



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Assignment of bachelor's thesis

Title:	Extensions of SoundPi project by social aspects and machine learning integration
Student:	Egemen Eroglu
Supervisor:	Ing. Michal Valenta, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

The aim of the thesis is to follow already started students project called SoundPi and extend it by particular functionalities.

Follow the steps below:

1. Describe the actual state of the project.
2. Design and implement the following extensions / modules / functionalities:
 - Songs recommendation. Recommender engine is developed in thesis running in parallel. Your responsibility is appropriate data model extension and integration into the application.
 - "Jukebox" for bar / house party events.
 - Host/admin access to re-edit event details.
 - Addition of a social aspect through user interaction via a comment section per event.

Properly test and document your solution.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Auxbox Project Extension

Egemen Eroglu

Department of Software Engineering
Supervisor: Michal Valenta

January 6, 2022

Acknowledgements

I want to thank all of my Software Project team members who worked hard and helped create the original AuxBox application which I have built up on, as well as my supervisor Michal Valenta for guiding me through the process of writing my thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No.121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on January 6, 2022

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2022 Egemen Erogul. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Erogul, Egemen. *Auxbox Project Extension*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce se zabývá rozšířením webové aplikace AuxBox o další funkce, které přináší nové případy užití aplikace, poskytují lepší navigaci, interaktivnější prostředí a zavádí do aplikace prvky sociálních sítí. Dále se věnuje přípravě doporučovacího systému pro výběr písní založeného na informacích o uživatelích v systémech AuxBox a Spotify.

Klíčová slova Webová aplikace, Spotify, Sociální, Strojové učení, Python, Django, Systém doporučení, Hudba.

Abstract

This thesis focuses on extending the web application AuxBox by adding additional functionalities which will introduce new use cases allowing the users to navigate and work with the application in a more interactive, social, convenient and extensive way along with a research on recommendation systems which is going to provide a road map of asserting trained data models that would be used to build the cores of a recommendation system based on the data provided by Auxbox/Spotify users.

Keywords Web Application, Spotify, Social , Machine learning , Python, Django, Recommendation System, Music.

Contents

Introduction	1
Sub Objectives:	1
Used Technologies And Frameworks	2
Overview	2
How Django Works - Overview	2
Database Management	2
Security	3
Request Handling And Views	3
Further Reading	3
1 Architecture And Class Design	5
1.1 Directory Structure	5
1.1.1 auxbox	6
1.1.2 docs	6
1.1.3 Documentation	7
1.1.4 events	7
1.1.5 dao	7
1.1.6 daointerface	7
1.1.7 factory	7
1.1.8 forms	7
1.1.9 migrations	7
1.1.10 models	7
1.1.11 services	8
1.1.12 static	8
1.1.13 templates	8
1.1.14 tests	8
1.1.15 views	8
1.2 Data Layer	8
1.3 Service Layer	9

1.3.1	Authentication	9
1.3.2	Query Processing	9
1.4	Presentation Layer	9
2	New Use Cases	11
2.1	User Comments	11
2.1.1	Model Design	11
2.1.2	Usage Specification	12
2.2	Event Likes	13
2.2.1	Model Design	13
2.2.2	Usage Specification	14
2.3	Create House Party	14
2.3.1	Design	15
2.3.2	Usage Specification	16
2.4	Add Playlist To House Party	16
2.4.1	Design	17
2.4.2	Usage Specification	18
2.5	Up Vote Playlist	18
2.5.1	Design	18
2.5.2	Usage Specification	19
2.6	Edit Event	19
2.6.1	Design	19
2.6.2	Usage Specification	20
3	Testing	21
3.1	Architecture	21
3.1.1	Overview	21
3.1.2	Design	21
4	Recommendation System	23
4.1	Abstract	23
4.2	Recommendation Methods	23
4.2.1	Abstract	23
4.2.2	Simple Recommendation Methods	24
4.2.3	Content Based Recommendation	25
4.2.4	Collaborative Filtering	26
4.2.5	Collaborative Recommendation System Types	27
4.2.6	Memory-Based Collaborative Filtering	27
4.2.7	User-User filtering	27
4.2.8	Item-Item filtering	27
4.2.9	Model-Based Collaborative Filtering	27
4.2.10	Clustering Algorithms	27
4.2.11	Matrix factorization	28
4.2.12	Deep Learning Methods	28

4.3 Integration With AuxBox	28
Conclusion	29
Future Work	31
Bibliography	33
A	35

List of Figures

Introduction

Auxbox is a web application which is designed to create events particularly taking place in bars and clubs, targeting to achieve a much more interactive environment by allowing the users to be able to manipulate the music that is to be played both during, and before the event. The current state of the application is more of a prototype rather than a complete app which is able to maintain a relatively simple database and provide simple use cases like creating and joining events. The main objective of this thesis is to extend the application in a way which will transform the application in to a much more responsive, functional and social platform and establishing the initial architecture by doing the necessary research for a recommendation system which will open the ways of using trained real data related to users music preferences to make the AuxBox experience even more engaging and dynamically improving over time.

Sub Objectives:

The thesis focuses on accomplishing the following sub goals:

- Implementation of host / admin access to re-edit event details.
- Addition of a social aspect through user interaction via a comment section per event.
- Implementation of a “Jukebox” type event for bars / house parties
- Updating / Reorganizing the database model of the application to fit the needs of the chosen machine learning model.
- Implementation of a song request feature with the token system
- Analysis of the optimal way to integrate the machine learning model with the back end of the currently existing application.

Used Technologies And Frameworks

Overview

Auxbox is written mainly in Python. The reason Python language was chosen is because of its easy to use, well documented frameworks which also have outstanding communities that makes it convenient to deal with possible bugs and library issues which are known to be problematic and time consuming. python is also known for its statistical libraries like Pandas and Numpy along with frameworks that are designed for handling big data sets and applying machine learning algorithms on them which is very crucial for the recommendation system aspect of Auxbox.

The main framework Auxbox uses to maintain its web server is the Django. Django is arguably one of the best frameworks for web development for Python. It is competitively fast and does a very good job of handling common web issues And provides very straightforward and easy to understand documentation. It is fair to say that Django framework is the main technology component which helped Auxbox become a stable , functional and highly scalable web application.

Auxbox uses a PostgreSQL database for storing maintaining all of its data. PostgreSQL is one of, if not the best database system int the world. It is used by technology leader companies and its safe to say that it proved itself to be very reliable and robust. Auxbox aims for simplicity, reliability aspects on maintaining its database and Posgres was able to full fill those requirements.

How Django Works - Overview

Database Management

One of the key aspects of Django framework is the amount of convenience it brings to the database management and maintenance problem. It is entirely possible to use Djnago without an actual database to connect the projects with. Though, Django comes prepared with a simple and functioning ORM(object-relational- mapping) component for developers who are looking for more convenience in the databse department. The Django object relational mapper describes the database table layout in Python. Their data model syntax comes with a lot of useful features to represent the table structure in the best way possible. This can be observed through the models directory in AuxBox projects where every table is represented in its own file and class as models. Once a model is ready to be used, developers only need to execute a few commands in order to connect to the database and alter/create tables based on the designed models.

Security

Django provides an administrative interface for professional use which allows the registered users to change the database objects of a specific model if the model is registered in the admin site. This way it is possible control user access among clients and staff.

Request Handling And Views

Django handles request urls in a fast and convenient way. Conventionally, `urls.py` file contains the urls as path objects which define a mapping between url patterns and call back functions called views or view functions. When a url is requested from the client Django goes through every single defined path in order and triggers the view function which is associated with the first url match. These urls are compiled to regular expressions at load time which makes it very fast.

The view functions(or classes) are responsible for either returning an http response object according to the request object which was implicitly passed to it or raising an error. In general, view functions would render a template after retrieving the necessary data with regards to the parameters.

Django also provides a template system which helps reduce redundancy in terms of HTML pages which comes with many features.

Further Reading

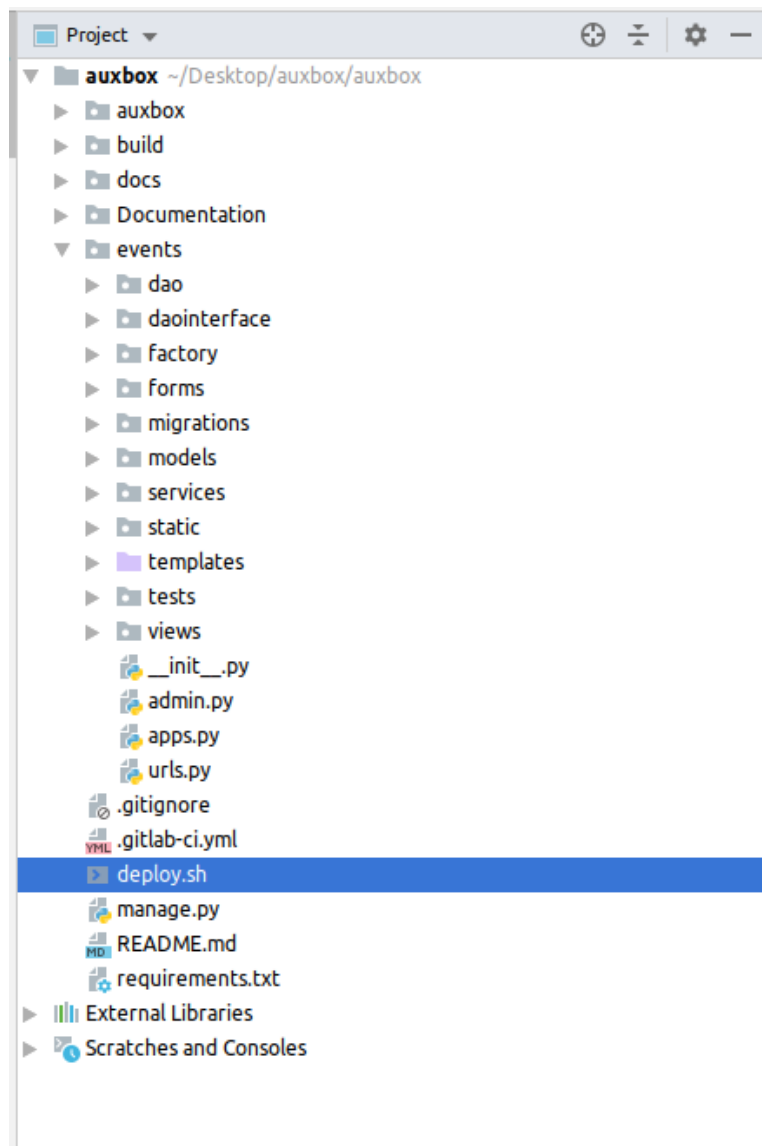
This section describes only the main components and just a small percentage of what Django has to offer as a general purpose python framework. For more information, visit [<https://docs.djangoproject.com/>]

Architecture And Class Design

1.1 Directory Structure

AuxBox directory structure follows the conventions of Django framework and python.

1. ARCHITECTURE AND CLASS DESIGN



AuxBox

directory structure (SCREENSHOT FROM IntelliJ IDE)

1.1.1 auxbox

The auxbox directory(not to be confused with the main project directory) is used to hold files regarding project settings and configuration variables like secret key, client id etc.

1.1.2 docs

The docs directory stores the files which represent automatically generated documentation of auxbox as rst files.

1.1.3 Documentation

The Documentation directory is the place where human readable documentation of AuxBox is stored in the form of md and pdf files.

1.1.4 events

Events is the main directory of AuxBox project where every piece of implemented logic that has direct impact is stored in related directories and files.

1.1.5 dao

Dao directory contains all the files which are classes that represent data access objects. These classes are responsible for querying the database based on the requests of the service objects.

1.1.6 daointerface

DaoInterface directory contains interface classes which are used for creating abstraction over dao classes.

1.1.7 factory

Factory directory consists of a single file which represents the class that is used by service classes to access data access objects effectively implementing the classic factory design pattern.

1.1.8 forms

Forms directory is the place where all the Django Form classes take place. These classes are used by the presentation layer to create form objects which are then rendered for the user to fill in and send post request to AuxBox for altering or creating new objects in the database based on the forms fields.

1.1.9 migrations

Django object relational mapping framework generates migration class files for mapping Django Models to the data base tables. All these auto-generated files are stored in the migration directory.

1.1.10 models

Models directory contains all the Django Models classes which are created by the developers. These classes represent the database tables and their attributes. Migrations are generated via these models.

1.1.11 services

The service directory is the core of AuxBox business layer. All the service classes which apply the requested logic(via view functions) are included in this directory.

1.1.12 static

Static directory contains css and js files which are used to render templates(HTML files). Static images like logos and album pictures are also stored in this directory.

1.1.13 templates

Templates directory contains every HTML file which are pages that are rendered on the client side browser.

1.1.14 tests

Tests directory contains a base file on the first level which is used to define test objects. Services directory under Test directory includes a class file for each unit test.

1.1.15 views

Views directory represents the core of AuxBox front end. All the view functions are included in their own file responsible for forwarding request information to other layers.

1.2 Data Layer

AuxBox data layer consists of three separate folders; dao, factory and daointerface. These are sets of classes that follow a traditional approach for implementing simple data access hierarchy. DaoFactory is the way data layer provides access to its data extraction methods to external classes and methods which exist outside of the data layer realm. DaoInterface folder represents abstraction over the actual data access methods where the definitions of actual methods take place. Dao is the main folder where all the data access classes and their method's implementations are present. Dao classes are responsible for querying the database according to the requested data set and filtering over the the dataset therefore allowing access to every table in database when needed.

Django database framework is used for the execution of the queries.

Overall, the main functionality of the data layer is to provide data according to the needs of the service layer.

1.3 Service Layer

Service layer is where all the logic is applied based on the queries we get from AuxBox/Spotify users including the authentication logic.

1.3.1 Authentication

AuxBox uses the Tekore api in order to handle the authentication process. Tekore provides high level methods which are easy to use to cope with the necessary token system which is set by Spotify Api itself. Once the users connect to our website, they are prompted a page where they need to use their Spotify account credentials to login and allow the AuxBox application to use their personal data related to their Spotify account. When the authentication process is done, users are allowed in AuxBox homepage and are able start using every aspect of the application.

1.3.2 Query Processing

AuxBox has a simple yet effective way of processing user requests. There exists a service class for each and every table in the database. These classes are responsible for taking the information from the presentation layer and applying the necessary logic that is needed to fulfill the queries demands. Service objects accomplish this task by their methods that are specifically designed to be composed together and communicate with the data layer to apply relatively complex logic therefore becoming the most important building block of AuxBox business layer. Service objects are created and used inside the view functions which represent the base of AuxBox presentation layer. Service methods interact with DaoFactory class in order to access data access objects which is simply a another layer for providing abstraction over the data layer

1.4 Presentation Layer

AuxBox presentation layer is fundamentally based off of the Django views system. The viewing process of every page starts with the "urls.py" file where all the urls that are available for users to access take place. Once a url is triggered by some client, Django makes an implicit call to view functions which all have their own files that are stored under the views directory. These functions are responsible for processing the request which was acquired from the triggered url and they are the very first step of every use case AuxBox has to provide. View functions operate by constructing and using the service objects. After confirming the successful execution of a particular query which is done by the service and data layer, view functions proceed to render and populate the related HTML page with the current information provided. They

1. ARCHITECTURE AND CLASS DESIGN

are also responsible for handling possible exceptions which might be caused by anything from false requests to database errors. HTML pages exist under the templates directory. AuxBox does not use any complex framework for the front end part other than the latest HTML technology and minimal JS scripting.

New Use Cases

This chapter explains the usage of the newly added use cases for the AuxBox application and their technical background.

2.1 User Comments

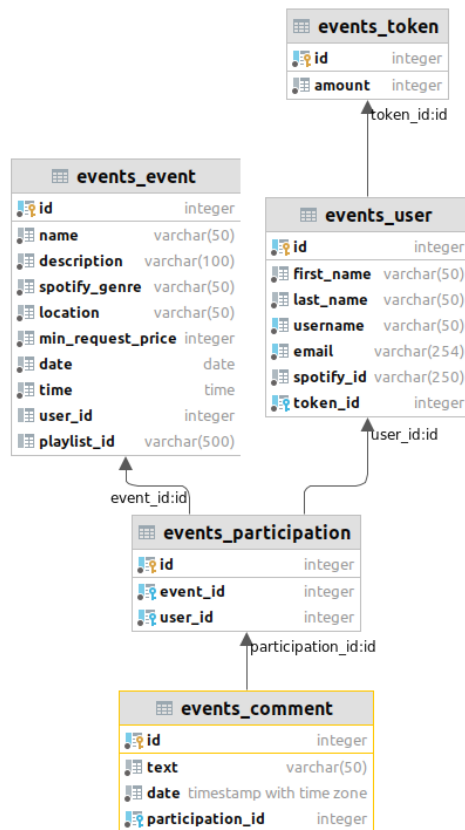
One of the most popular use cases of almost all social media platforms is undoubtedly the user comments. It allows people to express their feelings or ideas regarding whatever interests them on the platform. In the case of AuxBox, it will allow AuxBox/Spotify users to comment on the event which they joined and express their thoughts on it. Comments section is a quite simple but very crucial part of the AuxBox application.

2.1.1 Model Design

Comment model consists of three fields: `text`: The text of the comment. Maximum 200 characters. `date`: The date when the comments was made. `participation`: The participation object as Foreign key.

Participation has its own model with the fields `event id` and `user id` which is used to represent a unique relationship between an event and a user. Having this relationship field in comment model allows AuxBox to distinguish between comment objects and be aware of who it belongs to on which event.

2. NEW USE CASES



Comment Model

and its relations(screenshot from DataGrip)

2.1.2 Usage Specification

The use case of user comments starts at the presentation layer as soon as the url is triggered by the user. Comment button takes place on the event page. When the user triggers the button, view function `comment` is called with a parameter of type integer representing the id of the the event to be commented on. The view calls the participation service and checks whether user is a part of the event or not. If the user is in the joiner list of the event, a Django form is created and prompted to the user. Otherwise a warning message is formed and rendered on the same page indicating that the user should join the event in order to make a comment. In the case of correctly filled form, comment service is called and the user comment is registered in to the database. Another use case present in the comments system is to show all the comments of a particular event which is obviously vital for such functionality. There exists another button on the event page which triggers the show comments view function. The job of this function is simple. It calls the comment service and gets all the comment objects which have the id of

the event as a part of their participation foreign key and renders an HTML page to list them.

2.2 Event Likes

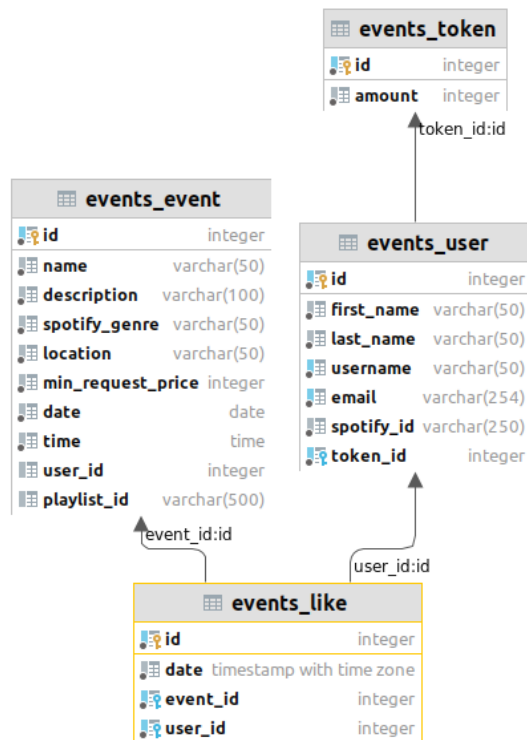
Arguably the most effective use case of any social media platform is the ability of the users to manifest their feelings towards any post with minimal effort, by just one click. Likes have been around for a while and they are not going anywhere. The convenience of uplifting any post you see is simply efficient and has great potential to provide the most simple and use full feedback to the developers and content creators which helps the platform to enhance. Thus, AuxBox implements likes so users can like any event that they like and have an affect on the bigger picture.

2.2.1 Model Design

The like model is relatively simple, it has three fields; date: Holds the date information regarding when the event was liked. user: The user as foreign key who liked the event. event: The event as foreign key which was liked by the user.

User and event are unique together and they are used to distinguish between like objects.

2. NEW USE CASES



Like Model and

its relations(Screenshot from DataGrip)

2.2.2 Usage Specification

The like button is placed on the event page. When the url is triggered, like view function is called and the existence of the specific like object is checked. If there exists a like object with the same event and user, the page is refreshed with a warning which informs the user that the event was already liked. Otherwise, the like object gets registered in the database and the like count is refreshed which also takes place on the event page.

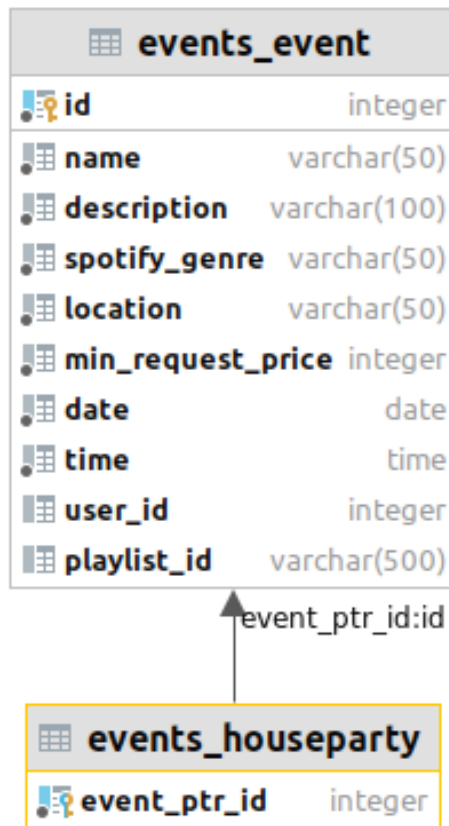
2.3 Create House Party

House parties are basically events with additional features. A regular event in AuxBox is assigned a single playlist which is chosen by the creator of the event. House parties introduce a collaborative way of handling playlists. As opposed to a regular event, a house party can have multiple playlists associated with it and the playlists can be added by any user who is a joiner of the house party. AuxBox will automatically determine the main playlist of the house party by

simply checking the amount of up votes of every playlist which is associated with the house party and choosing the one with the most up votes.

2.3.1 Design

House party model is the simplest class model in AuxBox, it only has one field which represent a pointer to the event table because House Party is implemented as a child class of Event model. AuxBox implements Django's Multi-Table inheritance model for representing House Party as a child class of Event. Django explains Multi-Table inheritance as follows: Each model in the hierarchy is a model all by itself. Each model corresponds to its own database table and can be queried and created individually. The inheritance relationship introduces links between the child model and each of its parents (via an automatically-created OneToOneField). It is entirely possible to add new fields to the House Party model in the future and they would be represented in the House Party model itself which is important for AuxBox Events structure since significant differences could cause unnecessary complications if implemented otherwise. The playlists and their up votes are handled through two different models called Playlist and Up Vote which are going to be explained on another section.



House Party model

and its relations(Screenshot from DATAGRIP)

2.3.2 Usage Specification

In order to create a house party event, the user needs to trigger the House Party button which is located in the homepage of AuxBox. The HouseParty button will trigger the create house party function and render an empty form, for the user to fill in the details of the House Party. After filling in the form the user triggers the create button right under the form and sends the POST request to AuxBox. Then the House Party object is created and registered in the database. The creation part works very similar to the Event creation. Usage of other features of House Party like adding and up voting playlists are going to be explained in the next sections.

2.4 Add Playlist To House Party

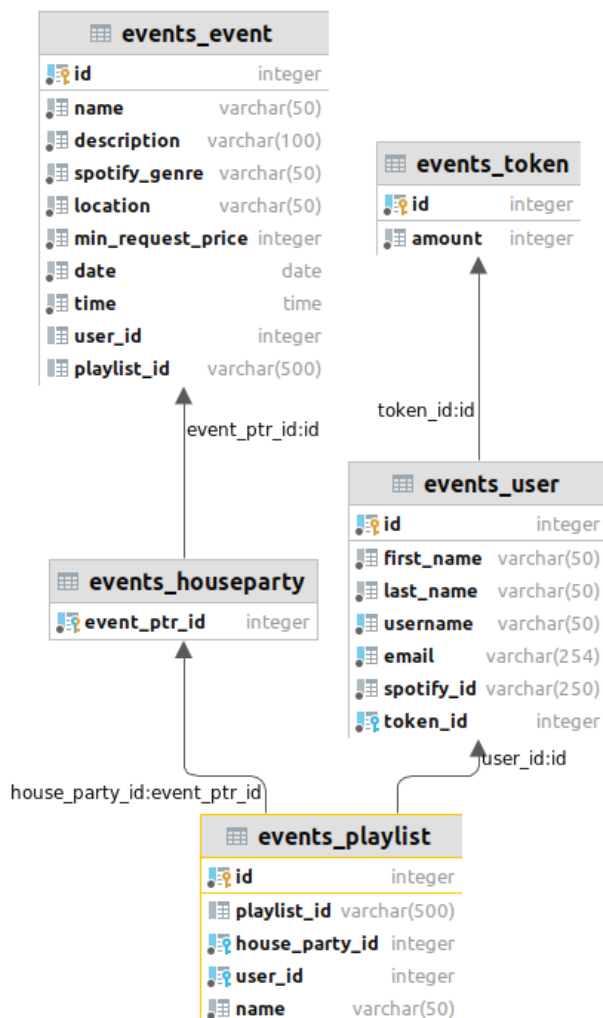
House Party playlists work in a collaborative way as opposed to regular Events. Unlike events, House Party can be associated with multiple playlists which can

be added by all of the joiners of the House Party. This use case represents the functionality of adding playlists to a House Party which everyone can see.

2.4.1 Design

Playlist model consists of four fields: playlist id; Holds the Spotify id of the playlist. house party id(Foreign key); Holds the Event/House Party instance id which the playlist is associated with. user id(Foreign key); Holds the id of the user who added the playlist. name; Holds the name of the playlist.

The user id and the house party id are the main foreign key fields which establish the association with the House Party model. The name field is automatically assigned by the Tekore API through *spotify,dprovidedbytheuser*.



Playlist model and its relations(Screenshot from Datagrip)

2.4.2 Usage Specification

The Add Playlist button is located on the House Party page. When the user triggers the button, add playlist view function is triggered which then creates a playlist form and prompts it for the user to fill on the browser. When the user submits the playlist form, the view function proceeds to create and register the playlist instance and saves it to the database by calling Playlist Service methods right after checking the validity of the playlist.

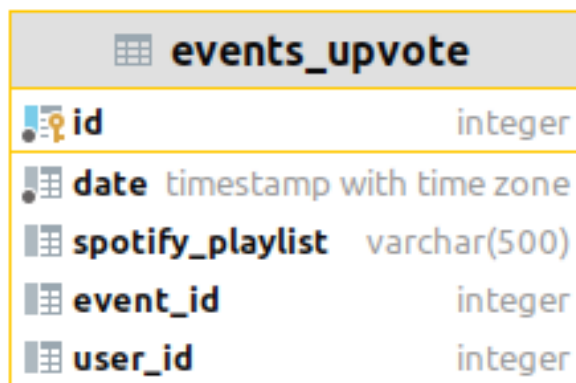
2.5 Up Vote Playlist

One of the defining features of House Party is the joiners ability to up vote the playlists which AuxBox uses to determine the main playlist of the House Party by counting the up votes. This use case aims to make the event more interactive and dynamic by allowing the users to participate in a simple and effective way.

2.5.1 Design

The Up Vote model has four fields: date: Holds the date which the up vote was registered. spotify playlist: Holds the Spotify id of the playlist. event id: Holds the id of the House Party which the playlist that got up voted was added to. user id: The id of the user who up voted the playlist.

The event id is used to filter the up vote objects by event and user id associates the user with the up vote. Both of these fields play the key role on establishing their relationship with the Up Vote model.



events_upvote	
id	integer
date	timestamp with time zone
spotify_playlist	varchar(500)
event_id	integer
user_id	integer

its relations(Screenshot from DataGrip

Up Vote model and

2.5.2 Usage Specification

In order to up vote a playlist, a user should go through a three step process. First, the show playlists button should be triggered which takes place in the House Party/Event page. This button triggers the show house party playlists view function which renders all the playlists associated with the current House party and presents it to the user. Every Playlist item on the page is represented as a card with two buttons attached to it. The up vote button sends a post request to AuxBox and the upvote gets registered in the system. The view playlist button triggers the playlist view function which renders a window that allows the users to go through the playlist and play any song on it. This functionality is accomplished by querying the Spotify API directly.

2.6 Edit Event

The ability of Event hosts to edit and update the event is introduced to AuxBox with this new use case. Hosts became able to change every field which defines the event like date, name, playlist etc after the creation of the Event.

2.6.1 Design

The Event model consists of nine fields which makes it the most complex model in AuxBox. name: Name of the event. description: Brief description of the Event's concept. spotify genre: The genre of the playlist to be played during the Event. location: The location of the Event. min request price: The minimum price a user has to pay in order to make a song request to the Event host. date: The date which the event will take place on. time: The time which the event will take place on. user id: ID of the user/host who created the event. playlist id: The Spotify ID of the Event's playlist.

events_event	
id	integer
name	varchar(50)
description	varchar(100)
spotify_genre	varchar(50)
location	varchar(50)
min_request_price	integer
date	date
time	time
user_id	integer
playlist_id	varchar(500)

Event model (Screen-

shot from DataGrip)

2.6.2 Usage Specification

In order to edit an Event, the user needs to view the even they created. In the Event page, only host users are presented with an edit event button. Edit event button triggers the edit event view function which loads the form using the current Event instance and renders it for the user with all fields filled in from the original information of the Event. At this stage, the user is allowed to change every field of the Event and submit it to update the information in the database. Django implicitly queries the database and updates the Event fields of the Event instance when the form is successfully submitted.

Testing

This chapter explains how and with what technologies and frameworks AuxBox implements unit testing.

3.1 Architecture

3.1.1 Overview

AuxBox uses Django's testing framework for unit testing. Django uses the standard python library module unittest. Django provides isolation over tests by introducing Test Case model and running each test inside a transaction. Python unittest defines tests with a class based approach. Classes can be used as test classes once they are passed the Test Case module as a parameter. Django's testing framework also comes with the functionality of creating a temporary mock database so the original database is not used during the testing session. By default, the mock database which was created gets destroyed when the execution of a particular test ends. In order to create objects in the mock database to use for testing, a set up method is used where objects and their fields are defined and saved.

3.1.2 Design

AuxBox stores all the test files under the tests directory. The base file under the tests directory is used to define all the mock objects which are going to be used for testing via the set up method under the Base Test Case class. The service directory under the tests directory consists of several classes/files which represent the test classes of several different use cases. These classes inherit the Base Test Class which allows them to use all the defined mock objects. These classes have methods for testing different outcomes of a particular use case by simple assertions that validate the result of the use case process.

Recommendation System

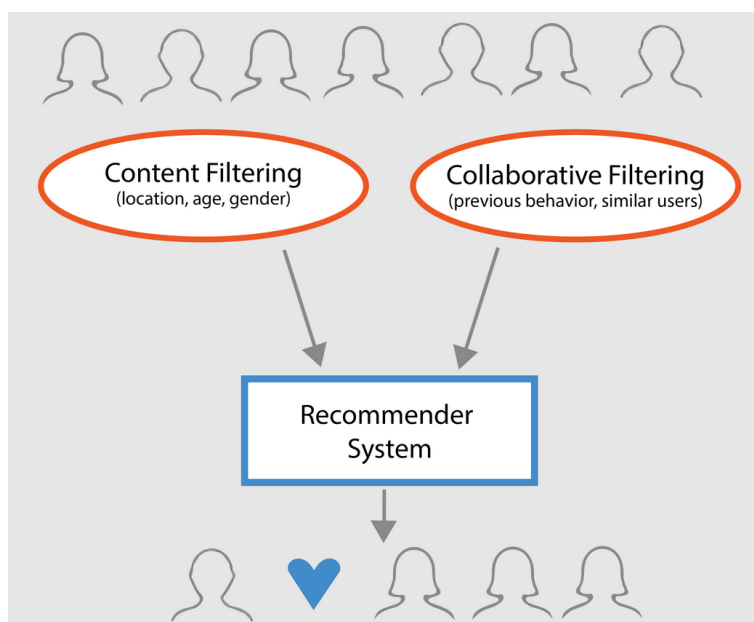
4.1 Abstract

Recommendation systems are widely used all over the world by most of the big and small enterprises in their applications in order to enhance the user experience by processing user data with the help of machine learning algorithms. These systems are essentially sub sets of Information Filtering systems and that is exactly what they are designed to do, filtering and analyzing user data in search of predicting a user's preference in terms of content on a particular context. Complex recommendation systems target building up on a specific part of user data and the relationships across thousands or even millions of users. The most successful companies like Netflix, YouTube, Facebook, Instagram and many others have successfully implemented at least one type of recommendation system in their web applications. They all have a way of presenting specific type of content for every user which might relate to their taste. Thus, AuxBox strives to build a recommendation system which will certainly improve user experience and help the application get bigger and better by constantly processing more data and producing meaningful results to recommend Events or maybe even playlists and songs to the users in the future. AuxBox database consists of a fair amount of tables and relatively simple relationships between them which most of the time involve the users themselves. These relationships are quite suitable as a core building block of an effective recommendation system.

4.2 Recommendation Methods

4.2.1 Abstract

There are a lot of ways to make a recommendation system work, but over the past 10 - 15 years, numerous methods have been used, tested and analyzed which allows us to generalize these methods in three different cate-



gories. Recommendation systems Diagram (SCREENSHOT FROM [1])

4.2.2 Simple Recommendation Methods

The most general and straight forward implementation of a Recommendation system focuses only on the history of popularity of the data to be presented to the end user. Simple recommendation methods analyze the popularity of a certain content and predict the likelihood of a users preference based on that since a type of content is more likely to attract users if it was liked or shared by a bigger amount of users compared to the average. This kind of approach is seen to be successful on simple data sets and simple use cases like listing the top rated movies on a single database by simply counting the stars/votes. Even though such methods mostly work fine to some extent, it fails to see the bigger picture where thousands of parameters come in to play which can be used to provide a much more accurate prediction. User similarity and specific user data like age, gender and location are other ignored aspects in the these simple recommendation methods. Only focusing on the general opinion of the user base is going to miss out on more intricate relationships among the users which our system could have build up on and define user groups to analyze user behavior in a much more extensive way. Thus, better recommendation systems were crafted which are explained in the following sections. For context, the below image shows a table of top rated movies and their ratings. Such output can be achieved by a weighted calculation of the ratings which considers the amount of ratings as well as the average rating of a movie.

	A	B	C	D	E
1	314	The Shawshank Redemption	8358	8.5	8.445869
2	834	The Godfather	6024	8.5	8.425439
3	10309	Dilwale Dulhania Le Jayenge	661	9.1	8.421453
4	12481	The Dark Knight	12269	8.3	8.265477
5	2843	Fight Club	9678	8.3	8.256385
6	292	Pulp Fiction	8670	8.3	8.251406
7	522	Schindler's List	4436	8.3	8.206639
8	23673	Whiplash	4376	8.3	8.205404
9	5481	Spirited Away	3968	8.3	8.196055
10	2211	Life Is Beautiful	3643	8.3	8.187171
11	1178	The Godfather: Part II	3418	8.3	8.180076
12	1152	One Flew Over the Cuckoo's Nest	3001	8.3	8.164256
13	351	Forrest Gump	8147	8.2	8.150272
14	1154	The Empire Strikes Back	5998	8.2	8.132919
15	1176	Psycho	2405	8.3	8.132715
16	18465	The Intouchables	5410	8.2	8.125837
17	40251	Your Name.	1030	8.5	8.112532
18	289	Leon: The Professional	4293	8.2	8.107234
19	3030	The Green Mile	4166	8.2	8.104511
20	1170	GoodFellas	3211	8.2	8.077459

4.2.3 Content Based Recommendation

Content based recommendation systems focus on content features/items associated with a user. By observing how a user reacts to features of some content and determining whether a user's taste corresponds to a feature in a positive way or not, we can predict the probability of a user enjoying another item which is similar in terms of features and that would lead to a recommendation as an output. Content based recommendation systems are capable of providing more accurate results due to the fact that they can use a huge amount of parameter scale therefore training a much more data rich model as well as correlations among them. An important aspect of content based systems is that they do not need other users' data in order to produce a recommendation for a user. Thus, it focuses on the history of the user's preferences with respect to the content and then proceeds to find other content which would be relevant for the user. In order to achieve this, the similarity measurements

4. RECOMMENDATION SYSTEM

are calculated through the content vector which includes the features of the content and users vector that includes his/her previous records about contents features.

The below table shows how multiple users and their ratings with regards to movie types(action, comedy) can be represented.

Movies	User 1	User 2	User 3	User 4	Action	Comedy
Item 1	1		4	5	Yes	No
Item 2	5	4	1	2	No	Yes
Item 3	4	4		3	Yes	Yes
Item 4	2	2	4	4	No	Yes

4.2.4 Collaborative Filtering

As opposed to content based recommendation systems, collaborative filtering method focuses on similarities among users rather than the features of an item/content. Users are mapped as feature vectors or some type of mapping of users to their preferences based on features. Therefore users and contents are both embedded in the same space. The collaborative method works by filtering users and creating subsets of users who has similar preferences. The items which the users liked are combined to form a list of recommendation candidates. Many different techniques can be used in order combine user preferences and create a list of recommendations. The below image shows a matrix representation of user vectors consisting of the ratings the users gave to items.

	Item 1	Item 2	Item 3	...	Item i	...	Item n
User 1	2	3	-		5		1
User 2	-	3	1		1		2
...							
User a	2	1					
...							
User k	5	1	-		1		-

4.2.5 Collaborative Recommendation System Types

4.2.6 Memory-Based Collaborative Filtering

Memory based filtering methods depend on memorizing tables/matrices which describe the relationship between a user and an item and how the user rated the item. There are two kinds.

4.2.7 User-User filtering

As the name suggests user-user filtering is based on the comparison of users with regards to their preferences on certain items and finding out similarities between them. A users liked items may be found suitable and be recommended to a another user if the system can detect enough similarities. It is important to note that the system should also handle biases among users. A user might be biased to give low ratings which is not related to whether the user is similar to other users or not. Thus, some normalization method is usually used to eliminate the bias.

4.2.8 Item-Item filtering

This method is purely item based. If a users preferred/liked item shows similarities to another item, that item is recommended to the user. The similarity between user vectors can be calculated via different measures like Cosine Similarity(cosine angle between vectors), Euclidian Distance(squared distance between two vectors).

4.2.9 Model-Based Collaborative Filtering

Instead of remembering the matrices, model based filtering tries to learn users behavior on items from the large matrix of user - item interactions via reduction and clustering algorithms. Machine learning models are used in order to predict a users rating on an arbitrary item. Multiple methods can be used to achieve this goal like clustering algorithms, matrix factorization and deep learning methods.

4.2.10 Clustering Algorithms

Clustering algorithms target the efficient categorization of the available data. These algorithms make use of creating groups out of a big data set and classifying those groups based on properties like density or frequency with respect to their similarities which leads to users being associated with certain data groups, therefore providing more precise predictions on what a user would prefer in his/her future interactions with new content. Thus, recommendation systems can benefit from the final product of the clustering algorithms

which define relationships between users and items based on the probability of a user being subscribed to a certain group.

4.2.11 Matrix factorization

Matrix factorization works by factorizing user-item relationship matrices in to smaller ones which can then be used to generate a more accurate interaction matrix. Factorization algorithms are similar to clustering algorithms on the surface but fundamentally different on a deeper practical level. These two algorithms both focus on some sort of classification of the provided data but the main objective of factorization is to simplify the data in order to come to a conclusion with regards to user-item similarities by factor analysis. Factor analysis is a statistical data reduction and analysis technique that strives to explain correlations among multiple outcomes as the result of one or more factors.

4.2.12 Deep Learning Methods

Deep learning methods usually outperform other traditional methods due to their ability to interact with and process non-linear complex data. This is a huge benefit since a lot of the data available all around the globe can only be represented as non-linear data structures like trees. Deep learning methods focus on higher level data analysis and are able to provide precise predictions due to a deeper user - item analysis capability. One of the most successful recommendation systems belongs to YouTube which makes use of deep learning methods in order to enhance user experience. It is fair to say that deep learning methods are the most advanced prediction and analysis tools and they still continue to get better and better over time.

4.3 Integration With AuxBox

AuxBox database tables contain a fair amount of information which can be used to define relationships between users and items either via collaborative filtering or content based filtering. Precisely the Like, Up Vote, Playlist together with Event and User tables represent meaningful connections within users and items(events , playlists). Moreover, factorization and clustering can be applied on the data sets created out of the relationships within the tables. These algorithms usually require a good amount of data in order to make precise predictions for a recommendation system. To achieve that in the early stages of the application being deployed, synthetic data can be generated using the small amount of real data available. Various legit data sources are available on the internet as well which contain information about general relationships between music and users that can be valuable for the AuxBox recommendation system.

Conclusion

The overall objective of this thesis was to extend the AuxBox application in order to make it a much more interactive and user friendly environment and make research on the possible ways of implementing a recommendation system. Overall, via all the newly introduced interactive use cases which allow users to be more in control and interact with the events they create or join while also providing feedback both for users and developers by their likes, comments and added playlists, AuxBox certainly improved overall user experience and productivity.

The research on the possible ways and methodologies of implementing a recommendation system provides a road map on how to prepare and take the first steps so that AuxBox and AuxBox users can benefit from accurate recommendations. The prerequisites of a database structure which would compliment the recommendation system became much more clear. The additional database tables provide a better overview on how to proceed with the data processing for use with user recommendations since tables; like, playlist and comment allow us to visualize and analyze the possible correlations between users , events and playlists more clearly which truly represents the core of every recommendation system.

AuxBox is built as a quite scalable and extendable application but there are still important design decisions to be made before it gets too complex for that to be done. More detailed explanation on possible design improvements are explained in the next chapter.

Future Work

The most ambitious and self evident future work for AuxBox to be done is no doubt a recommendation system. The research on the issue in this thesis sheds light on the possibilities of implementing user recommendations and it can certainly be implemented in the future but very important design decisions and a fair amount of rework should most certainly be done in order for AuxBox to reach its full potential. For example, even though the Django view functions are very simple and easy to use, they come with their restrictions because they are not complex structures by any means which almost completely eliminates the possibility of inheritance among views which then leads to repeated and unnecessary code. Django provides class based views which could be a much more logical option for a more complex system as AuxBox thrives to be.

One of, if not the most crucial aspect of a social web application is by no doubt, the user interface. AuxBox does not make use of a front end framework like Angular or React which is a huge drawback because the capabilities and effects of these types of frameworks are huge and used all over the world to create the best looking user interfaces. The integration of a front end focused framework has to be a priority for AuxBox in order to attract music lovers attention.

AuxBox application is currently not deployed on any cloud service. The future deployment plan is to use Microsoft Azure platform for deployment. Azure is a cloud platform which allows its users to build , run and manage their applications with a large set of framework choices. Django apps which use a Postgre SQL database are fully supported as well and its relatively easy to set up the configuration using Azure command line interface.

Lastly , the factory design pattern which is used all throughout the application and does not seem to fit in perfectly with the overall design. It can be altered to be more beneficial or changed to a completely different one which is not necessarily too difficult to accomplish in the current state of the application.

mybibliographyfile.bib [DjangoProject] [6] [4] [1] [5] [2] [3]

Bibliography

- [1] “Data Camp”. In: (). URL: <https://www.datacamp.com/>.
- [2] “Developers Google”. In: (). URL: <https://developers.google.com/>.
- [3] “Medium”. In: (). URL: <https://medium.com/>.
- [4] “Real Python”. In: (). URL: <https://realpython.com/>.
- [5] “Towards Science”. In: (). URL: <https://towardsdatascience.com/>.
- [6] “Wikipedia”. In: (). URL: <https://en.wikipedia.org/>.

APPENDIX **A**
