PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Moodsic

Aplicação de recomendação musical baseada em análise de sentimentos

Guilherme de Moraes Vassallo

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Sistemas de Informação

Rio de janeiro, novembro de 2023

Guilherme de Moraes Vassallo

Moodsic

Aplicação de recomendação musical baseada em análise de sentimentos

Monografia de Projeto Final, apresentado ao Departamento de Informática da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Ivan Mathias Filho

Rio de janeiro,
Novembro de 2023

“Música é vida interior, e quem tem vida interior jamais padecerá de solidão.”

-Artur da Távola

Agradecimentos

Ao meu pai, minha mãe e outros membros da minha família, por tudo que já fizeram e fazem por mim.

Ao orientador do meu projeto, Ivan, pelo apoio, aconselhamento, encorajamento, e bons papos sobre música e vida no geral.

A todo o corpo docente da PUC-Rio e aos amigos que fiz enquanto estudei na instituição.

Resumo

De Moraes Vassallo, Guilherme. Mathias Filho, Ivan. **Moodsic: Aplicação de recomendação musical baseada em análise de sentimentos.** Rio de Janeiro, 2023. 48p. Monografia de Projeto Final II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho apresenta o processo de estudo, prototipação e desenvolvimento de uma aplicação Web chamada Moodsic, que tem como objetivo realizar a análise sentimental de um relato pessoal digitado pelo usuário e utilizar o sentimento detectado para buscar uma *playlist* condizente (ou oposta) a esse sentimento no Spotify, também dando aos usuários a possibilidade de salvamento dessas buscas. Ela foi desenvolvida por meio do *framework* Django e da integração com API's do Spotify e do site Hugging Face. A solução explora a conexão da música com os sentimentos humanos, e explicita o avanço da área de *machine learning*. O produto do trabalho foi uma aplicação simples, porém intuitiva, funcional e com muito potencial.

Palavras-chave

Django. Spotify. Web. Música. Análise de Sentimentos.

Abstract

De Moraes Vassallo, Guilherme. Mathias Filho, Ivan. **Moodsic: Music recommendation application based on sentiment analysis.** Rio de Janeiro, 2023. 48p. Monografia de Final de Projeto Final II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This work presents the process of studying, prototyping and developing a Web application called Moodsic, which aims to perform the sentiment analysis of a personal report typed by the user and use the detected sentiment to search for a playlist consistent (or opposite) to it, while also giving users the possibility of saving those searches. The application was developed with the Django framework and by integrating services provided by Spotify and Hugging Face via API. The solution explores the connection between music and human sentiments, and makes advancements in the machine-learning field explicit. The product of this work was an application that is simple, but intuitive, functional and full of potential.

Keywords

Django. Spotify. Web. Music. Sentiment Analysis.

Sumário

1 - Introdução 9
2 - Situação Atual 11
3 - Objetivos 12
3.1 – *Login*, *Logout* e Registro de usuários 12
3.2 – Escrita de texto e análise sentimental 12
3.3 – Busca no Spotify 13
3.4 – Buscas opostas e outras opções 13
3.5 – Pesquisar novamente ou salvar busca e seus resultados 13
3.6 – Histórico de buscas 13
4 - Atividades Realizadas 14
4.1 – Estudos preliminares 14
4.2 – Estudos conceituais e de tecnologia 14
4.3 – Protótipos de baixa fidelidade 17
4.4 – Diagrama de classes 21
4.5 – Diagrama de fluxo de telas 22
4.6 – Modelo de casos de uso 23
4.6 – Tarefas 26
4.7 – Cronogramas 27
5 - Projeto e especificação do sistema 30
5.1 – A arquitetura do Django 30
5.2 – O Componente Model 31
5.3 – O Componente View 32
5.3 – O Componente Template 34
5.4 – Outros arquivos e pastas notáveis 34
5.5 – Utilização do Moodsic 35
6 – Comentários sobre a implementação 43
6.1 – Desafios 43
6.2 – Possíveis melhorias 44
7 - Considerações Finais 46
8 – Referências Bibliográficas 48

Lista de figuras

Figura 1: Fluxo Authorization Code Flow da API do Spotify. 16
Figura 2: Fluxo Client Credentials Flow da API do Spotify. 16
Figura 3: Protótipo da homepage 18
Figura 4: Protótipo da tela de resultados 19
Figura 5: Protótipo da tela de linha do tempo (Timeline) 20
Figura 6: Diagrama de Classes UML 22
Figura 7: Diagrama de Fluxo de Telas 23
Figura 8: Diagrama de Casos de Uso 24
Figura 9: Estrutura de pastas do Moodsic no VSCode 30
Figura 10: Homepage (tela de pesquisa) 35
Figura 11: Tela de resultados (sem login) 36
Figura 12: Tela de login 37
Figura 13: Tela de registro 37
Figura 14: Tela de resultados (com login) 38
Figura 15: Tela de timeline 39
Figura 16: Tela de detalhes de um Moodsic 40
Figura 17: Tela de exclusão de um Moodsic 41
Figura 18: Dashboard de administrador 42
Figura 19: Acessando informações de um Moodsic (admin) 42

Lista de tabelas

Tabela 1: Descrição de caso de uso 25
Tabela 2: Cronograma de tarefas planejadas para o projeto 27
Tabela 3: Cronograma de tarefas executadas no projeto 28

1 - Introdução

Nos dias de hoje escutar música está mais fácil e acessível do que nunca. Com o avanço da internet e de serviços de *streaming*, encontrar músicas está a poucos cliques de distância. Apesar disso, ainda não são comuns soluções que explorem a relação da música com nossos sentimentos e vivências do mundo real. Uma boa música pode tornar um dia complicado um pouco mais agradável, nos relembrar de coisas boas ou marcantes, ajudar na concentração, expandir horizontes, ajudar a extravasar a raiva e a frustração, e muito mais. Não é à toa que existem áreas como a Musicoterapia, que utiliza a música e seus atributos para tratar diversas condições mentais de pacientes.

Por outro lado, a área de Inteligência Artificial (IA) tem evoluído consideravelmente nos últimos tempos, estando mais poderosa, performática e acessível do que nunca. Temos sistemas de IA e algoritmos de *Machine Learning* trabalhando por baixo dos panos em muitos dos serviços de internet mais utilizados mundialmente (SANTO DIGITAL). Eles facilitam muito a nossa vida, tornando resultados de pesquisas mais relevantes, filtrando conteúdo indesejado e otimizando o tempo de entrega desse conteúdo, dentre outras aplicações. Sem contar o advento de Chatbots avançados, como ChatGPT (OPENAI) e Google Bard (GOOGLE), e a disponibilidade de material de aprendizado amplo e até bibliotecas pré-treinadas de uso descomplicado.

Neste cenário, e sabendo-se do potencial que a música e a Inteligência Artificial possuem, foi se criando uma ideia que porventura culminaria na realização deste projeto. E se fosse possível criar uma solução que explorasse essa relação sentimental da música com as pessoas e, portanto, fosse possível recomendar músicas para um usuário que se encaixassem perfeitamente em como ele está se sentindo ou no que ele viveu naquele dia?

Com isso, foi idealizado o Moodsic, cujo desenvolvimento foi o objetivo deste projeto. O Moodsic é uma aplicação que tem como objetivo recomendar uma *playlist* ou uma música que seja perfeita para o que um usuário está sentindo. Para isso, ele realiza uma análise emocional de um texto curto digitado pelo usuário, em que ele deve contar brevemente como se sente ou como foi seu dia. Depois, ele usa esta análise para fazer uma busca por *playlists* no Spotify (SPOTIFY), a famosa plataforma de streaming de músicas. O usuário também pode salvar e consultar estas buscas para referência futura, acompanhando o progresso de seu humor ao longo do tempo e relembrando as músicas que foram trilha sonora destes sentimentos.

O Moodsic foi desenvolvido como um website, pois ao longo da minha graduação tive bastante contato com desenvolvimento Web. Dessa forma, seria mais fácil desenvolver e testar a solução em vários dispositivos, sem necessidade demasiada de pesquisa ou conhecimentos externos. Além disso, desenvolvê-lo para a melhor plataforma e do jeito mais ideal possível, do ponto de vista de distribuição e usabilidade, não é o foco no momento. Foi definido que o mais interessante agora é provar o conceito por trás da aplicação, por se tratar de uma ideia relativamente nova. Também é importante mencionar que o Moodsic foi desenvolvido com interface em língua inglesa e com suporte para esta língua somente, por conta de uma maior disponibilidade de ferramentas de análise textual para essa linguagem.

Para seu desenvolvimento, foi utilizada a IDE Visual Studio Code (MICROSOFT), por conta de minha familiaridade prévia, boa performance e facilidade de configuração. A principal tecnologia escolhida foi o framework Django (DJANGO S.F.), também por conta de minha familiaridade e por ter diversas ferramentas e recursos, que facilitam o desenvolvimento Web, tanto no *front-end* quanto no *back-end*. O Django trabalha com as linguagens de marcação e estilização clássicas da internet, HTML5 e CSS, e utiliza a linguagem de programação Python para realizar a comunicação dos *templates* HTML com o *back-end*, por meio de requisições Web, e para configurações gerais do projeto. Também foi utilizada a linguagem Javascript em alguns momentos, com objetivo de programar componentes dinâmicos e reativos nas páginas. Já no *back-end*, a tecnologia escolhida foi o SQLite, pois é o gerenciador de banco de dados que o Django usa por padrão, além de ser simples de ser configurado e utilizado. Como a aplicação não tem tanta carga de usuários ou requisições, além de possuir uma base de dados simples, ele se demonstrou suficiente para o projeto.

Foram utilizadas também duas bibliotecas que intermedeiam chamadas à duas API's externas. Uma delas é a Transformers (HUGGING FACE), uma biblioteca Python do site Hugging Face, que permite a utilização de modelos de Machine Learning pré-treinados, pela sua comunidade de forma rápida e eficiente. O Modelo utilizado foi o "distilbert-base-uncased-emotion" (SAVANI), que analisa seis sentimentos diferentes em um texto em inglês: amor, alegria, tristeza, raiva, surpresa e medo. A outra biblioteca empregada é a SpotiPy (LAMERE, 2014), uma biblioteca Python que intermedeia chamadas à API oficial do Spotify assim como a autenticação do usuário.

Algumas bibliotecas secundárias também foram inclusas. Uma delas é a biblioteca Javascript CanvasJS (FENOPIX INC.), empregada para construir

gráficos reativos e animados dentro da aplicação. A outra é a Dotenv (KUMAR), uma biblioteca Python para gerenciar variáveis de ambiente. E, finalmente, foi utilizada a biblioteca *front-end* Bootstrap (BOOTSTRAP TEAM) para melhorar o visual e a responsividade do site.

O desenvolvimento do Moodsic se demonstra relevante para o curso de Sistemas de Informação por trabalhar diversos conhecimentos e conceitos apresentados durante o curso, dentre eles: Programação Orientada a Objetos, Programação para a Web (para produzir o código fonte), Modelagem de Software (para confecção de diagramas) e Engenharia de Requisitos (para descrição de casos de uso). Também trabalha a integração com API's, algo que é amplamente utilizado e requisitado em cargos de tecnologia atualmente (GR1D, 2022).

2 - Situação Atual

A existência de aplicativos similares ao Moodsic é escassa, por se tratar de uma ideia relativamente nova e bem específica. Porém, foram encontradas algumas soluções que possuem semelhanças com a minha proposta:

- Taranify (TARANIFY.APP) - Site que tenta recomendar playlists baseadas no sentimento atual do usuário. Para isso, ele coloca várias cores na tela e pede para o usuário selecionar, utilizando a ordem que elas são selecionadas para assim inferir o sentimento atual do usuário, e assim recomendar a *playlist*. O conceito é interessante, porém um pouco nebuloso e não muito personalizável. Ele afirma utilizar uma metodologia conhecida como o Teste de Cores de Lüscher. Nos testes realizados, no entanto, a aplicação recomendou muitas músicas aleatórias e não muito condizentes com meus sentimentos.

- Tunes For Tales (TANEJA) – Esse site se assemelha mais à concepção original do Moodsic. Ele recebe um texto do usuário contando a sua história ou a história de um personagem, e tenta recomendar músicas que seriam trilhas sonoras perfeitas para esta história. Ele é intuitivo e natural de usar, além de dar a opção de diversas músicas. De novo, peca bastante na escolha de músicas, o que torna claro que a recomendação de boas músicas é um desafio e um fator decisivo para a aceitação da aplicação. Ele também não permite guardar resultados prévios.

- O próprio mecanismo de busca do Spotify (SPOTIFY) – É fácil e rápido de pesquisar, tanto na aplicação Mobile quanto na Desktop, bastando apenas digitar os termos desejados na barra de pesquisa e depois escolher entre mostrar músicas, álbuns, *playlists* e etc. com os termos pesquisados. Funciona bem como

uma busca “padrão”, mas não é muito natural de usar. Por exemplo: se um usuário pesquisar “*I feel good*”, ele terá, à primeira vista, músicas com esse título e playlists com músicas parecidas com aquela famosa de James Brown (SEXTON, 2022), não necessariamente músicas que combinam com “alegria”, sendo o resultado de “*feel good*” ou “*good vibes*” mais adequado. Sem contar que algumas pessoas têm dificuldades de descrever emoções diretamente, o que dá valor à proposta de análise sentimental ao invés de só se analisar palavras chave, como o Spotify faz. Por fim, apenas os resultados de buscas prévias são salvos, não os termos utilizados.

Analisando-se todas essas soluções, pode-se notar que nenhuma delas possui o funcionamento desejado e nem todas as funcionalidades pretendidas para este projeto. Mais precisamente, as soluções não oferecem análise sentimental da busca do usuário (ou deixam muito a desejar), e não permitem o salvamento das pesquisas ou acompanhamento de evolução do humor, três coisas que o Moodsic pretende oferecer.

3 - Objetivos

Este projeto visa o desenvolvimento de uma aplicação Web, que possibilite aos usuários procurarem músicas que condizem com o seu estado de humor, por meio de um relato pessoal curto em forma de texto. Ele também deve permitir o salvamento dessas consultas e o acompanhamento da progressão humorística. Para isso, o aplicativo deve oferecer as seguintes funcionalidades:

3.1 – *Login*, *Logout* e Registro de usuários

Deve ser possível que cada usuário crie contas pessoais no site, preenchendo um formulário onde ele informa seu nome de usuário e sua senha, repetida duas vezes. Posteriormente, o usuário poderá acessar o site usando seu nome de usuário e sua senha registrada.

3.2 – Escrita de texto e análise sentimental

O site deve permitir que o usuário escreva um texto onde ele relate como está se sentindo ou como foi seu dia. Isso será feito com uma caixa de texto que ficará disponível para o usuário logo na primeira página do site (*homepage*). Depois, deve-se enviar esse texto para algum mecanismo de análise emocional, para que ele retorne as emoções detectadas no texto e a principal delas.

3.3 – Busca no Spotify

Após a análise emocional, a aplicação irá usar termos condizentes com a principal emoção detectada no texto para realizar uma busca por *playlists* no Spotify. Após a busca, deve-se selecionar uma *playlist* aleatória de todas as encontradas e extrair algumas de suas informações, como título, descrição, dono, imagem e o seu link, para que o usuário possa escutá-la. Finalmente, é preciso mostrar essas informações para o usuário.

3.4 – Buscas opostas e outras opções

É desejado que, antes de fazer a pesquisa, o usuário tenha algumas opções para customizá-la um pouco mais. Entre elas, a opção de procurar músicas opostas ao sentimento atual (ex.: músicas alegres para quando se sentir triste), a opção de escolher entre procurar por uma música ou uma *playlist*, ou de procurar um gênero específico de músicas, que poderia até ser adicionado como favorito pelo usuário, pois nem sempre o usuário é totalmente eclético.

3.5 – Pesquisar novamente ou salvar busca e seus resultados

Após a mostra dos resultados, o usuário deve ter a opção de ou realizar a busca novamente, caso não goste do que foi retornado, ou salvar a busca realizada. Isso inclui o salvamento de dados da busca, como o texto digitado, a emoção detectada, a data e horário em que a pesquisa foi feita, entre outros. Também envolve o salvamento de informações da *playlist* encontrada.

3.6 – Histórico de buscas

O usuário deve ter acesso às suas buscas salvas, que serão chamadas de “Moodsics”, podendo vê-las ordenadas, da mais recente até a mais antiga, e um resumo das informações de cada uma. O usuário também deve poder clicar sobre elas para ver mais detalhes, ou excluí-las. Nessa seção da aplicação, seria interessante também oferecer uma visualização das emoções detectadas ao longo do tempo por meio de um gráfico, assim como uma barra de pesquisa e/ou opções de filtro.

4 - Atividades Realizadas

4.1 – Estudos preliminares

O ambiente de desenvolvimento a ser utilizado não necessitou de muitos estudos de antemão pois, como já foi descrito anteriormente, tanto o Visual Studio Code como o *framework* Django eram relativamente familiares a mim. Apenas foi necessário relembrar alguns comandos de configuração e alguns detalhes da arquitetura padrão do Django, para isso foi utilizada a documentação oficial do *framework*.

O controle de versão do projeto foi feito por meio do sistema de versionamento Git e da plataforma GitHub, que já foram utilizados previamente por mim em diversos projetos. O repositório do projeto está disponível para acesso (VASSALLO).

4.2 – Estudos conceituais e de tecnologia

Para realizar a análise emocional do texto, foi necessário, antes de tudo, um estudo sobre processamento de linguagem natural e *Machine Learning*. A priori, a aplicação funcionaria com o português brasileiro e a análise sentimental do texto seria feita por meio do treinamento de um modelo próprio, com a técnica Naive Bayes (IBM CORP.). A técnica, resumidamente, se baseia no Teorema de Bayes e consiste em “alimentar” um algoritmo com frases já pré-classificadas, para que ele calcule a probabilidade de cada palavra-chave aparecer em cada uma das classes pré-definidas (no caso, sentimentos), assim gerando uma “tabela de probabilidades” para classificar frases futuras. Foi utilizado o ChatGPT para a geração de cerca de 570 frases de teste (pouco mais de 90 para cada emoção) e o modelo foi treinado por meio da biblioteca Python Scikit-learn (SCIKIT-LEARN TEAM) em um *notebook* do Google Colab (VASSALLO). Depois da limpeza dos textos (ex.: retirada de acentos e palavras sem significado) e do treinamento, o modelo demonstrava uma boa acurácia teórica (cerca de 89%), mas testes práticos revelaram muitos erros graves na análise. Após muitas tentativas frustradas de ajustar o treinamento, eu pude concluir que fazer um modelo de análise de sentimentos próprio seria uma tarefa bem mais árdua do que inicialmente estimado. Isso foi corroborado pela descoberta posterior na internet de poucas bibliotecas de análise sentimentais convincentes em português, sendo em sua maioria assunto de teses ou artigos científicos dedicados inteiramente à esta área da computação (SOUSA, 2016).

Portanto, eu decidi que, para que não fosse gasto tempo e esforço em demasia nesta parte do desenvolvimento, e tendo em mente que se trata de um sistema de informação, e não de uma pesquisa científica aprofundada, seria utilizado um modelo pré-treinado. Foi aí que descobri a biblioteca Transformers e o site Hugging Face, que possuía inúmeros modelos de análise sentimental de texto, gratuitos e disponíveis para uso, com quantidades muito maiores de dados, maior acurácia e boa performance. Com base nessa pesquisa também foi tomada a decisão do sistema ser em inglês, pois não foram encontrados modelos no Hugging Face em português brasileiro que analisassem múltiplas emoções em um texto (apenas sentimento positivo, negativo e/ou neutro). O aprendizado da biblioteca Transformers foi bem rápido, com instalação descomplicada via terminal, e um código de exemplo pronto na própria página do modelo escolhido (SAVANI).

Depois, foi necessária uma pesquisa sobre a API do Spotify e as diferentes maneiras de se trabalhar com ela. Foi decidido que a biblioteca Python SpotiPy seria a melhor delas, por ser a linguagem que o Django usa por padrão para diversos propósitos e por ela simplificar diversos comandos que seriam mais complexos sem a sua utilização.

Também pude aprender que o Spotify possui dois tipos de fluxo de autenticação: o Client Credentials Flow, exibido na Figura 2, e o Authorization Code Flow, exibido na Figura 1. No primeiro é apenas necessário que o desenvolvedor insira seu ID e chave secreta, que podem ser adquiridos no site Spotify For Developers (SPOTIFY) ao se registrar a aplicação a ser desenvolvida. Já no segundo, além dessas informações, é necessário que o usuário final seja redirecionado para uma página de *login* do Spotify e esteja “logado” em sua conta ao se realizarem as requisições. O primeiro fluxo é mais rápido e simples, com a desvantagem de não possuir acesso às informações pessoais do usuário. Como essa funcionalidade não era necessária para o funcionamento pleno do Moodsic, eu optei pelo uso do Client Credentials Flow.

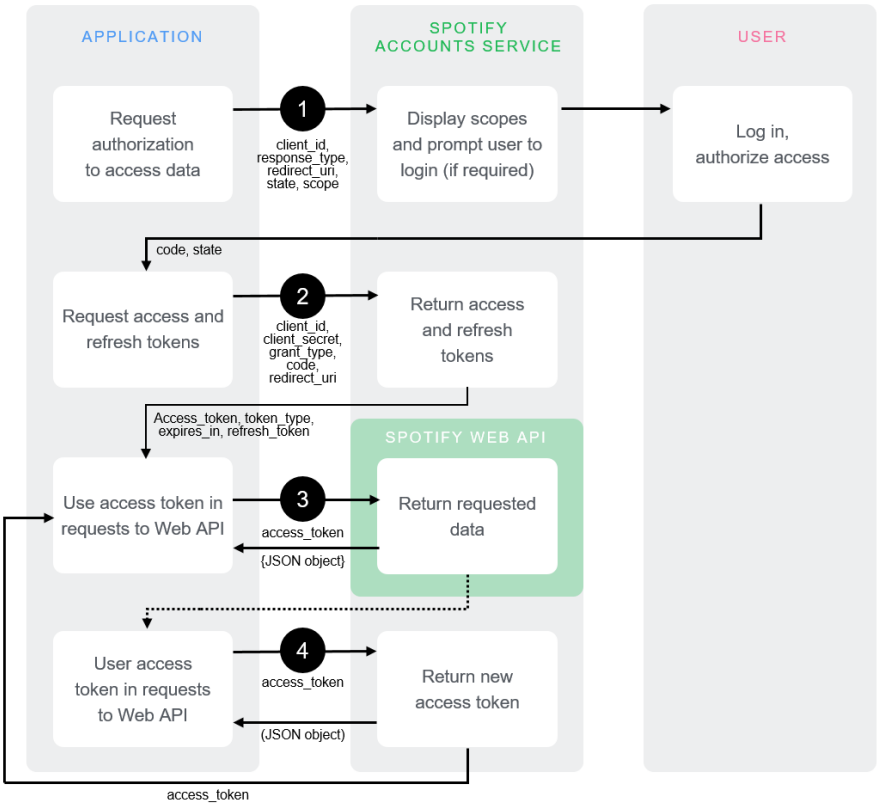

Figura 1: Diagrama mostrando o fluxo Authorization Code Flow da API do Spotify. Fonte: Spotify For Developers (SPOTIFY)

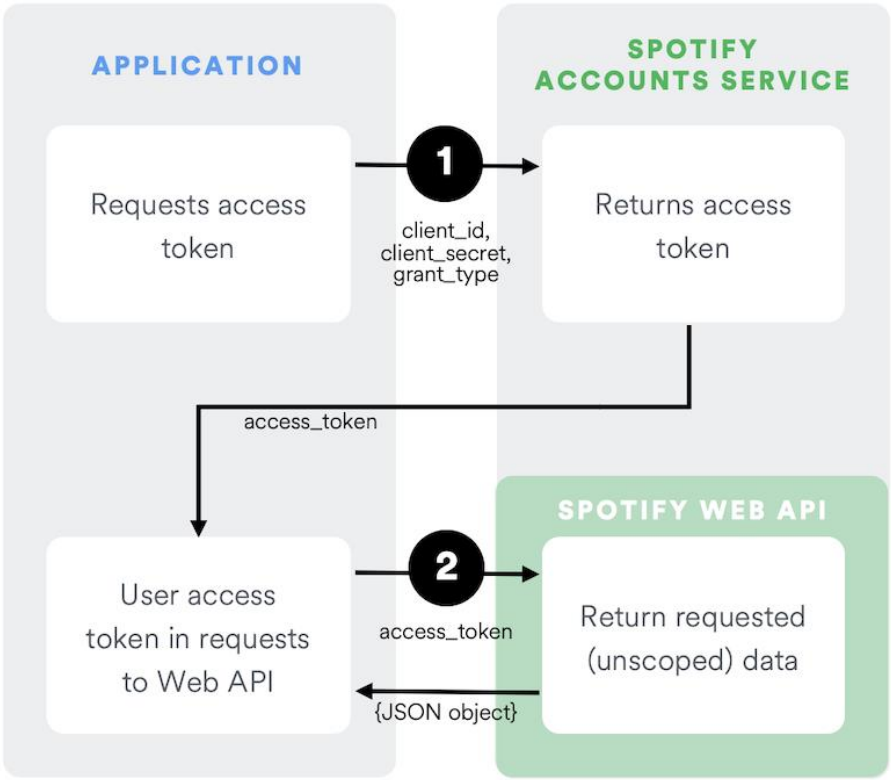

Figura 2: Diagrama mostrando o fluxo Client Credentials Flow da API do Spotify. Fonte: Spotify For Developers (SPOTIFY)

Com a integração do Spotify, também julguei interessante aprender mais sobre variáveis de ambiente. Por meio delas, pode-se “isolar” dados de configuração importantes de um projeto, de forma que alguém que tenha acesso ao código fonte não as veja. Esse conceito foi importante no Moodsic pois foi possível esconder o ID e a chave secreta do Spotify de modo que essas informações não ficassem “*hard-coded*” (expostas no código), o que não seria bom do ponto de vista de privacidade. Isso pôde ser feito de forma bem fácil e direta por meio da biblioteca Dotenv (KUMAR), do Python, e da inclusão de um arquivo “.env” no projeto que contém as duas variáveis sensíveis e não precisa ser incluído no controle de versão. Tudo isso é bem explicado na própria documentação da biblioteca.

Finalmente, foi necessário aprender a biblioteca CanvasJS (FENOPIX INC.) para a confecção de gráficos informativos e reativos com Javascript. O aprendizado foi bem simples pois já haviam exemplos prontos para uso de diversos tipos de gráficos no site da biblioteca.

4.3 – Protótipos de baixa fidelidade

Para se ter uma ideia inicial de como seria a interface do Moodsic, foram criados, com as ferramentas de desenho do Google Docs, três desenhos bem simples das três telas principais (*homepage*, resultados e *timeline*), que podem ser vistos nas Figuras 3, 4 e 5.

Os desenhos estão em formato vertical, com elementos de interface típicos de smartphones, pois, nesse estágio inicial, ainda se considerava a possibilidade de desenvolver o Moodsic como um aplicativo para celulares. De qualquer forma, foi de algum valor realizar os desenhos nesse formato, pois eles ajudaram a visualizar como poderia ser a parte responsiva do site (adaptá-lo para diversos formatos de tela, especialmente celulares e tablets). Também vale mencionar que estão em português brasileiro, para melhor entendimento e por conta de decisão de fazê-lo em inglês ter sido tomada só posteriormente às suas criações.

Figura 3: Protótipo da homepage

Na página inicial, ou *homepage*, da aplicação (Figura 3) é onde o usuário digitará e confirmará o seu relato pessoal, que posteriormente será analisado e servirá de base para a busca por uma *playlist*. Desde o início do projeto, era desejado que o usuário pudesse ter essa possibilidade instantaneamente, na primeira tela e sem nenhum empecilho. Isso foi mantido, assim como a caixa de texto, o texto de boas-vindas e o botão. As diferenças são que o botão foi centralizado e que o menu “hambúrguer” na parte superior esquerda foi substituído pelo header do site. A ideia de haver buscas opostas ao sentimento atual veio depois. Por isso, o *checkbox* responsável pela seleção da funcionalidade não aparece no desenho, mas sim na versão finalizada da homepage, retratada na Figura 9.

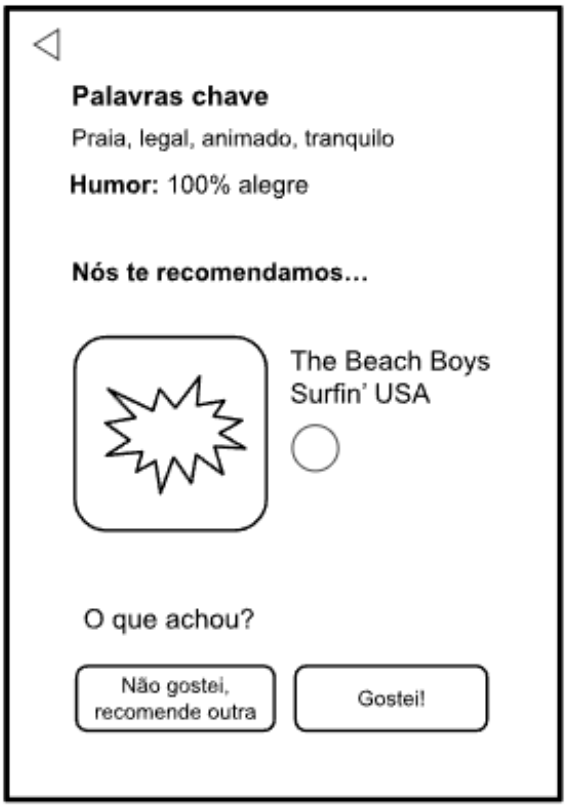

Figura 4: Protótipo da tela de resultados

Na tela de resultados (Figura 4), após a análise emocional do texto digitado pelo usuário e da busca no Spotify, são mostrados os resultados da análise e da pesquisa, que incluem informações como o sentimento encontrado, a *playlist*, seu título, seu link do Spotify (representado pelo círculo) e sua imagem. Também existem, na parte inferior da tela, botões de feedback positivo ou negativo.

Nesse desenho, a ideia ainda era a de procurar uma música somente, ao invés de uma playlist. Porém, assim que descobri mais sobre como a busca do Spotify funciona, e considerando que uma playlist dá mais horas de músicas similares ao usuário, optei por pesquisar por uma playlist na solução desenvolvida. Também aparecem certas *keywords* extraídas do texto digitado, pois, além da análise emocional, seriam extraídas algumas palavras do texto para tornar a pesquisa mais efetiva e específica (Ex.: O usuário está feliz na praia e, ao invés de retornar só músicas felizes, retornar músicas felizes sobre praia.). Infelizmente, por falta de tempo e por preocupações a respeito de sua eficácia na prática, essa funcionalidade não foi adicionada, ficando como algo desejável que poderia ser interessante no futuro.

No mais, essa tela na versão desenvolvida ficou bem parecida em termos de layout, com poucas diferenças. Uma delas é a presença de informações de resultado mais completas (ex.: o texto digitado, análise sentimental detalhada, descrição da *playlist*, dono da *playlist*, se a *playlist* é oposta ao sentimento detectado ou não). Além disso, na versão final, todos os itens foram empilhados verticalmente, exceto os botões ao final. Esses botões também tiveram suas funções mais explicitadas (“não gostei” para repetir a busca, e “gostei” para salvar o resultado).

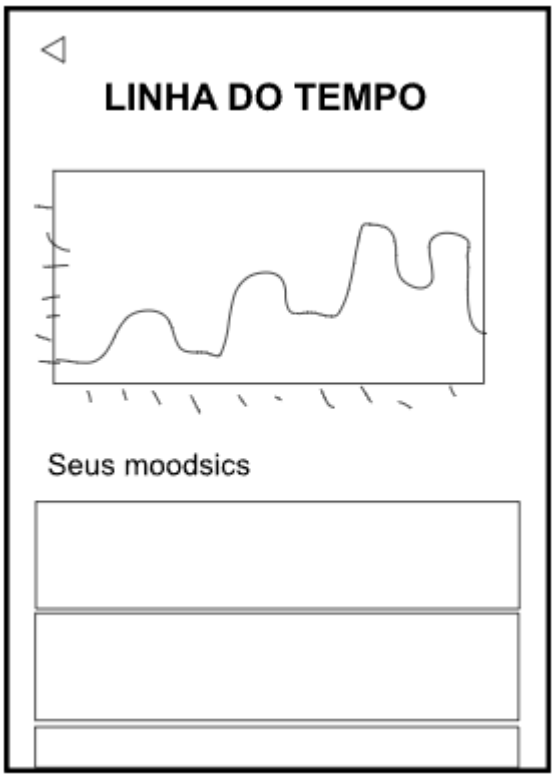

Figura 5: Protótipo da tela de linha do tempo (Timeline)

O último desenho feito foi o da tela de linha do tempo (Figura 5), em que as buscas e seus resultados estarão disponíveis após o salvamento delas pelo usuário. Ele mostra primeiramente um gráfico que agrega informações sobre todas as pesquisas salvas (no protótipo, a ideia era mostrar uma reta com o sentimento predominante em cada mês, no produto desenvolvido é mostrado um gráfico de barras com todas as emoções e quantas vezes elas foram detectadas), e abaixo, uma lista das pesquisas salvas, que foram apelidadas de “Moodsics”

Um dos problemas percebidos após a confecção desses protótipos foi a falta de especificidade dos elementos. Isso pode ser notado pelos itens da lista de Moodsics serem apenas retângulos, não sendo possível inferir quais informações de cada pesquisa seriam mostradas ao usuário, como texto digitado, data ou playlist encontrada, ou até mesmo como elas seriam ordenadas na lista. Isso foi posteriormente definido e, na solução desenvolvida, será possível ver com clareza essas informações. Também poderão ser vistas as telas das quais não foram feitos protótipos, que incluem as telas de detalhes e exclusão de uma pesquisa, assim como as telas de *login* e registro.

Em relação ao layout geral dos elementos, essa página ficou praticamente igual à versão desenvolvida, sendo as únicas diferenças a centralização dos textos e a inclusão de uma barra de pesquisa.

4.4 – Diagrama de classes

O Moodsic não possui uma base de dados muito complexa, mas ainda assim decidi fazer um diagrama de classes na linguagem UML (Figura 6), para que se entendesse melhor a disposição e o tipo dos dados, além dos atributos que seriam necessários.

A princípio, existem apenas três classes na aplicação, podendo-se expandir esse número no futuro, caso a aplicação ofereça a possibilidade de se buscar músicas ou álbuns, por exemplo. A primeira delas é User, que são as informações pessoais do usuário, já oferecidas de forma padrão no Django. Um usuário pode ter zero ou mais Moodsics, que são as informações sobre cada pesquisa salva pelo usuário. Finalmente, um ou mais Moodsics estão relacionados com uma Playlist, que salva as informações da *playlist* encontrada no Spotify.

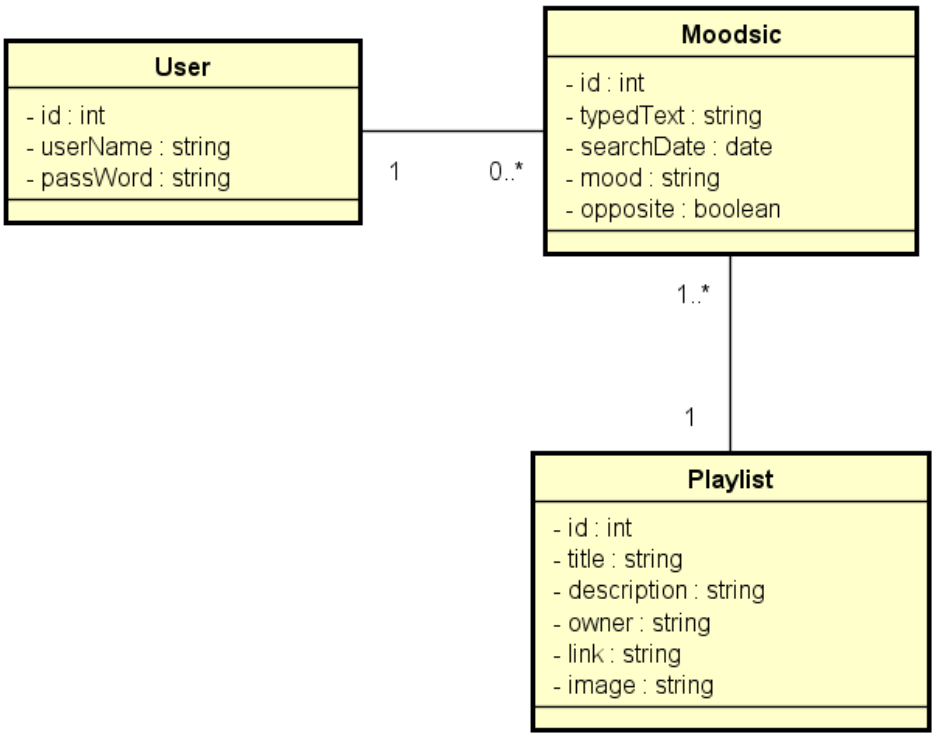

Figura 6: Diagrama de classes UML

4.5 – Diagrama de fluxo de telas

O próximo diagrama realizado (Figura 7) tem por objetivo demonstrar como seria a navegação do usuário pelo site, além de enumerar as telas que ele encontraria. Os retângulos arredondados representam cada uma das telas atualmente disponíveis; as setas representam o sentido de navegação; os textos em cima ou perto das setas representam a ação tomada para um redirecionamento e, finalmente, a área pontilhada engloba as telas que são acessíveis a qualquer momento no site por meio de botões no header (isso foi feito para se evitar excesso de setas e aumentar a legibilidade).

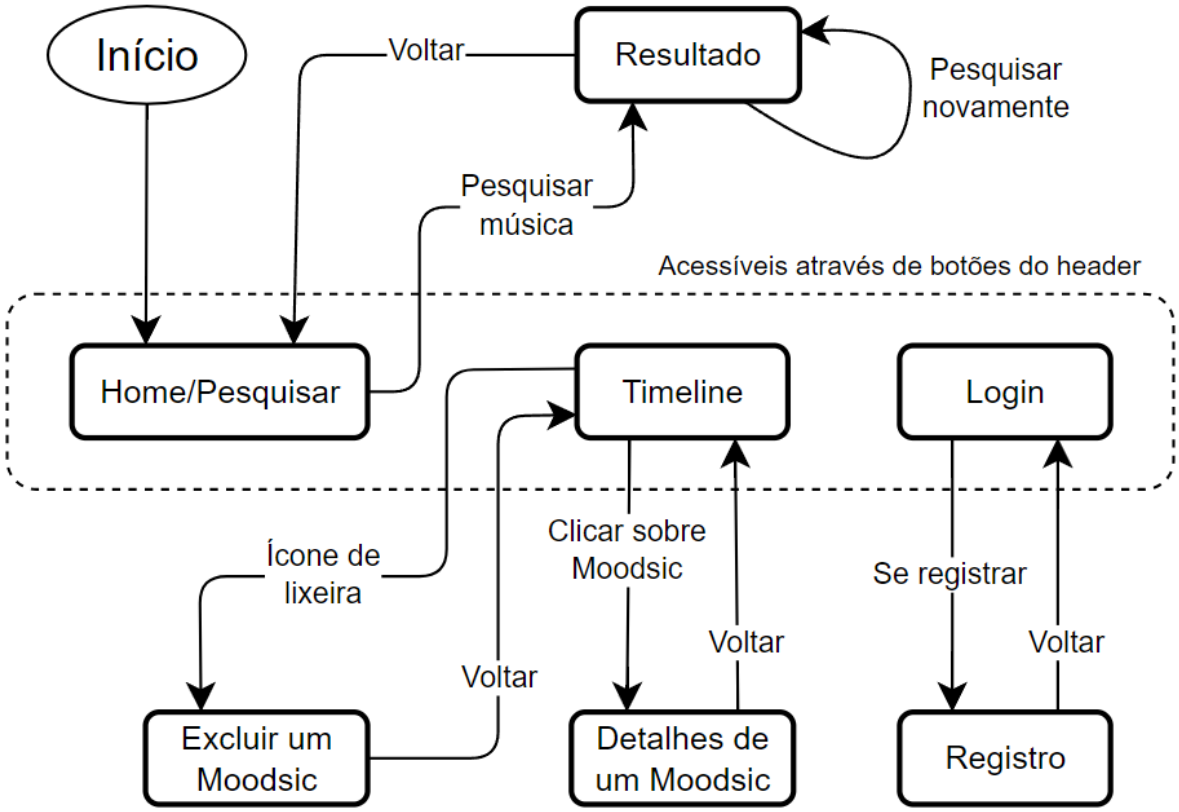

Figura 7: Diagrama de Fluxo de Telas

4.6 – Modelo de casos de uso

O próximo passo foi descrever com mais detalhes todas as funcionalidades que um certo tipo de usuário teria à sua disposição ao utilizar o Moodsic. Isso foi feito por meio da criação de um Diagrama de Caso de Uso (Figura 8), no qual é possível ver claramente os três tipos de autenticação possível no sistema (não logado, logado e administrador) assim como todas as funcionalidades disponíveis para cada um desses níveis de autorização.

As setas com um triângulo na ponta representam uma generalização de uma classe por outra, ou seja, que uma entidade (ou ação) tem a maioria das características daquela que generaliza. Portanto, cada Administrador do sistema também tem as mesmas permissões de um usuário que fez login, e cada usuário logado também tem permissões de um usuário “comum”, sem login. As setas pontilhadas representam ações automáticas a partir de outras (*include*) ou opcionais (*extend*). As elipses representam todas as ações do sistema, que são conectadas a seus respectivos atores por retas.

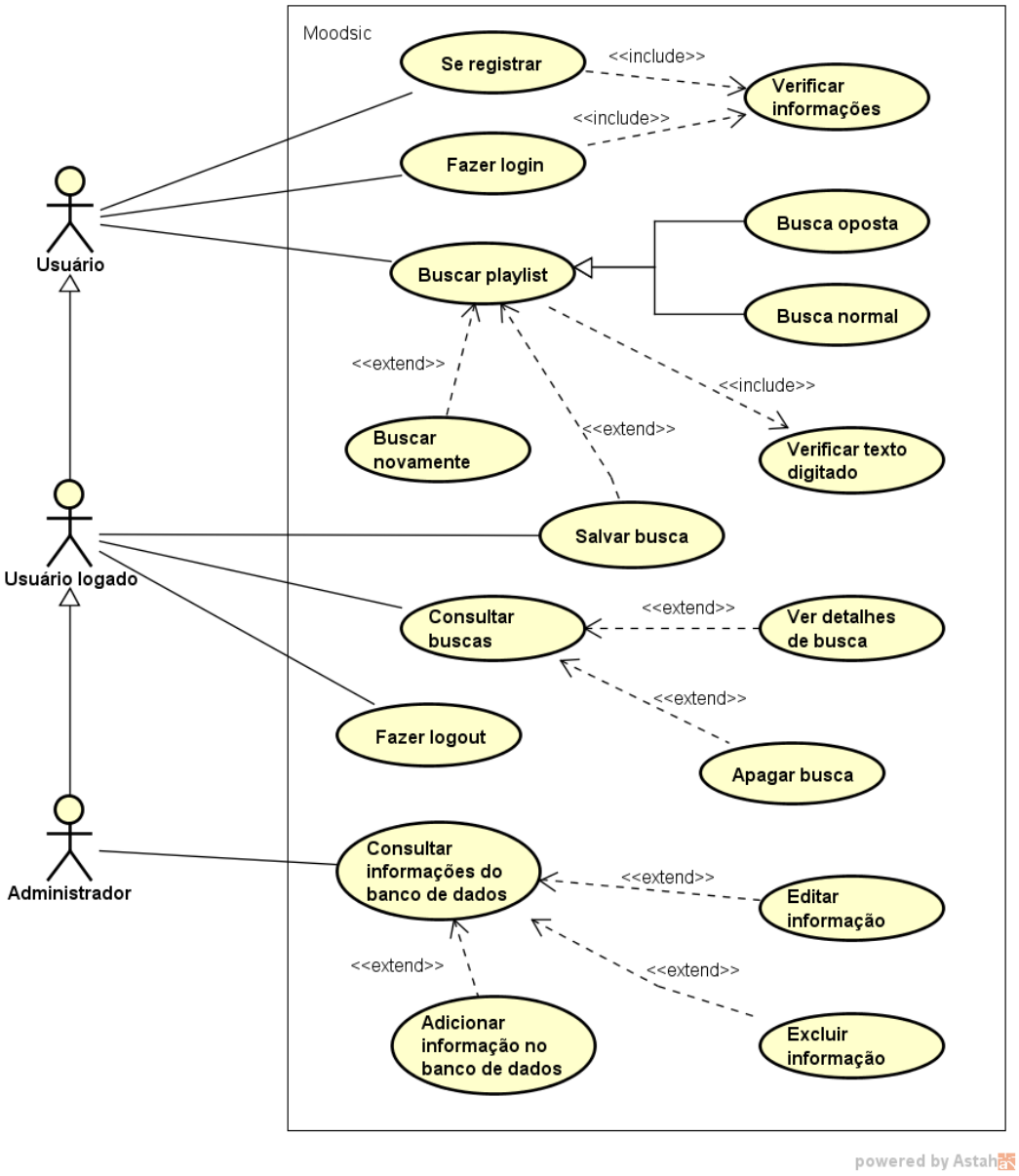

Figura 8: Diagrama de Casos de Uso

Depois de feito o diagrama, decidiu-se descrever de forma mais detalhada como seria a principal jornada do usuário pelo sistema (procurar por uma *playlist*), e isso foi feito por meio de uma Descrição de Caso de Uso (Tabela 1). Nela podemos ver claramente os passos que o usuário deve tomar para achar uma playlist e, ao final, ter acesso a ela e aos detalhes da busca, com os passos já fazendo referência a elementos da interface final. A descrição também contempla um fluxo alternativo.

UC001 – Buscar Playlist	
Objetivo	Que o usuário possa pesquisar por uma *playlist* no Moodsic e ver os resultados de sua busca.
Requisitos	Não se aplica.
Atores	Usuário.
Prioridade	Alta.
Pré-condições	Nenhuma.
Pós-condições	Ter acesso à página de resultados, que inclui todas as informações da busca realizada e da *playlist* encontrada, assim como as opções de salvamento ou repetição da busca.
Fluxo Principal	[P1] – Usuário acessa a *homepage*. [P2] – Usuário preenche a caixa de texto com seu relato. [P3] – Usuário marca ou não a *checkbox* de pesquisa oposta. [P4] – Usuário clica em "*Find me some music*". [A1] [P5] – Sistema mostra ícone de carregamento e desativa botão de pesquisa. [P6] – Sistema recebe informações digitadas e realiza análise emocional do texto [P7] – Sistema faz a busca pela *playlist* com as informações da análise textual, retornando as informações. [P8] – Usuário é redirecionado para página de resultados da pesquisa. [P9] – Usuário vê informações da pesquisa e tem as opções de refazê-la, ou salvá-la (se estiver logado).
Fluxos Alternativos	[A1] – Caixa de texto está vazia quando usuário clica no botão. [P1] – Sistema impede o avanço para [P5] do Fluxo Principal até a caixa ser preenchida.

Tabela 1: Descrição de caso de uso

4.6 – Tarefas

Existe uma peculiaridade no planejamento do Moodsic, que é o fato de que eu já havia sido aprovado em Projeto Final I com outro projeto. Portanto, o desenvolvimento do Moodsic foi planejado para durar pouco menos de um período letivo (seis meses) e não dois períodos (um ano), como geralmente acontece em projetos finais da PUC-Rio. Esse planejamento mais acelerado foi possível pois, quando ele foi proposto e seu desenvolvimento estava para começar, eu já havia sido aprovado em todas as demais disciplinas do curso, podendo, dessa forma, me dedicar de forma integral ao desenvolvimento do Moodsic.

Dito isso, as tarefas que foram planejadas para o desenvolvimento deste projeto foram as seguintes:

1. Pesquisas sobre modelos de *Machine Learning*
2. Testes / treinamento do modelo
    a. Obs.: no projeto final, como já dito, foi utilizado um modelo pré-treinado ao invés de um treinado por conta própria.
3. Modelagem de classes
4. Implementar tela principal
5. Integrar análise do texto
6. Integrar pesquisa com API do Spotify
7. Implementar tela de música encontrada
8. Fazer primeiro teste
9. Implementação do *back-end*
10. Implementar tela de *login/logout*
11. Fazer escrita do documento final (monografia)
12. Implementar possibilidade de salvar um resultado
13. Implementar tela de histórico (*timeline*)
14. Implementar tela de perfil/configurações
15. Fazer testes finais
16. Montagem e ensaio da apresentação para a banca

4.7 – Cronogramas

Na proposta de mudança de tema, entregue no início do período de 2023.2, o cronograma proposto para a realização destas tarefas foi o seguinte:

Tarefa/Mês	Julho				Agosto				Setembro				Outubro			
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																

Tabela 2: Cronograma de tarefas planejadas para o projeto

No entanto, no desenvolvimento que de fato ocorreu, a distribuição do desenvolvimento das tarefas ficou aproximadamente conforme a tabela seguinte:

Tarefa/Mês	Julho				Agosto				Setembro				Outubro			
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																

Tabela 3: Cronograma de tarefas executadas no projeto

Algumas diferenças são notáveis, especialmente a infeliz ausência de testes formalizados. Os testes realizados foram feitos por mim, ao longo do desenvolvimento e de forma informal. A cada tela adicionada ou a cada mudança na lógica, tentei averiguar todos os possíveis caminhos a serem tomados e o resultado de todas as operações, sejam elas de alteração do banco de dados ou do nível de interface. A tela de configurações também ficou de fora a priori, pois decidi priorizar a escrita do documento e a preparação para a apresentação, e também por julgar que a sua implementação não adicionaria tanto ao projeto já que, a princípio, ela só teria uma configuração (estilo musical favorito).

Também é possível observar que algumas tarefas tiveram durações diferentes do esperado. O treinamento do modelo precisou de duas semanas ao invés de uma, dado que era um tema novo para mim e as dificuldades já relatadas, o que me levou a optar por um modelo pré-treinado no fim das contas. Porém, esse atraso foi compensado na implementação da análise do texto, já que a biblioteca Transformers se demonstrou bem intuitiva de ser implementada, e por conta de a análise ter se restringido ao aspecto emocional, com a extração de *keywords* do texto ficando de fora. A integração com a API do Spotify também demorou menos tempo que o esperado, graças à biblioteca SpotiPy e às facilidades que ela oferece. A implementação do *back-end* também foi mais simples que o previsto, sendo muito facilitada pelo próprio Django e seus recursos. Já com a tela de *timeline* aconteceu a situação oposta e seu desenvolvimento levou um pouco mais de tempo que o planejado, muito por conta do seu desenvolvimento envolver recursos adicionais que não foram considerados no primeiro cronograma: a tela de detalhes de um Moodsic, a tela de confirmação de exclusão de um Moodsic e o gráfico de emoções.

Por fim, vale lembrar que algumas tarefas realizadas não estão presentes nestes cronogramas. Uma delas é a estilização CSS, que foi implementada durante as duas últimas semanas de setembro. Outras são a possibilidade de pesquisar novamente uma *playlist* com o mesmo texto e a de pesquisar uma *playlist* oposta ao sentimento atual, que foram feitas na segunda e terceira semanas de setembro, respectivamente.

5 - Projeto e especificação do sistema

5.1 – A arquitetura do Django

Para se compreender como o Moodsic funciona, é também necessário entender como o *framework* Django se organiza. Em um projeto, podem existir vários “apps”, que seriam módulos independentes de um sistema, com suas próprias subpastas, arquivos, configurações e testes. Porém, dada a escala pequena do Moodsic, decidi usar apenas um *app* para o desenvolvimento de todo o site. Na Figura 9, abaixo, encontra-se a estrutura de pastas básica do projeto, a primeira pasta Moodsic sendo a do projeto em si, contendo configurações gerais e o banco de dados por exemplo, enquanto a segunda pasta Moodsic é a do *app*, contendo arquivos de desenvolvimento das páginas como *templates* HTML, arquivos estáticos, *views* e configurações de URL.

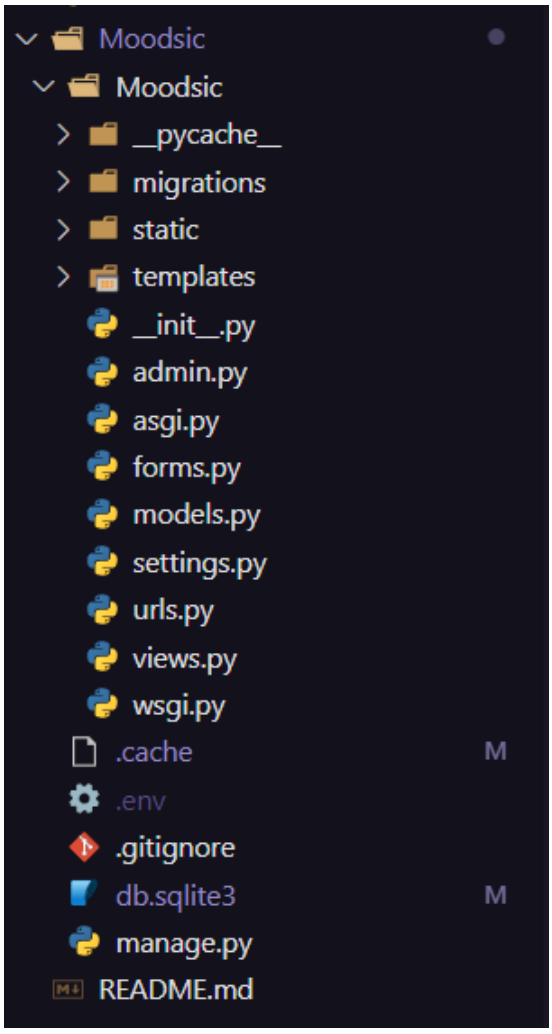

Figura 9: Estrutura de pastas do Moodsic no VSCode

A arquitetura utilizada pelo *framework* Django é a chamada MVT (Model Template View), que por sua vez é uma variação do padrão MVC (Model View Controller) (DJANGO S.F.). A seguir cada um dos três componentes do MVT serão explicados, assim como seus papéis no contexto do Moodsic.

5.2 – O Componente Model

Primeiramente, o componente Model é responsável pela definição dos dados que serão criados no banco de dados e salvos pela aplicação. Neste sentido seu papel é basicamente idêntico ao Model do MVC.

O arquivo “models.py” define essas informações, no caso do Moodsic contendo as definições das classes Moodsic e Playlist, que herdam da classe genérica “Model” do Django, e também a declaração de seus atributos com seus respectivos tipos. A definição da classe User não é necessária, pois já está definida como padrão no Django, bastando importar a classe “User” da biblioteca “django.contrib.auth.models” quando for utilizada.

A classe Moodsic, como já definido no diagrama de classes (Figura 6), possui os seguintes atributos: “user” (chave estrangeira para um User), “playlist” (chave estrangeira para uma Playlist), “mood” (tipo Char, emoção detectada no texto da pesquisa), “typedText” (tipo Char, o texto da pesquisa em si), “searchDate” (tipo DateTimeField, guarda a data e hora da pesquisa) e “opposite” (tipo Booleano, guarda se pesquisa foi oposta ou não).

Já a classe Playlist possui os seguintes atributos: “title” (tipo Char, título da *playlist*), “description” (tipo Char, descrição da *playlist*), “owner” (tipo Char, dono da *playlist*), “link” (tipo Char, link para ver a *playlist* no Spotify) e “image” (tipo Char, URL da imagem da *playlist*).

Após essas definições (ou alguma mudança), basta utilizar os comandos “python manage.py makemigrations” e “python manage.py migrate” para que a base de dados correspondente seja criada, que é o arquivo “db.sqlite3”.

No arquivo “forms.py” teríamos a configuração de quais atributos dessas classes seriam utilizados em formulários no site, mas como não existe a necessidade de um formulário explícito para o salvamento de Moodsics (pesquisas), esse arquivo não foi utilizado.

5.3 – O Componente View

O componente View do MVT, diferente daquele do MVC, que lida com a forma com que as informações são exibidas para o usuário, lida com somente quais serão as informações a serem mostradas, fazendo a intermediação entre o Model e os Templates do Django, definindo informações de contexto para as páginas e fazendo redirecionamentos.

O arquivo “views.py” é o responsável por abrigar as *views* da aplicação. Cada *view* é uma função ou classe Python que recebe uma *web request* (GET, com parâmetros na própria URL ou POST, com parâmetros no corpo da *request*) e retorna uma *web response* (que pode ser desde uma mensagem simples até um redirecionamento para outra página). Algumas das principais *views* da aplicação serão explicadas a seguir.

A view searchPlaylist, por exemplo, lida com todo o processo para buscar e retornar uma *playlist*. Assim que ela recebe um *request* POST (no caso, o texto digitado pelo usuário) ela recupera e verifica as informações. Se o texto digitado não for vazio, ela continua suas operações, se não, sua execução é abortada para que o usuário permaneça na mesma página. O valor da *checkbox* “busca oposta” também é recuperado (Sim ou Não). Validadas as informações, é inicializado o motor de análise textual da biblioteca Transformers e a ele é passado o texto digitado pelo usuário. A função getHighestMood recebe o resultado da análise, descobrindo o sentimento com maior “score” (sendo assim, o principal), e o retornando. Então, é utilizado o sentimento principal do texto para se selecionar uma entre diversas palavras chaves condizentes com esse sentimento na função getKeywords. Com a *keyword* selecionada, são recuperadas as informações de autenticação do Spotify (salvas de forma segura como variáveis de ambiente no arquivo oculto “.env”) e elas são utilizadas para inicializar o motor do Spotify por meio da biblioteca SpotiPy. Depois, é feita a pesquisa em si, utilizando-se a *keyword* selecionada e especificando-se o tipo de retorno como “playlists”. Uma *playlist* aleatória é selecionada dentre até 20 encontradas, e dela são extraídas informações como seu nome, descrição e link. Estas informações, então, são incluídas em um dicionário e a *view* redireciona o usuário para a página de resultados, com o dicionário enviado como contexto para que as informações da playlist possam ser exibidas para ele na página seguinte.

Outra *view* interessante de ser explicada em detalhes é a TimelineView, por conta da diferença na sua construção e por ela ter diversas responsabilidades. Ela é responsável por fornecer as informações da tela de linha do tempo, o que inclui a lista de pesquisas do usuário, seja ela filtrada pela barra de pesquisa ou não, e os dados a serem mostrados no gráfico de sentimentos. Sua construção se deu por meio de uma classe ao invés de uma função, por conta de ser uma *view* que lista um certo conjunto de informações, caso para o qual o Django já possui uma classe genérica (listView) que pode ser herdada. Começamos definindo informações sobre a view, como o Model com a qual trabalha, o Template que referencia e a página para onde o usuário é redirecionado quando suas operações são bem-sucedidas. Depois, especifica-se que o conjunto de pesquisas deve ser ordenado da mais atual para a mais antiga, recupera-se a lista de pesquisas do usuário atual, e são contadas quantas dessas pesquisas existem com cada um dos seis sentimentos detectáveis pelo *app*, para montar o conjunto de dados do gráfico. Após isso, é recuperado o texto digitado na barra de pesquisa que, não sendo vazio, é usado para filtrar o conjunto de pesquisa do usuário. Isso é feito percorrendo-se todas as pesquisas e verificando se seu campo typedText contém o texto digitado. Finalmente, a lista de Moodsics filtrada e os dados do gráfico são retornados como contexto. É interessante mencionar também que essa view herda da classe LoginRequiredMixin do Django, que somente permite sua execução caso o usuário esteja “logado”, e o redireciona para a página de *login* caso ele não esteja.

Temos também a *view* save_results, que recupera da tela de resultados os dados da playlist encontrada e da pesquisa feita, inicializando um objeto Playlist e um objeto Moodsic, salvando-os em nome do usuário atual na base de dados e, finalmente, redirecionando o usuário para a *homepage*.

Outras *views* incluem MoodsicDelete e MoodsicDetails, que tratam das operações de deletar e mostrar detalhes de uma pesquisa respectivamente, e *register*, que trata do registro de novos usuários. Os procedimentos de *login* e *logout* são padrões do Django (classes LoginView e LogoutView), então não precisaram ser criadas views para seus funcionamentos.

O arquivo “urls.py” é o responsável por definir as páginas pelas quais o usuário vai navegar, incluindo seu endereço (URL), a *view* à qual está atrelada e um nome curto para facilitar sua referência no código. Aqui também temos a declaração das páginas de *login*, *logout* e *admin*, que não precisam de *views* correspondentes.

5.3 – O Componente Template

Por fim, temos o componente Template. Ele é o responsável por definir como as informações serão apresentadas ao usuário de forma visual no que diz respeito a layout e estilo, se assemelhando mais à View do MVC. No caso do Django, isso se dá principalmente por meio de *templates* HTML. Nestes *templates*, definimos quais componentes gráficos estarão presentes em cada página, seus tipos, seus nomes para referência, assim como informações gerais da página e quais são os arquivos estáticos atrelados a ela. Os templates também são acrescidos de *tags* Django (sinalizadas por "{{ }}" ou "{% %}") que oferecem possibilidades adicionais ou sinalizam informações que o template deve mostrar. Exemplos seriam a *tag* {% load static %}, que aparece no início da página e sinaliza que ela possui conteúdo estático (ex.: imagens, folhas de estilo), as *tags* {% if %}, {% else %}, {% elif %} e {% endif %} que permitem a construção de mecanismos condicionais, e tags de referência a informações de contexto como {{ playlist_info.title }}.

Cada tela da aplicação possui um template correspondente dentro da pasta "templates/Moodsic" (exceto a tela de *admin*, que é padrão do Django e é implementada da biblioteca nativa django.contrib). Isso inclui as telas de início, resultados, login, registro, *timeline*, detalhes de Moodsic e exclusão de Moodsic. Porém existe também um template separado para o *header* da página, que é reutilizável e é incluído nas outras páginas por meio da *tag* Django {% include %}. Isso faz sentido, pois é um componente que aparece em quase todas as páginas da aplicação e assim evita-se a escrita do mesmo código HTML repetidas vezes.

A folha de estilo CSS também tem um papel grande em definir como serão mostrados os elementos e as informações. A pasta "static" abriga todos os arquivos estáticos da aplicação, que incluem imagens fixas, como o logo da aplicação e a folha de estilos "styles.css". Nessa folha de estilo são definidos atributos adicionais de estilo de todas as páginas, incluindo cores, opções de layout, margens, sombras e animações, por exemplo.

5.4 – Outros arquivos e pastas notáveis

O arquivo "settings.py" abriga uma miríade de configurações e predefinições da aplicação, como diretórios padrão, informações de linguagem e configurações da base de dados. O arquivo "admin.py" define quais informações salvas no banco o administrador do site poderá ver na tela de admin. No caso do Moodsic, apenas o campo "typedText" das pesquisas do usuário foi excluído, pois pode conter informações sensíveis ou relatos muito pessoais do usuário, que não queremos que o admin tenha acesso para que seja mantida a privacidade. O arquivo

“manage.py” contém uma função main() que permite a utilização de diversos comandos pelo terminal para inicialização do site, aplicar mudanças no Model e no banco de dados, reiniciar configurações, entre muitos outros fins.

E, finalmente, a pasta “migrations” abriga todas as mudanças feitas no Model e no banco de dados, desde a sua criação.

5.5 – Utilização do Moodsic

Agora que é possível entender um pouco de como o Moodsic funciona internamente, será explicada a sua utilização na prática, da perspectiva do usuário e com imagens do site finalizado.

A primeira tela que o usuário encontra ao abrir o site é a *homepage* (Figura 10), onde ele poderá realizar a busca. Ele verá a caixa de texto para escrever seu relato, uma *checkbox*, que diz se ele quer uma playlist oposta ao seu sentimento atual ou não, e o botão de pesquisa. Ele também vê o *header*, que contém links para a *homepage*, a linha do tempo e a página de *login* (o link de configurações apenas o redireciona para a *home*, pois a tela de configurações não foi implementada).

Após o preenchimento das informações e o clique no botão, as informações serão processadas e o processo de busca começará, enquanto o botão se torna inativo para evitar cliques duplos e, por consequência, *requests* duplicadas. Um ícone animado de carregamento também aparece no botão.

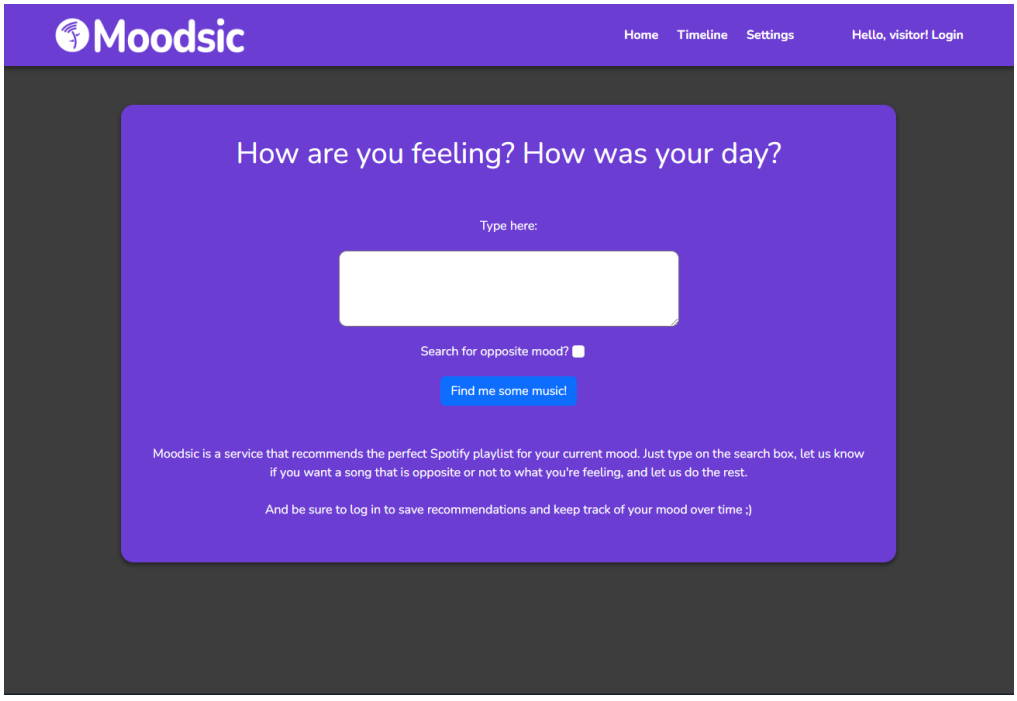

Figura 10: Homepage (tela de pesquisa)

Após o retorno da *playlist* encontrada, o usuário é redirecionado automaticamente para a tela de resultado (Figura 11), onde ele verá informações sobre o texto que digitou, a análise feita e o sentimento encontrado, assim como informações da *playlist*, que incluem seu nome, descrição, dono e imagem.

Também aparecem três botões no fim da página: um para escutar a *playlist* no Spotify, um para pesquisar novamente e um para salvar a playlist. Clicando em "*Search again*" a pesquisa é refeita com a mesma análise e a página é recarregada com o novo resultado.

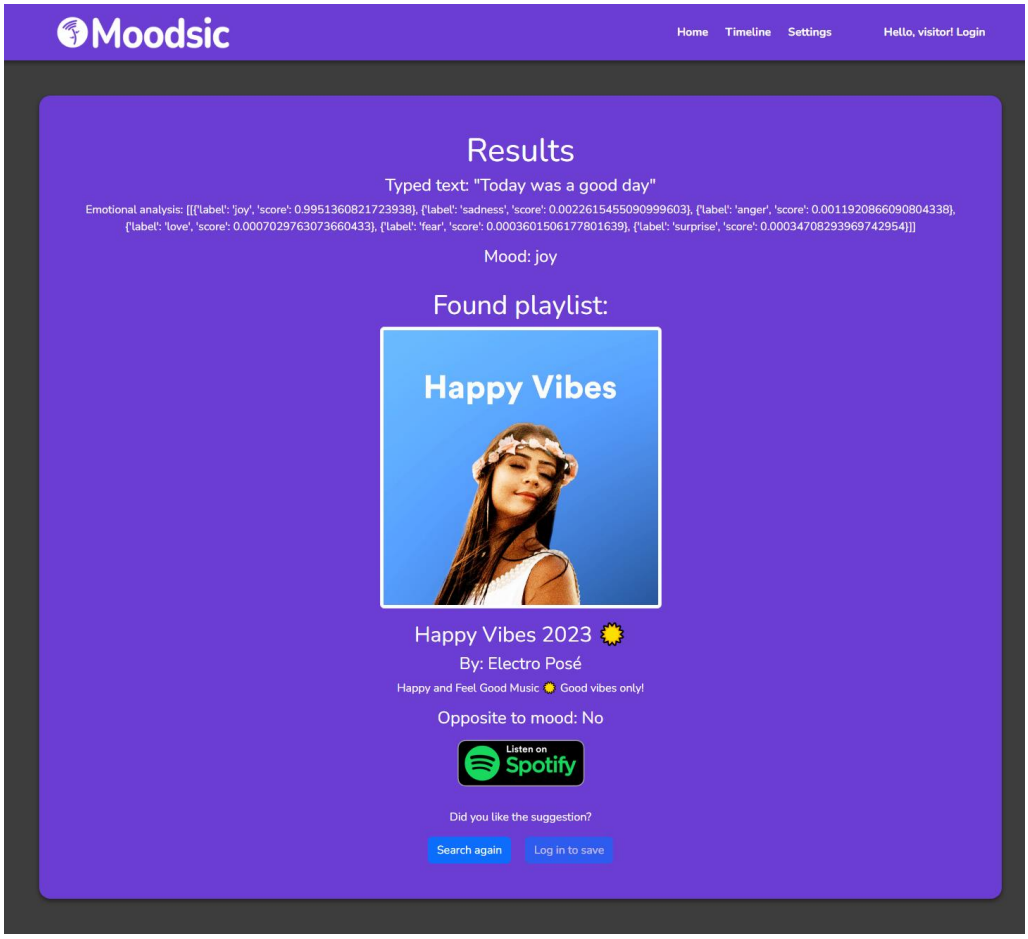

Figura 11: Tela de resultados (sem login)

O usuário pode realizar a pesquisa sem fazer *login*, mas nesse caso o botão de salvar o resultado fica desativado e diz "*Log in to save*", já que precisamos dessa ação do usuário para que seja possibilitado o salvamento. Clicando em "*Hello, Visitor! Login*" no *header*, o usuário é levado para a tela de *login* (Figura 12), que conta com um formulário simples onde ele deve inserir seu nome de usuário e senha.

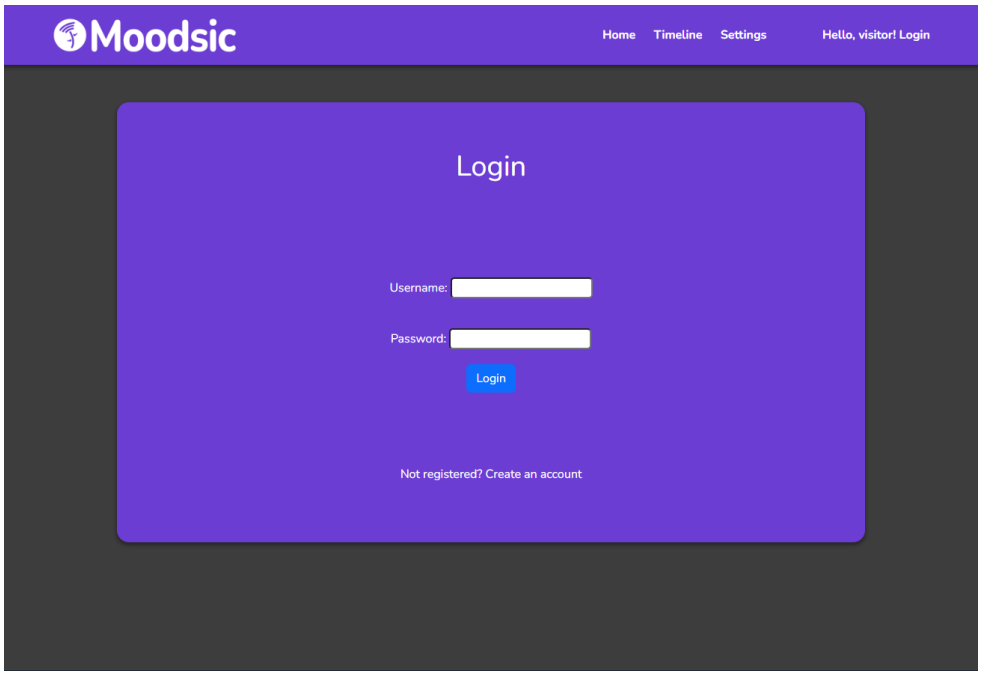

Figura 12: Tela de login

Caso o usuário não possua uma conta, basta clicar em "*Create an account*" para que ele seja levado à tela de registro (Figura 13), onde poderá cria-la. O procedimento de criação de conta é o padrão do Django, com algumas exigências de segurança, como não usar senhas muito comuns ou somente números. O usuário também deve repetir a senha depois de digitá-la. Feito isso, basta clicar em "*Create User*" para que se crie a conta.

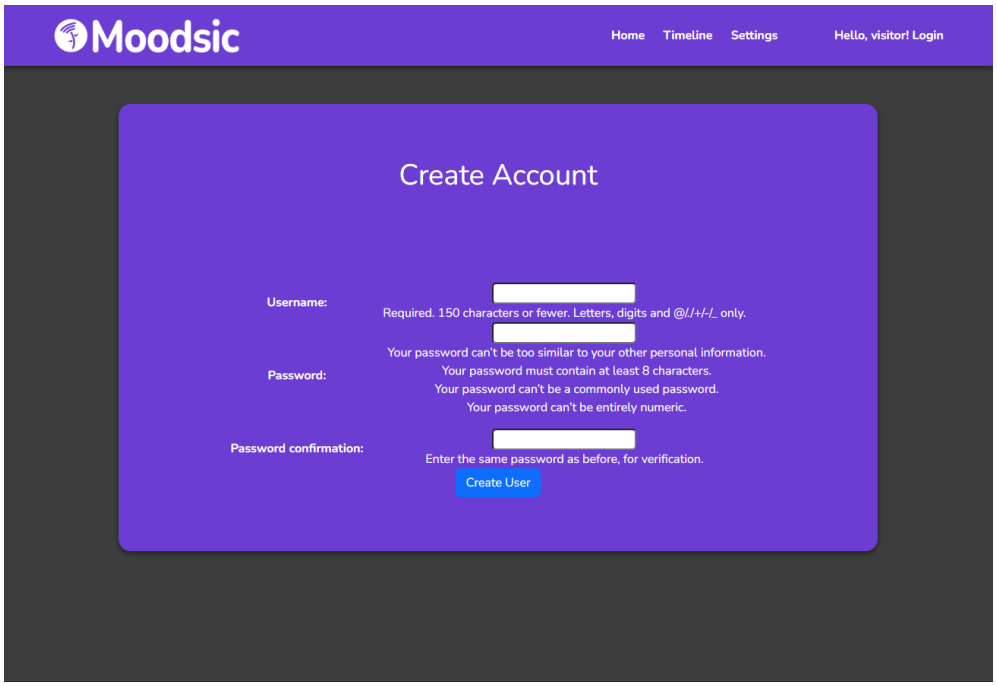

Figura 13: Tela de registro

Após a criação da conta e o login na mesma, o nome de usuário irá aparecer no canto direito do header, juntamente com a opção de fazer *logout*. E, agora que está “logado”, o usuário pode fazer outra pesquisa e salvar suas consultas com o botão de salvamento da tela de resultados (Figura 14), que está habilitado e dizendo “*Save Moodsic*”. Com o clique neste botão, a pesquisa feita e a *playlist* encontrada são ambas salvas no banco de dados em nome do usuário, redirecionando o usuário de volta para a tela de início.

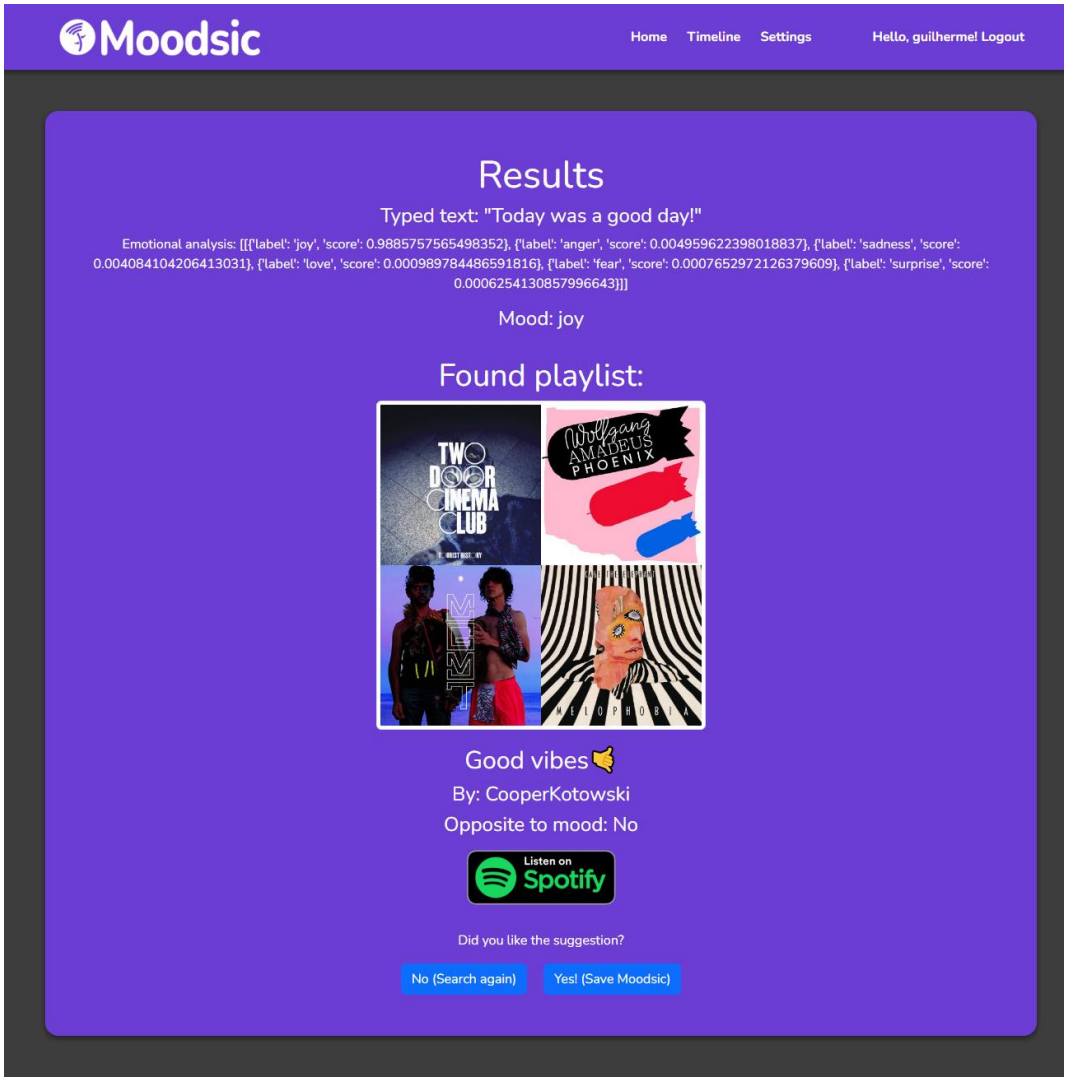

Figura 14: Tela de resultados (com login)

Clicando em “Timeline”, o usuário autenticado pode acessar a tela de linha do tempo (Figura 15), e, caso ele não esteja autenticado, o sistema o redireciona para a tela de *login*. Nessa tela, o usuário encontrará uma lista vertical com todas as pesquisas que já fez, ordenadas da mais atual para a mais antiga, e para cada uma delas mostrando um resumo de suas informações e um botão de exclusão. Além disso, ele também vê um gráfico que mostra todos os sentimentos já detectados em seus relatos e uma barra de pesquisa para procurar Moodsics.

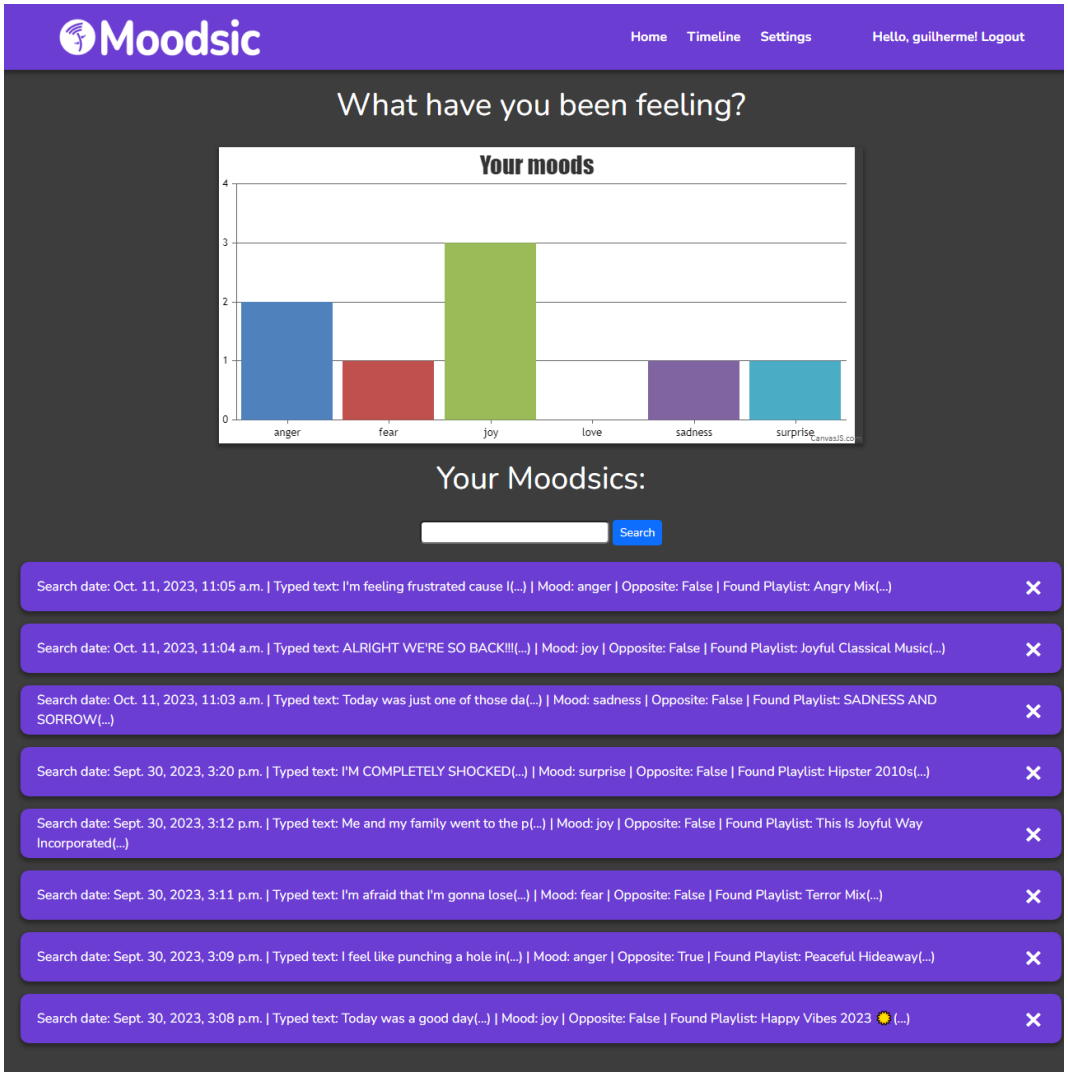

Figura 15: Tela de timeline

O usuário pode passar o mouse por cima das barras do gráfico para ver exatamente quantas ocorrências de cada sentimento existem nas suas pesquisas e, clicando sobre uma pesquisa, ele é levado para a tela de detalhes sobre aquela pesquisa.

Na tela de detalhes (Figura 16), o usuário vê todas as informações sobre uma pesquisa, ou Moodsic, incluindo o texto digitado completo e todas as informações da *playlist* relacionada, ou seja: foto, dono, descrição, e seu link do Spotify.

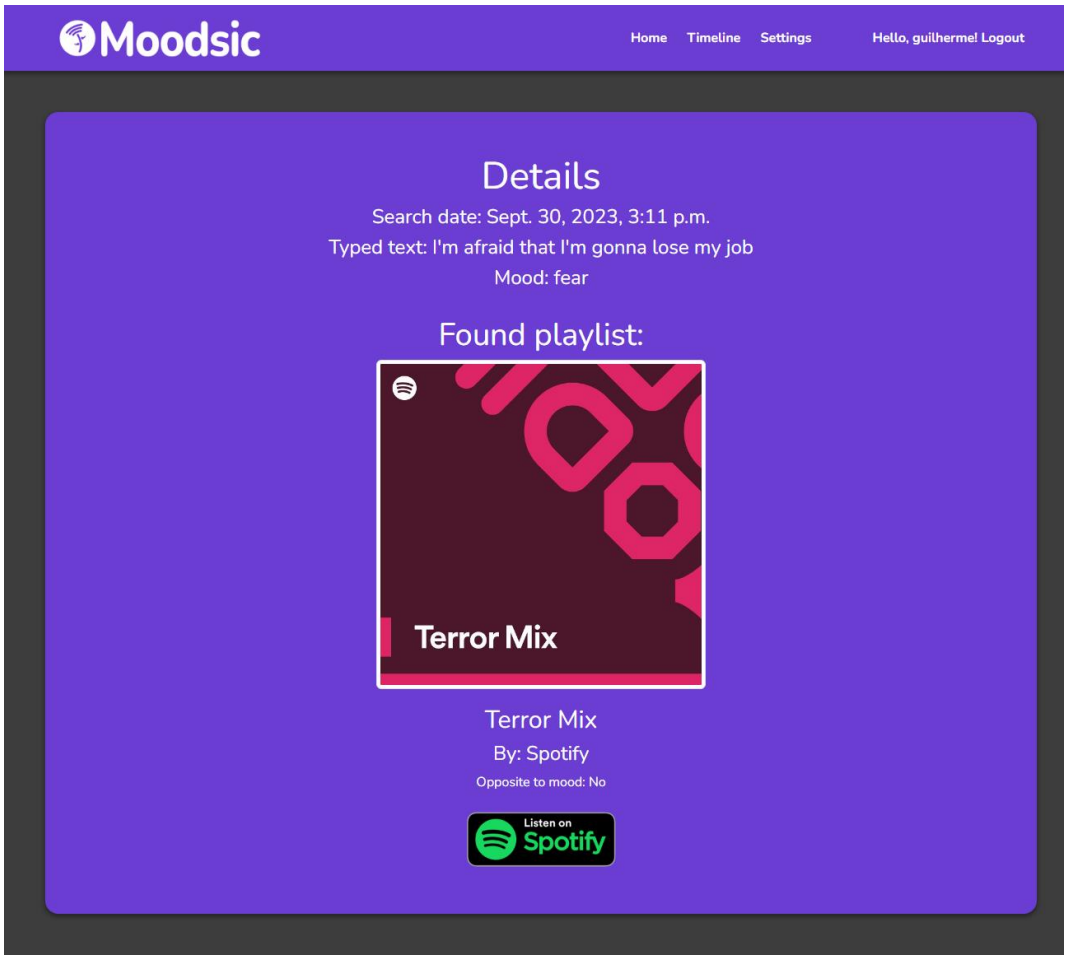

Figura 16: Tela de detalhes de um Moodsic

Ao voltar para a tela de Timeline e clicar no ícone de “X”, que aparece na direita de cada um de seus Moodsics, o usuário é levado para uma tela de exclusão (Figura 17), onde pode realizar esta operação. Nessa tela, aparece um resumo bem sucinto das informações da pesquisa (o texto parcial e sua data), e um botão de confirmação. Ao clicar sobre ele, o Moodsic é excluído permanentemente do banco de dados, e o usuário é redirecionado para a tela de Timeline, onde o Moodsic não estará mais presente.

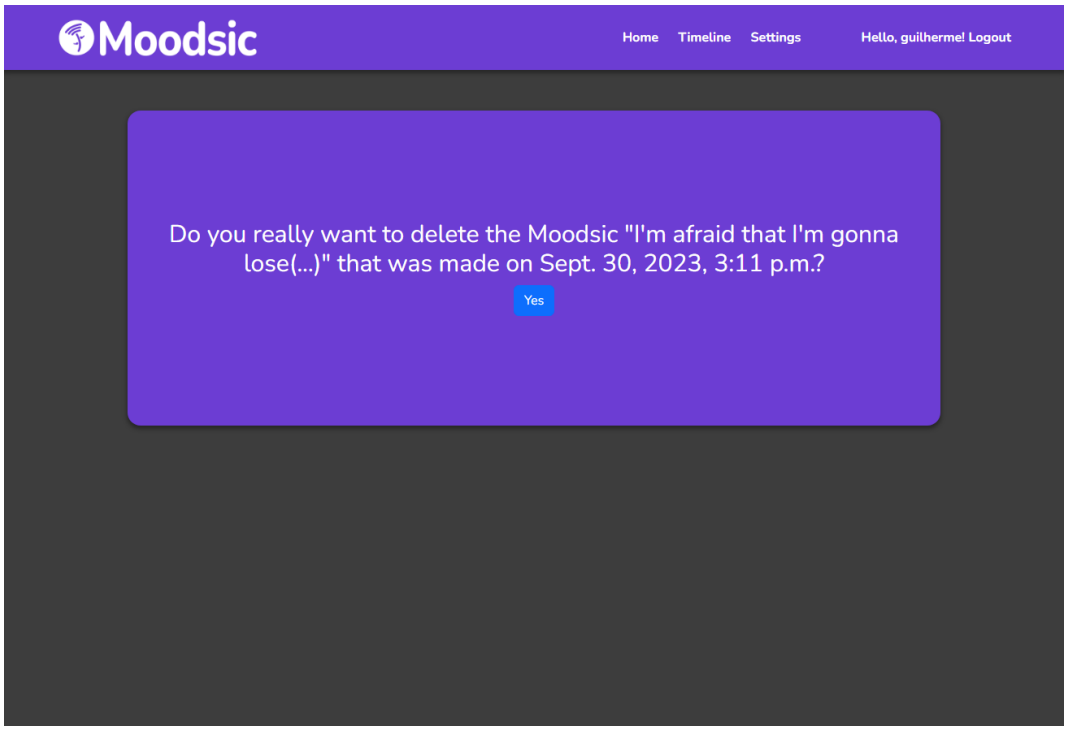

Figura 17: Tela de exclusão de um Moodsic

Por fim, existe também uma seção de administração no site, que pode ser acessada digitando-se a URL da *homepage* + "/admin" na barra de endereço do navegador. O usuário deve fazer *login* com o nome de usuário e senha de administrador (que é uma conta já pré-criada, e só um administrador pode criar outras contas de administrador caso desejado) assim tendo acesso ao dashboard de controle. Ele primeiro vê uma tela (Figura 18) onde pode acessar todos os dados, usuários e grupos (apesar destes não serem utilizados no Moodsic) que foram salvos no projeto. Também pode ver as últimas mudanças que foram feitas no banco de dados, assim como adicionar novas instâncias de cada classe ou novos usuários.

Clicando em "Moodsics" e em um Moodsic específico, por exemplo, ele pode ver (Figura 19) e editar quase todas as informações do mesmo: humor, data de pesquisa, se é uma busca oposta, e qual é a *playlist* e o usuário atrelado a ela. Ele também pode apagar o registro do banco de dados. A única informação que não está disponível nem em visualização quanto em edição para o administrador é o texto digitado, para que seja preservada a privacidade do usuário.

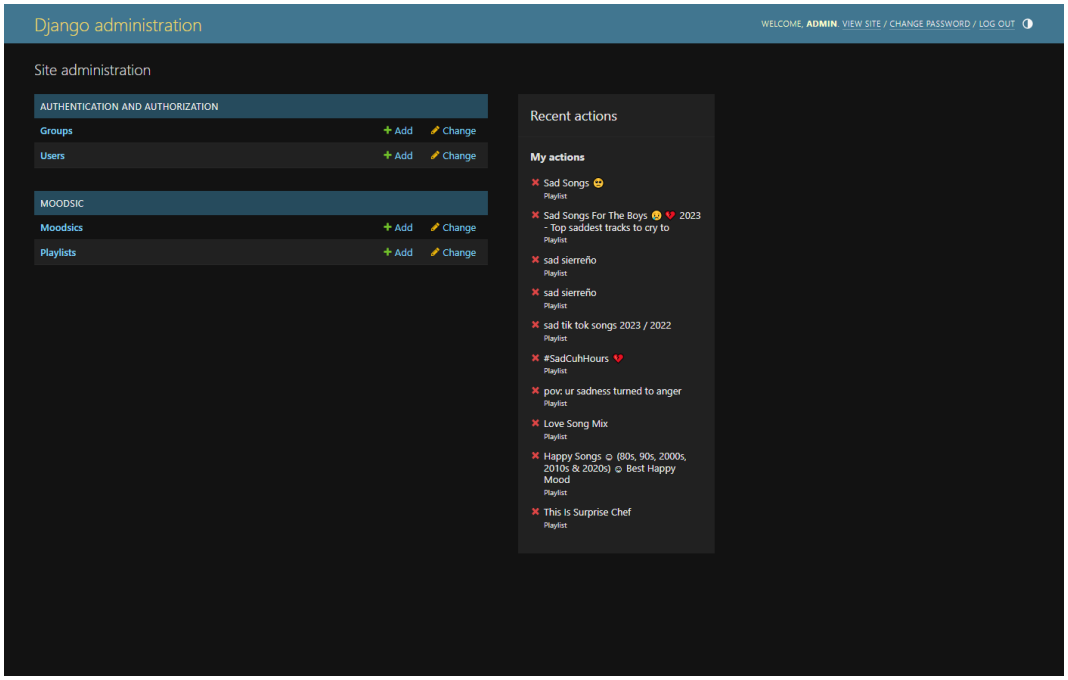

Figura 18: Dashboard de administrador

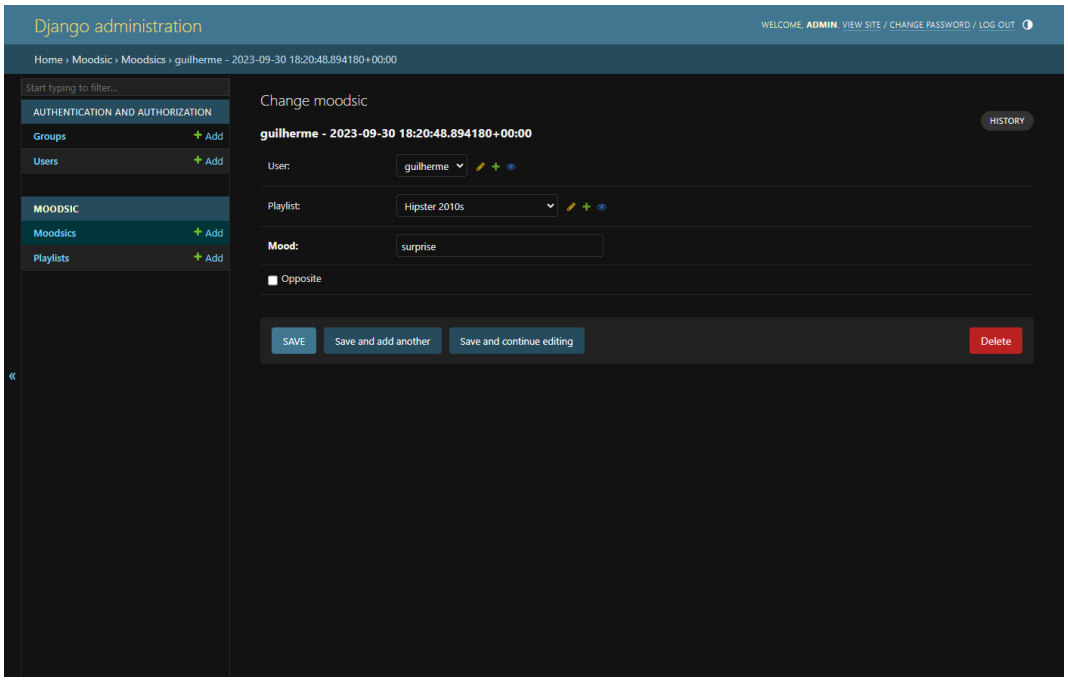

Figura 19: Acessando informações de um Moodsic pela dashboard de administrador

6 – Comentários sobre a implementação

Os testes manuais, conduzidos por mim ao longo do desenvolvimento, e o produto final indicam que a implementação da solução se deu de forma satisfatória, com a vasta maioria das funcionalidades planejadas sendo desenvolvidas sem muitos obstáculos, e funcionando conforme o esperado. O sistema cumpre bem seu propósito, e aparenta funcionar de forma intuitiva, ágil, estável e segura. Porém, como em quase todo projeto, alguns desafios ou situações curiosas apareceram.

6.1 – Desafios

Um dos desafios foi tornar a interface mais responsiva e comunicativa, uma característica que não foi considerada na fase de planejamento. Foi percebido que, por exemplo, o carregamento da busca por playlists podia demorar um pouco, especialmente nas primeiras vezes em que o site é carregado, mas não havia nada na interface da homepage que comunicasse explicitamente que havia um carregamento acontecendo. Portanto, foi decidida a utilização do componente *Spinner*, do Bootstrap, dentro do botão, que apareceria assim que fosse clicado. Ademais, o botão seria desativado para evitar *requests* duplos e sinalizar que é uma ação “final”. Essa implementação foi um pouco não ortodoxa, o que deu origem a uma função Javascript no *template* que verificava, novamente, se o texto era não vazio na hora do clique (apesar de isso já ter sido feito na View). Caso fosse, ela alterava o HTML interno do botão para exibir o ícone de carregamento animado e o texto alterado (“Loading...”), e depois desabilitava o botão. Porém, o que acontecia era que o botão era desativado antes mesmo do POST *request* ser enviado, e assim a pesquisa por *playlists* nunca era feita. Isso foi resolvido por meio da utilização do atributo “onSubmit” do formulário, que só altera a propriedade “disabled” do botão quando o formulário é enviado com sucesso.

Um outro desafio foram as questões de segurança do site. Um dos problemas graves de segurança do site, percebido no início do desenvolvimento, era que um usuário comum poderia ver detalhes de um Moodsic pertencente a outro usuário, e até excluí-lo. Para isso, bastava que ele alterasse a URL da página de detalhes ou de exclusão de um Moodsic para o id correspondente a um outro salvo na base de dados, independentemente de quem fosse seu dono. Isso foi resolvido verificando-se se o usuário atual é o mesmo daquele Moodsic todas vezes que essas páginas são mostradas, ou que um POST *request* de exclusão é enviado, mostrando uma página de erro, em caso negativo.

Outra medida de segurança tomada foi a proteção contra ataques CSRF (*Cross Site Request Forgery*) (IBM CORP.). Este tipo de ataque acontece quando um site externo malicioso contém um link ou um código Javascript que realiza alguma ação em um site que o usuário acessa, usando suas credenciais de acesso. A proteção contra esses ataques é habilitada por padrão no Django, mas deve-se utilizar a *tag* Django {% csrf_token} em todos os formulários que enviam uma solicitação POST, como foi feito, para que seja validado um *token* que confirma que a solicitação veio de um endereço interno ou confiável.

Refinar a relevância das playlists encontradas também se demonstrou, como esperado, um pouco desafiador. A solução atual, de selecionar uma palavra chave aleatória de um conjunto de palavras-chave condizentes com um sentimento, foi um aperfeiçoamento da ideia de simplesmente pesquisar o sentimento no Spotify. Ela funciona relativamente bem, além de manter as pesquisas com um grau de variedade. Porém, alguns resultados estranhos ainda aparecem ocasionalmente, como, por exemplo, se pesquisarmos "*joy*" (relativo ao sentimento de alegria), existe uma chance de a aplicação recomendar playlists de artistas como Joy Division ou Joy Incorporated, que não necessariamente só possuem músicas alegres. Tentou-se minimizar esses tipos de caso ao máximo, por meio da escolha de boas palavras chave para cada sentimento.

6.2 – Possíveis melhorias

Apesar dos sucessos da solução, muitos pontos passíveis de melhora também foram identificados por mim. Um dos principais é a qualidade da busca em si. A busca poderia utilizar mais e melhores palavras chaves possíveis para cada sentimento, ou até utilizar outros métodos além da busca por palavras chave, caso sejam possíveis, evitando-se, assim, os casos onde se encontram playlists de bandas que possuem as palavras chaves em seu nome, por exemplo. Além disso, sua performance, apesar de aceitável, talvez possa ser mais otimizada em alguns casos.

Uma das principais ideias para refinar uma busca, que não foi implementada, é a busca por palavras chaves específicas. Seriam termos presentes dentro do texto, que vão além dos sentimentos, como, por exemplo, "praia", "trabalho" ou "família". Depois, seriam utilizados esses termos, em conjunto com os sentimentos, para tornar os resultados mais personalizados para a situação atual. Acredito que isso seria interessante, caso bem implementado, e até poderia ser oferecido como uma opção configurável para o usuário.

Outra ideia parecida é a possibilidade de o usuário definir o gênero de músicas a ser encontrado, ou que ele defina um estilo preferido em suas configurações. Por exemplo, se um usuário é adepto ferrenho de Heavy Metal e afins, e se sente alegre, ele poderia encontrar músicas desse estilo que soam mais alegres, e não músicas alegres de qualquer gênero, que muito provavelmente pertencerão a outros estilos.

A última melhoria que pensei para as buscas seria a possibilidade de escolher entre a procura por *playlists* ou por apenas uma música. Uma dá mais conteúdo para o usuário, enquanto a outra poderia surpreender mais. Por outro lado, isso envolveria várias mudanças em todos os componentes da aplicação, incluindo *templates*, funções da View e o Model.

Um outro aspecto do site que se demonstrou aprimorável foi a interface de usuário. Poderia haver telas ou notificações simples de confirmação após o usuário realizar certas ações com sucesso, como salvar e excluir um Moodsic. Também seria interessante que a interface fosse mais responsiva em telas menores, como de *smartphones* ou *tablets*. Nos testes que fiz, percebi que todos os itens do corpo do site adaptavam-se de forma aceitável em telas menores, exceto pelo header do site, cujos botões ficam embaralhados e confusos em telas pequenas. Um último *oversight* na interface foi o fato de não haver botões de "voltar" nas páginas ou uma linha do tempo de navegação. Seria uma opção, tecnicamente falando, redundante, pois todo o site é navegável por meio dos botões do *header* e dos próprios controles do navegador, mas poderia tornar a navegação mais amigável de qualquer forma.

Por mais que a privacidade do usuário tenha sido levada em conta (dada a omissão do texto dos relatos do usuário para o *admin*), julgo que teria sido interessante ir um passo além, fazendo-se a criptografia dos dados dos usuários. No momento, apenas as senhas dos usuários são criptografadas com o algoritmo SHA-256 (ENCRYPTION CONS. LLC.), pois isto é padrão no Django, as demais informações sendo salvas em texto plano. Isto é melhor que nada, mas não é ideal, pois qualquer um que conseguir acessar a base de dados por alguma via externa terá acesso aos relatos dos usuários, por exemplo.

Sobre a tela de linha do tempo, a barra de pesquisa da mesma poderia suportar pesquisas em tempo real, enquanto se digita o texto, por meio do AJAX (MOZILLA CORP.). No momento é necessário apertar o botão de pesquisa para ver os resultados. Também poderiam existir opções de filtro ou de ordenação da lista de pesquisas, e mais opções de visualização para o gráfico de sentimentos.

7 - Considerações Finais

Considero que foi muito interessante e de grande valor desenvolver esse projeto, que possibilitou a união de duas de minhas paixões pessoais (música e tecnologia) de uma forma coesa e divertida, ao mesmo tempo em que pude explorar e exercitar muito do que aprendi no meu curso de Sistemas de Informação.

Acredito que o projeto cumpre o seu propósito, pois desenvolvi uma aplicação relativamente simples, mas funcional, e que mostra o potencial de uma plataforma intuitiva que explore o *Machine Learning* e a conexão das nossas emoções com a música. Assim ela dá a seus usuários uma nova maneira de descobrir música, além de, também, poder atuar como uma ferramenta de recordação e acompanhamento emocional, única e engajadora.

Acho factível que o Moodsic contribua positivamente para o meio acadêmico, pois ainda existem poucos projetos que utilizam bibliotecas de *Machine Learning* de forma parecida com o que foi implementado, e, também, poucos projetos de Sistemas de Informação que têm a música como tema. Portanto, ele pode inspirar projetos parecidos e mais interesse nessas áreas. Caso o Moodsic se torne uma aplicação comercial no futuro, acredito, também, que ele tem potencial para cair nas graças do povo e afetar positivamente a vida do cidadão comum, desde que sejam feitas algumas melhorias como funcionalidades extras, aperfeiçoamentos de usabilidade, a implementação de um *back-end* mais robusto (como MySQL (ORACLE)) e, possivelmente, versões *mobile* na forma de aplicativos.

Se o projeto fosse iniciado hoje em dia, algumas coisas teriam sido feitas de forma diferente. Obviamente, eu não teria tentado treinar de um modelo de análise sentimental próprio, e teriam sido adotados, logo de início, os modelos pré-treinados do Hugging Face. Essa tentativa custou tempo e esforço considerável para algo que não teve utilidade prática no projeto (apesar de ter sido uma experiência curiosa). Claramente, também, eu deveria ter realizado mais testes e ter dado a devida atenção a eles, tentando-se fazer testes funcionais desde os primórdios do desenvolvimento (algo como *Test Driven Development* (ALEXANDRE, 2016)). Eu também deveria ter organizado testes com usuários, com a utilização das metodologias adequadas, que me permitiriam adquirir *feedbacks* valiosos que poderiam ter auxiliado muito no desenvolvimento e gerado melhorias na usabilidade.

Aprendi muito com essa experiência, tendo-se adquirido conhecimento e experiência prática sobre diversos assuntos, incluindo o uso de diversas API's, e sobre o funcionamento interno e implementação de técnicas de Machine Learning. Fora todos os conceitos que já conhecia, puderam ser revisitados e aperfeiçoados assuntos como desenvolvimento web, modelagem de software e todos os outros que já foram citados no início do documento. Também aprendi sobre a importância de se priorizar tarefas e de se manter um escopo factível.

É importante destacar de que formas esse trabalho aparenta ter muito potencial para expansões futuras ou para inspirar trabalhos similares por outros alunos. Já foram apresentadas inúmeras melhorias possíveis, e o potencial criativo para mudanças e para o desenvolvimento de outras aplicações que expandem a ideia central é imenso.

Seria interessante, por exemplo, investigar como as redes sociais poderiam ser integradas em uma solução como essa, seja de um jeito mais simplista, que poderia ser dando aos usuários opções de compartilhamento dos seus sentimentos, seja permitindo a visualização da evolução emocional de amigos, ou de jeitos ainda mais interessantes: Uma das ideias para o futuro do Moodsic seria integrar a análise sentimental com posts que o usuário faz em uma rede social, como o Twitter, dando a ele recomendações musicais baseadas na análise de sentimento dos *tweets* que fez em um certo período de tempo.

Uma outra ideia seria adquirir os relatos pessoais por meio de outros meios além do meramente textual, como o reconhecimento de voz (*speech to text*), que permitiria até uma implementação em assistentes pessoais inteligentes como a Amazon Alexa (AMAZON INC.).

Poderia ser implementado o suporte à outras línguas, como o Português Brasileiro. Para isso, deveria ser treinado um modelo próprio que seja performático, convincente, e que analise vários sentimentos de um texto na língua escolhida, ou encontrar um modelo pré-treinado que faça o mesmo. De acordo com as pesquisas que fiz durante o desenvolvimento, um modelo com essas exatas características ainda aparenta não ter sido criado, ou ao menos disponibilizado publicamente.

Por fim, seriam interessantíssimos a consulta e o envolvimento de profissionais como psicólogos, psiquiatras ou musicoterapeutas no desenvolvimento, para tornar os processos de análise e acompanhamento das emoções ainda mais robustos e completos, além de dar mais autoridade e validação ao projeto.

8 – Referências Bibliográficas

ALEXANDRE. Test Driven Development: TDD Simples e Prático. **DevMedia**, 2010. Disponivel em: <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 21 out. 2023.

AMAZON INC. Conheça Alexa. **Amazon**. Disponivel em: <https://www.amazon.com.br/b?ie=UTF8&node=19949683011>. Acesso em: 21 out. 2023.

BOOTSTRAP TEAM. **Bootstrap**. Disponivel em: <https://pypi.org/project/python-dotenv/>. Acesso em: 21 out. 2023.

DJANGO SOFTWARE FOUNDATION. **Django**. Disponivel em: <https://www.djangoproject.com>. Acesso em: 21 out. 2023.

DJANGO SOFTWARE FOUNDATION. FAQ: General. **Django documentation**. Disponivel em: <https://docs.djangoproject.com/en/4.2/faq/general/>. Acesso em: 21 out. 2023.

ENCRYPTION CONSULTING LLC. What is SHA? What is SHA used for? **Education Center**. Disponivel em: <https://www.encryptionconsulting.com/education-center/what-is-sha/>. Acesso em: 21 out. 2023.

FENOPIX, INC. **CanvasJS**. Disponivel em: <https://canvasjs.com>. Acesso em: 21 out. 2023.

GOOGLE. **Bard**. Disponivel em: <https://bard.google.com/chat?hl=pt-BR>. Acesso em: 21 out. 2023.

GR1D. **Principais tendências em API's para 2022 e além**, 2022. Disponivel em: <https://gr1d.io/2022/09/09/inovacao/>. Acesso em: 21 out. 2023.

HUGGING FACE. **Transformers documentation**. Disponivel em: <https://huggingface.co/docs/transformers/index>. Acesso em: 21 out. 2023.

IBM CORPORATION. **What are Naïve Bayes classifiers?** Disponivel em: <https://www.ibm.com/topics/naive-bayes>. Acesso em: 10 Julho 2023.

IBM CORPORATION. Prevenção de ataques Cross-site Request Forgery (CSRF). **IBM Security Verify Access**. Disponivel em: <https://www.ibm.com/docs/pt-br/sva/10.0.6?topic=configuration-prevention-cross-site-request-forgery-csrf-attacks>. Acesso em: 21 out. 2023.

KUMAR, S. python-dotenv. **PyPI**. Disponivel em: <https://pypi.org/project/python-dotenv/>. Acesso em: 21 out. 2023.

LAMERE, P. Welcome to Spotipy! **Spotipy**, 2014. Disponivel em: <https://spotipy.readthedocs.io/en/2.22.1/>. Acesso em: 10 Julho 2023.

MICROSOFT. **Visual Studio Code**. Disponivel em: <https://code.visualstudio.com>. Acesso em: 21 out. 2023.

MOZILLA CORPORATION. AJAX. **MDN Web Docs**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX>. Acesso em: 21 out. 2023.

OPENAI. **ChatGPT**. Disponivel em: <https://chat.openai.com/>. Acesso em: 21 out. 2023.

ORACLE. **MySQL**. Disponivel em: <https://www.mysql.com>. Acesso em: 21 out. 2023.

SANTO DIGITAL. **Quais são as principais aplicações do Machine Learning?** Disponivel em: <https://santodigital.com.br/quais-sao-principais-aplicacoes-machine-learning/>. Acesso em: 21 out. 2023.

SAVANI, B. distilbert-base-uncased-emotion. **Hugging Face**. Disponivel em: <https://huggingface.co/bhadresh-savani/distilbert-base-uncased-emotion>. Acesso em: 21 out. 2023.

SCIKIT-LEARN TEAM. **scikit-learn**. Disponivel em: <https://scikit-learn.org/stable/>. Acesso em: 21 out. 2023.

SEXTON, P. 'I Got You (I Feel Good)': James Brown's Pop Conquest Continues. **uDiscoverMusic**, 2022. Disponivel em: <https://www.udiscovermusic.com/stories/james-brown-i-got-you-i-feel-good-song/>. Acesso em: 21 out. 2023.

SOUSA, R. C. C. D. **Identificando Sentimentos de Textos em Português com o SentiWordNet Traduzido**. Universidade Federal do Ceará. Quixadá, p. 49. 2016.

SPOTIFY. **Spotify Free**. Disponivel em: <https://www.spotify.com/br-pt/free/>. Acesso em: 21 out. 2023.

SPOTIFY. **Spotify for Developers**. Disponivel em: <https://developer.spotify.com>. Acesso em: 21 out. 2023.

TANEJA, A. **Tunes For Tales**. Disponivel em: <https://tunesfortales.fun>. Acesso em: 10 Julho 2023.

TARANIFY.APP. **Taranify**. Disponivel em: <https://taranify.app>. Acesso em: 10 Julho 2023.

VASSALLO, G. Moodsic. **GitHub**. Disponivel em: <https://github.com/GuiLTheGr8/Moodsic>. Acesso em: 21 out. 2023.

VASSALLO, G. Treinamento_NaiveBayes_Moodsic.ipynb. **Google Colab**. Disponivel em: <https://colab.research.google.com/drive/1oxwa-ZXBO08W_ePt5s5ezFfTG8VDszsD?usp=sharing>. Acesso em: 21 out. 2023.