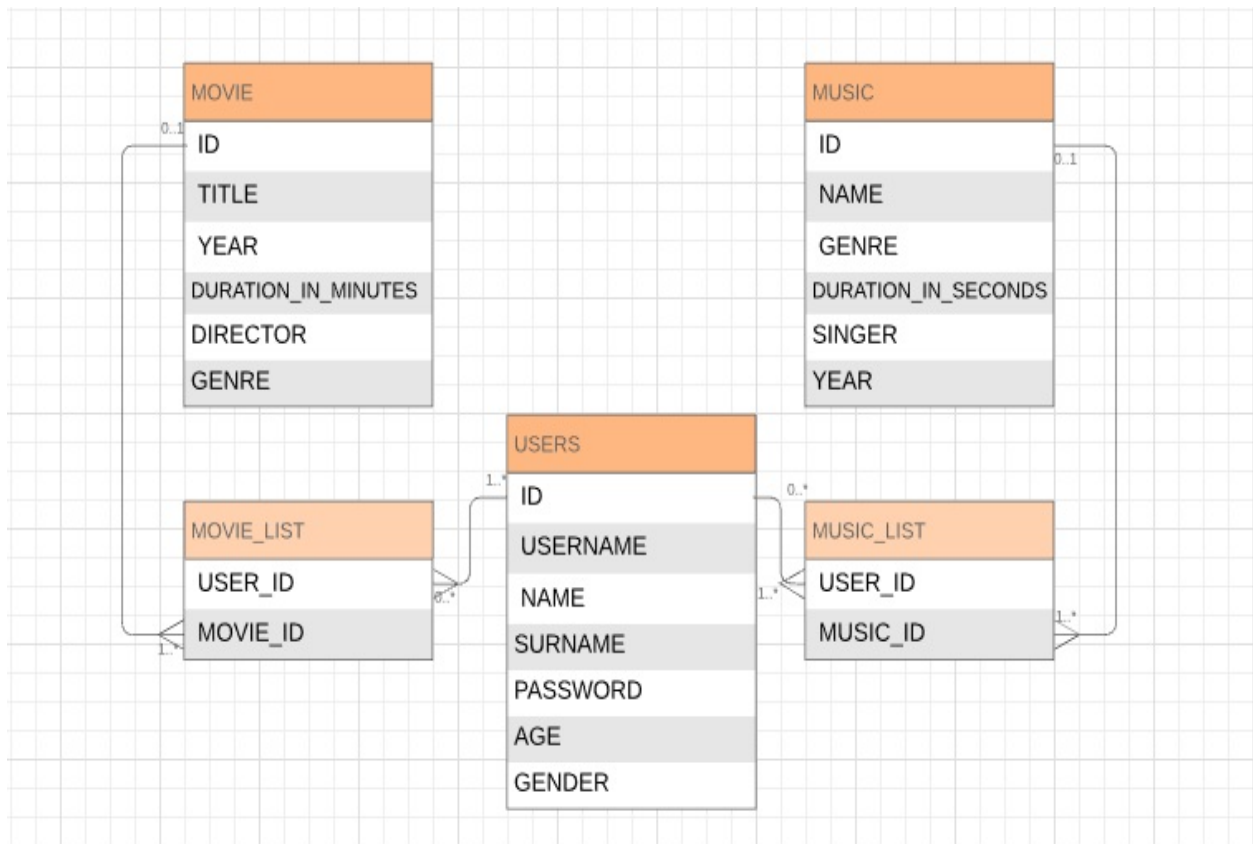
Musics

#	Name	Genre	Duration	Singer	Year	
5	I Can't Go On Without You	Rock	377	Kaleo	2016	<button>Delete</button>

Developer Guide

Database Design

There are three main tables; **USERS**, **MOVIE** and **MUSIC**. **USERS** table is designed to keep the information for user. **MOVIE** table is for movies, and **MUSIC** table is for musics. There are also two additional tables; **MOVIE_LIST** and **MUSIC_LIST**. These are designed to keep user's favorited musics/movies. These keep ID's for USER and MOVIE/MUSIC.



Code

The code can be explained in three parts.

- Python & SQL

- HTML/CSS

Python & SQL

There are the following python(.py) files;

- server.py
- models.py
- dbinit.py
- forms.py
- db_table_operations.py
- config.py

server.py

This is the main python file for the project. It renders pages, initializes and transfers variables and realizes methods with necessary HTML requests.

Firstly, the following packages are imported.

```
from flask import Flask, render_template, request, redirect
from flask_bootstrap import Bootstrap
from flask_login import login_user, LoginManager

from forms import *
from config import Config
from db_table_operations import *
from werkzeug.security import generate_password_hash
```

Then configuration, bootstrap and LoginManager is initialized. LoginManager is used for authentication.

```
app = Flask(__name__)
app.config.from_object(Config)
Bootstrap(app)

login = LoginManager(app)
```

The rest of the server.py works for routes. For example, this renders homepage.html when main route ("/") is accessed.

```
@app.route("/")
def home_page():
```

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

For the sign-up operation, route and necessary HTML requests are announced. Then a form is created (forms will be discussed in the forms.py). When user clicks to the “Register” button, then a new user object is created with information provided by the user. Then the user object is sent to insert_user which will work to insert user to the USERS table. With a successful sign-up operation, users are redirected to the signin page.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup_page():
    form = RegistrationForm(request.form)
    if request.method == 'POST' and form.validate():
        user = User(form.username.data, form.name.data, form.surname
                    form.password.data, form.age.data, form.gender.d
        insert_user(user)
        return redirect('/signin')
```

The similar operations are realized for sign in; route, HTML requests and form. When user clicks “Sign in” button, find_user_by_username method runs a SQL query to find the username in the database. If it can, Flask Login app is informed about the signed up user. The username is also shown in the menu bar at Mylists page.

```
@app.route('/signin', methods=['GET', 'POST'])
def signin_page():
    form = LoginForm()
    if form.validate_on_submit():
        found_user = find_user_by_username(form.username.data)
        user = User(found_user[1], found_user[2], found_user[3], fou
                    found_user[5], found_user[6], found_user[7])
        user.set_id(found_user[0]) # to load user id
        if user is None or not check_password(user.password, form.pa
            print("User signing failed")
            return redirect('/signin')
        login_user(user, remember=form.remember_me.data)
        print("User signed successfully")
        return redirect('/mylists')
    return render_template("signin.html", form=form)
```

From this point on, there are the following methods left in the server.py;

- mylists

- movies (add/update/delete)
- musics (add/update/delete)

Operations for movies and musics are very similar. For the sake of simplicity, mylists and movies will be discussed.

Mylists page tries to get elements from user's own lists as a first step. If there is any element exists, it shows them.

Also, mylists page has "Delete" button for items in the user's list. When it is clicked, it takes the related elements id (music or movie) and user's id. Then it sends it to delete_movie (delete_music for music) operation to run SQL query to delete it from the database.

As it can be noticed, mylists page provides user_movies and user_musics variables while it is rendering the html file. These variables are going to be processed with the html file to fill the tables.

```
@app.route('/mylists', methods=['GET', 'POST'])
def mylists_page():
    user_movies = userlist_get_movies()
    user_musics = userlist_get_musics()

    if request.method == 'POST':
        movie_id = request.form['movie_id']
        user_id = request.form['user_id']
        music_id = request.form['music_id']

        if movie_id != "0": # request to delete movie
            userlist_delete_movie(user_id, movie_id)
            print("delete movie with id " + movie_id)
            return redirect('/mylists')

        elif music_id != "0": # request to delete music
            userlist_delete_music(user_id, music_id)
            return redirect('/mylists')

    return render_template("mylists.html",
                           user_movies=user_movies,
                           user_musics=user_musics)
```

Movies page works similarly with the mylists page. It shows the movies in the MOVIE table in sorted order. It has "Add" button for user to add the selected movie to his/her lists. Added elements are going to be shown in users list in

mylists page.

```
def movies_page():
    movies = get_movies()

    if request.method == 'POST':
        movie_id = request.form['movie_id']
        user_id = request.form['user_id']
        userlist_add_movie(user_id, movie_id)
        return redirect('/mylists')

    return render_template("movies.html", movies=sorted(movies))
```

Adding a movie is fairly simple and similar to the user sign up operation. It works with MovieAddForm and creates a movie object with provided information. Then to run SQL query, it sends the movie object to insert_movie() method.

```
@app.route('/mylists/movies/add', methods=['GET', 'POST'])
def movies_add_page():
    form = MovieAddForm(request.form)
    if request.method == 'POST' and form.validate():
        movie = Movie(form.title.data, form.year.data, form.duration,
                      form.director.data, form.genre.data)
        insert_movie(movie)
        return redirect('/mylists/movies')
    return render_template("add_movies.html", form=form)
```

Movie update page also gets id to be deleted.

```
def movies_update_page():
    form = MovieUpdateForm(request.form)
    if request.method == 'POST' and form.validate():
        movie = Movie(form.title.data, form.year.data, form.duration,
                      form.director.data, form.genre.data)
        movie_id = form.id.data # which movie to update
        update_movie(movie_id, movie)
        return redirect('/mylists/movies')

    return render_template("update_movies.html", form=form)
```

Movie delete page gets id that will be deleted from the database table, MOVIE.

```
def movies_delete_page():
    form = MovieDeleteForm(request.form)
```

```

id = form.movie_id.data
if id.__len__() > 0:
    delete_movie(id)
    return redirect('/mylists/movies')
return render_template("delete_movies.html", form=form)

```

models.py models.py includes User, Movie and Music classes.

The most technical part is about the login. initialize_login method is implemented as a solution for circular import between server.py and models.py. After the initialization of “login”, it is used to load users. This is related with Flask’s LoginManager package.

```

def initialize_login():
    from server import login
    global login
    login = login

class User(UserMixin):
    initialize_login()

    def __init__(self, username, name, surname, email, password, age,
                 gender):
        self.id = 0
        self.username = username
        self.name = name
        self.surname = surname
        self.email = email
        self.password = password
        self.age = int(age)
        self.gender = gender
        print("User object created")

@login.user_loader
def load_user(id):
    if id == 0:
        print("User not logged in, id is ", id)
        return
    else:
        found_user = find_user_by_id(int(id))
        return found_user

    def __repr__(self):
        return '<User %r>' % self.username

    def set_id(self, id):
        self.id = id

    def get_id(self):

```



```

        return self.id;

class Music:
    def __init__(self, name, genre, duration_in_seconds, singer, year):
        self.name = name
        self.genre = genre
        self.duration_in_seconds = duration_in_seconds
        self.singer = singer
        self.year = year
        print("Music object created")

class Movie:
    def __init__(self, title, year, duration_in_minutes, director, genre):
        self.title = title
        self.year = year
        self.genre = genre
        self.duration_in_minutes = duration_in_minutes
        self.director = director
        print("Movie object created")

```

dbinit.py dbinit.py is used for database initialization. It runs INIT_STATEMENTS which has the purpose to create the tables, USERS, MOVIE, MUSIC.

```

INIT_STATEMENTS = [
    "DROP TABLE IF EXISTS USERS CASCADE", # to test changes quickly
    "DROP TABLE IF EXISTS MUSIC CASCADE",
    "DROP TABLE IF EXISTS MOVIE CASCADE",
    "DROP TABLE IF EXISTS MUSIC_LIST CASCADE",
    "DROP TABLE IF EXISTS MOVIE_LIST CASCADE",

    "CREATE TABLE IF NOT EXISTS USERS("
        "ID SERIAL,"
        "USERNAME VARCHAR(30) NOT NULL,"
        "NAME VARCHAR(30),"
        "SURNAME VARCHAR(30),"
        "EMAIL VARCHAR(30),"
        "PASSWORD VARCHAR(100),"
        "AGE VARCHAR(8),"
        "GENDER VARCHAR(15),"
        "PRIMARY KEY(ID)"
    ")",

    "CREATE TABLE IF NOT EXISTS MUSIC("
        "ID SERIAL,"
        "NAME VARCHAR(30),"
        "GENRE VARCHAR(30),"

```

```

"DURATION_IN_SECONDS VARCHAR(8), "
"SINGER VARCHAR(30), "
"YEAR VARCHAR(8), "
"PRIMARY KEY(ID)"
)"",

"CREATE TABLE IF NOT EXISTS MOVIE("
"ID SERIAL, "
"TITLE VARCHAR(40), "
"YEAR VARCHAR(8), "
"DURATION_IN_MINUTES VARCHAR(8), "
"DIRECTOR VARCHAR(30), "
"GENRE VARCHAR(30), "
"PRIMARY KEY(ID)"
)"",

```

Notice that USER_ID and MUSIC_ID/MOVIE_ID are the primary keys for these tables. Also, they got cascading.

```

"CREATE TABLE IF NOT EXISTS MUSIC_LIST("
"USER_ID INTEGER REFERENCES USERS(id) ON DELETE CASCADE, "
"MUSIC_ID INTEGER REFERENCES MUSIC(id) ON DELETE CASCADE, "
"PRIMARY KEY(USER_ID, MUSIC_ID)"
)"",

"CREATE TABLE IF NOT EXISTS MOVIE_LIST("
"USER_ID INTEGER REFERENCES USERS(id) ON DELETE CASCADE, "
"MOVIE_ID INTEGER REFERENCES MOVIE(id) ON DELETE CASCADE, "
"PRIMARY KEY(USER_ID, MOVIE_ID)"
)"",

```

]

forms.py forms.py is used to create forms necessary for the related operations.

```

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Sign In')

class RegistrationForm(Form):
    username = StringField('Username', [validators.Length(min=3, max=
    email = StringField('Email Address', [validators.Length(min=4, m
    name = StringField('Name', [validators.Length(min=0, max=20)])
    surname = StringField('Surname', [validators.Length(min=0, max=2
    age = StringField('Age', [validators.Length(min=0, max=20)])
    gender = StringField('Gender', [validators.Length(min=0, max=15)

```

```

password = PasswordField('New Password', [
    validators.DataRequired(),
    validators.EqualTo('confirm', message='Passwords must match'
)])
confirm = PasswordField('Repeat Password')

# MOVIE OPERATIONS
class MovieAddForm(Form):
    title = StringField('Title', [validators.Length(min=3, max=30)])
    year = StringField('Year', [validators.Length(min=4, max=30)])
    duration_in_minutes = StringField('Duration', [validators.Length
director = StringField('Director', [validators.Length(min=0, max
genre = StringField('Genre', [validators.Length(min=0, max=20)])

class MovieUpdateForm(Form):
    id = StringField('Id of The Movie', [validators.Length(min=1, ma
    title = StringField('Title', [validators.Length(min=3, max=30)])
    year = StringField('Year', [validators.Length(min=4, max=30)])
    duration_in_minutes = StringField('Duration', [validators.Length
    director = StringField('Director', [validators.Length(min=0, max
    genre = StringField('Genre', [validators.Length(min=0, max=20)])

class MovieDeleteForm(Form):
    movie_id = StringField('Id', [validators.Length(min=1, max=5)])

#MUSIC OPERATIONS
class MusicAddForm(Form):
    name = StringField('Name', [validators.Length(min=3, max=30)])
    genre = StringField('Genre', [validators.Length(min=3, max=30)])
    duration_in_seconds = StringField('Duration(seconds)', [validato
    singer = StringField('Singer', [validators.Length(min=0, max=20)
    year = StringField('Year', [validators.Length(min=0, max=20)])

class MusicUpdateForm(Form):
    id = StringField('Id of The Music', [validators.Length(min=1, ma
    name = StringField('Name', [validators.Length(min=3, max=30)])
    genre = StringField('Genre', [validators.Length(min=3, max=30)])
    duration_in_seconds = StringField('Duration(seconds)', [validato
    singer = StringField('Singer', [validators.Length(min=0, max=20)
    year = StringField('Year', [validators.Length(min=0, max=20)])

class MusicDeleteForm(Form):
    music_id = StringField('Id', [validators.Length(min=1, max=5)])

```

db_table_operations.py This file is to execute SQL queries needed for database operations.

Insert a user;

```
# USER AUTHENTICATION TABLE OPERATIONS #
def insert_user(object):
    query = 'INSERT INTO USERS (USERNAME, NAME, SURNAME, EMAIL, PASSWORD,
        'VALUES(%s, %s, %s, %s, %s, %s, %s)'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        # hash the password
        print("User pw:" + object.password)
        password_hash = generate_password_hash(object.password)
        print("User pw:" + password_hash)
        cursor.execute(query, (object.username, object.name, object.surname,
            password_hash, object.age, object.gender))
    cursor.close()
```

object is user object which has fields like username, name, password etc. They are executed as strings(%s). Also notice that the password is hashed for security.

Find a user; Columns are selected with SELECT operation.

```
def find_user_by_username(username):
    query = "SELECT * FROM USERS WHERE USERNAME = %s"
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        rows_count = cursor.execute(query, (username,))
        print("User is found in DB")
        found_user = cursor.fetchone()
        cursor.close()
        return found_user

def find_user_by_id(id):
    query = "SELECT * FROM USERS WHERE ID = %s"
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        rows_count = cursor.execute(query, (id,))
        print("User is found in DB")
        found_user = cursor.fetchone()
        cursor.close()
        return found_user
```

Check password; Hashed passwords are checked.

```
def check_password(user_password_hash, form_password):
    # compare the passwords
```

```
return check_password_hash(user_password_hash, form_password)
```

Add/Update/Delete elements from User Lists (i.e. mylists);

```
# USER LIST OPERATIONS
```

```
def userlist_add_music(user_id, music_id):
    query = 'INSERT INTO MUSIC_LIST (USER_ID, MUSIC_ID) VALUES(%s, %s)'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (user_id, music_id))
        cursor.close()

def userlist_add_movie(user_id, movie_id):
    query = 'INSERT INTO MOVIE_LIST (USER_ID, MOVIE_ID) VALUES(%s, %s)'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (user_id, movie_id))
        cursor.close()
```

User's lists, i.e, MOVIE_LIST and MUSIC_LIST keeps only the ID's of the movie/music and user. So it is necessary to use JOIN operation to find out the row related with these ID's.

```
def userlist_get_movies():
    query = 'SELECT MOVIE.* FROM MOVIE_LIST ' \
            'INNER JOIN MOVIE ON MOVIE_LIST.MOVIE_ID = MOVIE.ID ' \
            'INNER JOIN USERS ON MOVIE_LIST.USER_ID = USERS.ID'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query)
        movies = cursor.fetchall()
        cursor.close()
    return movies

def userlist_get_musics():
    query = 'SELECT MUSIC.* FROM MUSIC_LIST ' \
            'INNER JOIN MUSIC ON MUSIC_LIST.MUSIC_ID = MUSIC.ID ' \
            'INNER JOIN USERS ON MUSIC_LIST.USER_ID = USERS.ID'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query)
        musics = cursor.fetchall()
        cursor.close()
```

```
    return musics
```

Delete operation finds the element with provided ID and deletes it from the related table.

```
def userlist_delete_movie(user_id, movie_id):
    query = "DELETE FROM MOVIE_LIST WHERE USER_ID = CAST(%s AS INTEG
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (user_id, movie_id))
        cursor.close()
        print("Movie with id " + movie_id + " from user with id " +

def userlist_delete_music(user_id, music_id):
    query = "DELETE FROM MUSIC_LIST WHERE USER_ID = CAST(%s AS INTEG
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (user_id, music_id))
        cursor.close()
        print("Music with id " + music_id + " from user with id " +
```

Rest of the MOVIE table operations are similar to the former operations.

```
# MOVIE TABLE OPERATIONS #
def get_movies():
    query = 'SELECT * FROM MOVIE'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query)
        movies = cursor.fetchall()
        cursor.close()
        return movies

def insert_movie(movie):
    query = 'INSERT INTO MOVIE(TITLE, YEAR, DURATION_IN_MINUTES, DIRE
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (movie.title, movie.year, movie.durati
        cursor.close()

def update_movie(movie_id, movie):
    query = "UPDATE MOVIE " \
```

```

        "SET TITLE = %s, " \
        "YEAR = %s, " \
        "DURATION_IN_MINUTES = %s, " \
        "DIRECTOR = %s, " \
        "GENRE = %s " \
        "WHERE ID = %s "
url = get_db_url()
with dbapi2.connect(url) as connection:
    cursor = connection.cursor()
    cursor.execute(query, (movie.title, movie.year, movie.durati
                           movie.director, movie.genre, movie_id,
    # id = cursor.fetchone()[0] # get the inserted row's id
    cursor.close()
    print("Movie with id " + movie_id + " deleted")
    # return id

```

Operations in MUSICS table are pretty the same. Though, they are provided to provide details for SQL operations.

```

# MUSIC TABLE OPERATIONS #
def get_musics():
    query = 'SELECT * FROM MUSIC'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query)
        movies = cursor.fetchall()
        cursor.close()
        return movies

def insert_music(music):
    query = 'INSERT INTO MUSIC (NAME, GENRE, DURATION_IN_SECONDS, SINGER, YEAR)'
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (music.name, music.genre, music.duration_in_seconds,
                                music.singer, music.year))
        cursor.close()

def update_music(music_id, music):
    query = "UPDATE MUSIC " \
            "SET NAME = %s, " \
            "GENRE = %s, " \
            "DURATION_IN_SECONDS = %s, " \
            "SINGER = %s, " \
            "YEAR = %s " \


```

```

        "WHERE ID = %s "
url = get_db_url()
with dbapi2.connect(url) as connection:
    cursor = connection.cursor()
    cursor.execute(query, (music.name, music.genre, music.durati
                           music.singer, music.year, music_id,))
    cursor.close()
    print("Music with id " + music_id + " deleted")

def delete_music(music_id):
    query = "DELETE FROM MUSIC WHERE ID = CAST(%s AS INTEGER)"
    url = get_db_url()
    with dbapi2.connect(url) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (music_id,))
        cursor.close()
        print("Music with id " + music_id + " deleted")

```



Lastly, config.py has the secret key for security.

HTML/CSS

There are a differentiated HTML code for each page. Though, they all share the common one, layout.html. Let's start with it.

Let's start with CSS part of the layout.html.

.. code-block:: python

```

<style>
html, body, h1, h2, h3, h4, h5, h6 {
font-family: 'Hammersmith One', serif;
}

```

The code part above, font is determined. Then the rest of the code layouts the general style.

```

* {
    box-sizing: border-box;
}

/* Create two equal columns that floats next to each other */
.column {
    float: left;
    padding: 10px;
}

```



```

padding-right: 15px;
}

.left {
width: 85%;
}

.right {
width: 15%
}

{# responsive to screen size #}
@media screen and (max-width: 600px) {
.column {
width: 100%;
}
}

/* Clear floats after the columns */
.row:after {
content: "";
display: table;
clear: both;
}
</style>


```

Below you can see the jquery and Bootstrap scripts initializations.

```

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jque
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/boot

```



Now we create “title” block which will be different for each page.

```

<title>{% block title %}{% endblock %}</title>

```

After, we create our menu. It is designed as two lines. The first one consists of couple links for necessary redirections. Because it’s same for all the pages, it is not labeled as block, as in the second menu bar. Second menu bar includes links to the community lists. It also has a welcome message with included username. Because it’s only necessary for mylists page it is labeled as secondmenubar.

```

<div class="w3-bar w3-text-yellow w3-large">
  <h5>
    <a href="/mylists"><button class="w3-bar-item w3-button w3-r
    <a href="/signup"><button class="w3-bar-item w3-button w3-ri
    <a href="/signin"><button class="w3-bar-item w3-button w3-ri


```

```

        <a href="/"><button class="w3-bar-item w3-button w3-left"><b
    </h5>
</div>

{% block secondmenubar %}
<div class="w3-bar w3-text-brown w3-large w3-border-top w3-border-bo
    <h5>
        <a href="/mylists"><button class="w3-bar-item w3-button w3-l
        <a href="/mylists/movies"><button class="w3-bar-item w3-butt
        <a href="/mylists/musics"><button class="w3-bar-item w3-butt
        <button class="w3-bar-item w3-button w3-right"><b>Welcome {{
    </h5>
</div>
{% endblock %}

```



Then create a block for body. It's empty for the homepage.

```

<body>
    {% block content %}{% endblock %}
</body>

```

Create the footer;

```

{    % block footer %}
<footer class="w3-bottom w3-container w3-text-gray">
    <p>Developed by Hakan Eroztekin</p>
</footer>
{% endblock %}

```

Block for additional styles;

```

{% block additional_styles %}
<style>
    body {
        padding-left: 10px;
    }
</style>
{% endblock %}

```

That's it for layout.html. All the other pages are derived from this page. While it is common, they differ in contents in the blocks. There are 12 more html pages.

- **User Authentication:** signup.html, signin.html
- **Homepage:** homepage.html
- **User Lists:** mylists.html

- **Community Lists for Movies:** movies.html, add_movies.html, update_movies.html, delete_movies.html
- **Community Lists for Musics:** musics.html, add_musics.html, update_musics.html, delete_musics.html

Almost all the pages are created to work with forms. The forms has fields according to the related database tables.

User Authentication *signup.html* The technical part of the page can be seen below:

```
{% block content %}
    <h1>Sign-up to Intellect</h1>
    <form method=post>
    <dl>
        {{ render_field(form.username) }}
        {{ render_field(form.email) }}
        {{ render_field(form.name) }}
        {{ render_field(form.surname) }}
        {{ render_field(form.age) }}
        {{ render_field(form.gender) }}
        {{ render_field(form.password) }}
        {{ render_field(form.confirm) }}
    </dl>
    <p><input type=submit value=Register>
{% endblock %}
```

Each form element is rendered in the html.

User Authentication *signin.html* In sign-in, username and password is asked. Also, there is an option to “Remember” the user later on.

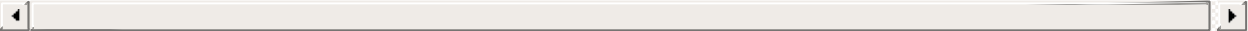
```
{% block title %} Sign-in for Intellect {% endblock %}
{% block content %}
    <h1>Log In To Intellect</h1>
    <form action="" method="post" novalidate>
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}
        </p>
    </form>
```

```

        <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
        <p>{{ form.submit() }}</p>
    </form>
    {% if error %}
        <p class="error">ERROR: {{ error }}</p>
    {% endif %}

{% endblock %}

```



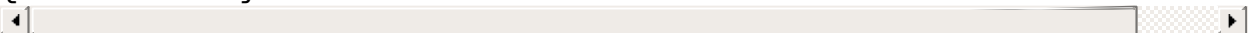
Homepage *homepage.html* Homepage has a difference in the additional styles here. A specific background image is implemented.

```

{% block additional_styles %}
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        body {
            background-image: url('/static/image/bg.jpg');
            height: 100%;
            background-repeat: no-repeat;
            background-size: cover;
            position: relative;
        }
    </style>

{% endblock %}

```



At this point, the remaining pages are the following, - **User Lists:** mylists.html - **Community Lists for Movies:** movies.html, add_movies.html, update_movies.html, delete_movies.html - **Community Lists for Musics:** musics.html, add_musics.html, update_musics.html, delete_musics.html

Because the operation is very similar, to keep the documentation neat, only the technical parts will be discussed.

User Lists: mylists.html MyLists is the page shows user's own lists. It creates table and fills it with user_movies list. user_movies is provided to html in server.py.

```

{% if user_movies %}
    <table class="w3-table-all">
        <thead>
            <tr class="w3-orange">
                <th>#</th>
                <th>Title</th>
            </tr>
        </thead>
        <tbody>
            {% for movie in user_movies %}
                <tr>
                    <td>{{ movie.id }}</td>
                    <td>{{ movie.title }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endif %}

```

```

        <th>Year</th>
        <th>Duration</th>
        <th>Director</th>
        <th>Genre</th>
        <th></th>
    </tr>
</thead>
    {% for movie in user_movies %}
<tr class="w3-text-black">
    <td>{{ movie[0] }}</td>
    <td>{{ movie[1] }}</td>
    <td>{{ movie[2] }}</td>
    <td>{{ movie[3] }}</td>
    <td>{{ movie[4] }}</td>
    <td>{{ movie[5] }}</td>

    <form method=post>

        {# get movie id#}

        <td>
            <input type="hidden" name="movie_id" value="{{ mo
            <input type="hidden" name="user_id" value="{{ cur
            <input type="hidden" name="music_id" value="0">
            <input type=submit value="Delete">
        </td>
    </form>

</tr>
{% endfor %}
</table>
</div>
    {% endif %}

```

The very same operation is realized for movies.html and musics.html.

- **Community Lists for Movies & Musics :**

Adding, updating and deleting movies and music operations are pretty similar. They process the form fields as text boxes, and with the click of the button, the information is transferred to the related variable and it's processed in the database.

```

<form method=post>
    <dl>
        {{ render_field(form.title) }}
        {{ render_field(form.year) }}

```

```
    {{ render_field(form.duration_in_minutes) }}
    {{ render_field(form.director) }}
    {{ render_field(form.genre) }}
</dl>
<p><input type=submit value="Add Movie">
```

- [Parts Implemented by Member Name](#)

Parts Implemented by Member Name

All the parts are implemented by Hakan Eröztekin.

Index

Parts Implemented by Member Name

All the parts are implemented by Hakan Eröztekin.