САЙТ ДЛЯ РАЗБОРА МУЗЫКАЛЬНЫХ КОМПОЗИЦИЙ «SONGLIB»

Авторы:

ученики 10 «А» класса ГБОУ

Школа №199 Маричев Фёдор

Алексеевич и Марченко Иван

Алексеевич

Руководитель – Гришина Арина

Александровна

Москва, 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

В современном мире музыка играет очень важную роль: она расслабляет, развлекает и умиротворяет. Поэтому многие музыканты и любители хотят разобраться в том, как музыкальные произведения устроены и в чём их смысл. Однако разбор и анализ песни – это очень непростое занятие, так как оно требует навыков для анализа музыкальных партий, а также наличия развитого музыкального слуха, вследствие чего это требует много времени, усилий и нервов. В связи с этим создание сайта, на котором выложены файлы с аранжировками песен, становится актуальной задачей.

Сейчас это стало ещё актуальнее, так как из-за международных санкций у многих пользователей не получается оплатить подписки на различные музыкальные (и не только) сервисы. Скачивать их взломанные версии и рисковать своим телефоном или компьютером не все желают. Существует сайт Songsterr.com, но подписку на него сейчас невозможно оплатить из-за санкций. В эту подписку включены очень многие важные функции, без которых выучить какой-либо фрагмент песни или песню целиком становится очень сложно.

Поэтому было решено создать сайт, который будет служить заменой иностранному songsterr.com и будет содержать функции, которых на других сайтах нет. Этот проект размещён на GitHub и находится в открытом доступе.

Цели и задачи

- Создать сайт для удобного разбора музыкальных композиций с возможностями регистрации, просмотра и загрузки пользователями файлов с устройством этих композиций, поиска среди загруженных на сайт файлов и добавления их в избранные (только зарегистрированным пользователям доступна загрузка файлов и добавление в избранные или удалении из них).
- Сделать его импортозамещением такого популярного иностранного сайта как songsterr.com.
- Сделать приятный для глаза дизайн.

- Сделать сайт максимально интуитивно понятным, чтобы любой пользователь мог в нём разобраться.

Методы решения этих задач

- Backend: Python, Django.
- Frontend: HTML, CSS, JavaScript.
- Инструменты разработки: Visual Studio Code, Figma.
- База данных: SQLite (по умолчанию в Django).

1. ОСНОВНАЯ ЧАСТЬ

1.1. Описание сайта

Наш сайт создавался с использованием языков Python, HTML, JavaScript и CSS. На Python написан весь backend (все функции сайта написаны на этом языке программирования с использованием фреймворка Django). На HTML написана вся разметка страниц. На CSS написан дизайн сайта и стили кнопок и надписей, а на JavaScript – ещё один стиль для надписей. Сайт содержит несколько страниц. Основные из них представлены ниже:

Основные страницы сайта:

- Главная страница. На ней располагаются кнопки для перехода: на страницы с файлами песен, на несколько страниц вперёд или назад, в меню «избранное» и на страницу для регистрации или входа пользователя. Дело в том, что сохранение, размещение файлов на сайте и добавление файлов в «избранное» доступно только зарегистрированным пользователям.

- Страницы «Регистрация» и «Вход». Эти страницы для регистрации аккаунта и входа в него (если уже существует). На них располагается форма с двумя полями: имя пользователя и пароль. Эта форма принимает пароли с длиной не менее 8 символов.

- Страница «Загрузить песню». На этой странице пользователь может загрузить на сайт файл музыкальной композиции (только в формате PDF). Эта страница доступна только зарегистрированному пользователю.

- Остальные страницы: «Избранное», «Мои песни», «Поиск», «Просмотр файла» и «Профиль» сформированы по такому же принципу, как и предыдущие**.**

Также сайт содержит «шапку» (header), позволяющую легко попасть на любую нужную страницу с помощью удобно расположенных кнопок, и «подвал» (footer) с контактной информацией.

1.2. Архитектура проекта

Структура базы данных:

Проект использует реляционную базу данных с двумя основными моделями:

- Модель UploadedFile:

  Поля: file, title, uploaded_at, user (Foreign Key к User).

  Описание: Хранит загруженные пользователями файлы.

  Пример: представлен на листинге 1.

- Модель FavoriteFile:

  Поля: user (Foreign Key к User), file (Foreign Key к UploadedFile), added_at.

  Описание: Хранит избранные файлы пользователей.

  Пример: представлен на листинге 2.

Листинг 1 — Пример модели UploadFile

```
# Модель для загруженных файлов

def validate_pdf_file(value):
  """
  Валидатор для проверки, что файл имеет формат PDF.
  """
  if not value.name.endswith('.pdf'):
    raise ValidationError('Only PDF files are allowed.')

class UploadedFile(models.Model):
  file = models.FileField(upload_to='uploads/', validators=[validate_pdf_file])  #
Добавляем валидатор
  title = models.CharField(max_length=255)
  uploaded_at = models.DateTimeField(auto_now_add=True)
  user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)

  def __str__(self):
    return self.title
```

Благодаря модели UploadedFile зарегистрированный пользователь может загружать файлы на сайт через форму. Модель обеспечивает хранение файлов и связь с пользователем, который их загрузил. А над моделью UploadFile вы также можете видеть модель для загруженных файлов. Она представляет собой валидатор для проверки, что файл имеет формат PDF. Если файл имеет другой формат, при попытке загрузить сайт выдаст ошибку: «Only PDF files are allowed.».

Листинг 2 — Пример модели FavoriteFile.

```
# Модель для избранных файлов

class FavoriteFile(models.Model):
  user = models.ForeignKey(User, on_delete=models.CASCADE)  # Связь с пользователем
  file = models.ForeignKey(UploadedFile, on_delete=models.CASCADE)  # Связь с файлом
  added_at = models.DateTimeField(auto_now_add=True)  # Дата и время добавления в
избранное

  def str(self):
    return f"{self.user.username} - {self.file.title}"  # Строковое представление объекта
```

Благодаря модели FavoriteFile пользователь может добавлять какие либо файлы песен в избранные и удалять их оттуда. Модель обеспечивает хранение информации о файлах, добавленных пользователями в избранное. Также модель обеспечивает уникальность записей, чтобы один и тот же файл не мог быть добавлен в избранное несколько раз одним пользователем.

Схема самой базы данных представлена на рисунке 1:

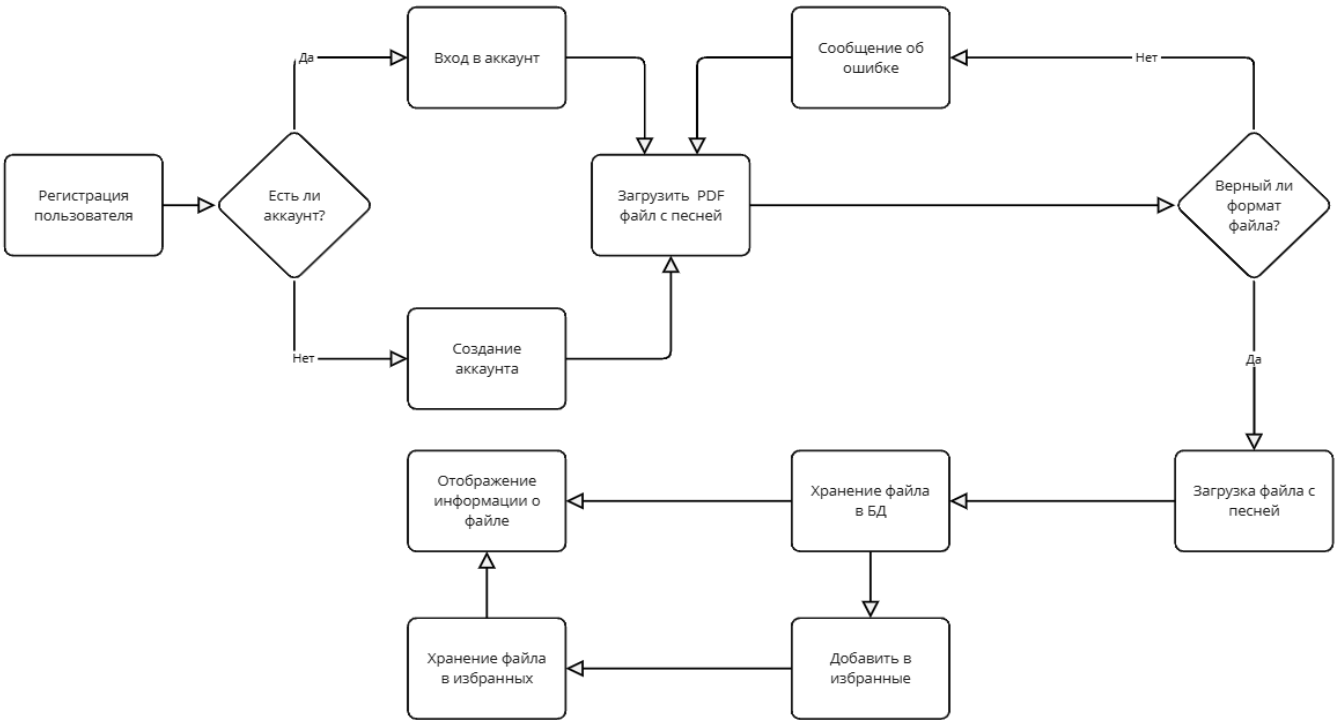

Рисунок 1 — Схема БД.

Архитектура приложения:

Проект следует архитектуре MTV (Model-Template-View), характерной для Django:

- Модели (Models)**:** Определяют структуру данных и взаимодействие с базой данных.
- Шаблоны (Templates): Отвечают за отображение данных пользователю (HTML, CSS, JavaScript).
- Представления (Views): Обрабатывают запросы пользователя и возвращают ответы (логика приложения).

Также было решено использовать шаблонизатор для динамического отображения данных для разных пользователей, упрощения работы с кодом и для использования наследования шаблонов для повторного использования кода. Использование шаблонизатора на примере страницы load_song.html представлено на листинге 3.

Листинг 3 — Пример использования шаблонизатора в load_song.html.

```
{% extends 'base.html' %}

{% block title %}Load song{% endblock %}

{% block content %}
<div class="form-container">
  <h2>Load song</h2>
  <form method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Upload</button>
  </form>
  {% if form.errors %}
    <div class="error-message">
      <p>Error: {{ form.file.errors }}</p>
    </div>
  {% endif %}
</div>
{% endblock %}
```

Так как было решено сделать загрузку файлов доступной только зарегистрированным пользователям, был использован @login_required (декоратор в Django, который используется для ограничения доступа к представлениям только для зарегистрированных пользователей). Если пользователь не авторизован (не вошёл в систему), Django автоматически перенаправит его на страницу входа, указанную в настройках проекта. Пример реализации регистрации/входа в аккаунт представлен на листинге 4.

Листинг 4 — Пример представлений для регистрации/входа.

```
# Страница регистрации
def register(request):
  if request.method == 'POST':
    form = UserCreationForm(request.POST)  # Создаем форму с данными из запроса
    if form.is_valid():
      form.save()  # Сохраняем пользователя в базу данных
      username = form.cleaned_data.get('username')  # Получаем имя пользователя
      raw_password = form.cleaned_data.get('password1')  # Получаем пароль
      user = authenticate(username=username, password=raw_password)  # Аутентифицируем
пользователя
      login(request, user)  # Входим в систему
      return redirect('home')  # Перенаправляем на главную страницу
  else:
    form = UserCreationForm()  # Создаем пустую форму
  return render(request, 'register.html', {'form': form})  # Рендерим шаблон с формой

# Страница входа
def login_view(request):
  if request.method == 'POST':
    form = AuthenticationForm(request, data=request.POST)  # Создаем форму с данными из
запроса
    if form.is_valid():
      username = form.cleaned_data.get('username')  # Получаем имя пользователя
      password = form.cleaned_data.get('password')  # Получаем пароль
      user = authenticate(username=username, password=password)  # Аутентифицируем
пользователя
      if user is not None:
        login(request, user)  # Входим в систему
        return redirect('home')  # Перенаправляем на главную страницу
  else:
    form = AuthenticationForm()  # Создаем пустую форму
  return render(request, 'login.html', {'form': form})  # Рендерим шаблон с формой
```

Как уже было сказано ранее, добавлять в избранное какие либо файлы может только зарегистрированный пользователь. Для реализации этой функции был также использован @login_required.

Структура проекта:

Приложение main содержит:

- models.py: Определение моделей базы данных.
- views.py: Обработка запросов и логика приложения.
- forms.py: Формы для взаимодействия с пользователем.
- urls.py: Настройка маршрутов (URLs).

- templates/: HTML-шаблоны для отображения страниц.
- static/: Статические файлы (CSS, JavaScript, изображения).

Взаимодействие компонентов представлено на рисунке 2:

Рисунок 2 — Схема взаимодействия компонентов.

1.3. Методика выполнения работы

Разработку сайта можно разделить на 2 основных этапа:

- Первый этап — разработка дизайна и интерфейса.
- Второй этап — реализация поставленных задач в коде.

Первый этап (разработка дизайна и интерфейса):

С помощью онлайн редактора Figma был разработан минималистичный дизайн с интуитивно понятным интерфейсом. Были Подобраны приятные фиолетовый и белый цвета. Причина выбора цветов — Фиолетовый цвет вдохновляет и мотивирует на творчество, а белый цвет расслабляет. Таким образом, дизайн максимально приятен глазу. Дизайн вдохновляет на разбор и изучение нового музыкального материала, что как раз и нужно этому проекту. Логотип же был создан с помощью встроенной нейросети в adobe photoshop.

Для реализации фронта использовались такие языки, как HTML, CSS и JavaScript. Примеры некоторых страниц сайта представлены на рисунках 3-9.

Было решено сделать так, чтобы загружать файлы с песнями сможет только зарегистрированный пользователь, поэтому для незарегистрированного пользователя на хедере не будет кнопки «load song». Добавлять в избранное, просматривать загруженные файлы и заходить в профиль смогут тоже только зарегистрированные пользователи, поэтому на главной странице для незарегистрированного пользователя нету соответствующих кнопок «favorites», «my songs» и «profile».

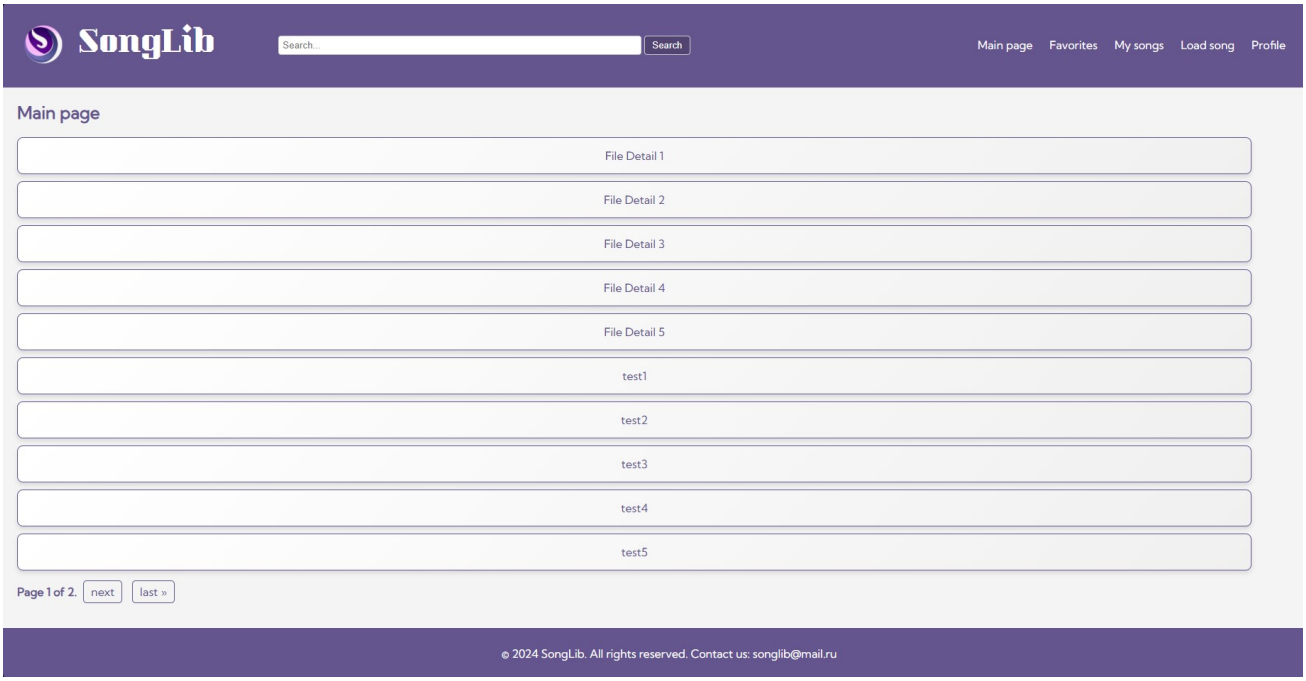

Рисунок 3 — Пример главной страницы от лица зарегистрированного пользователя.

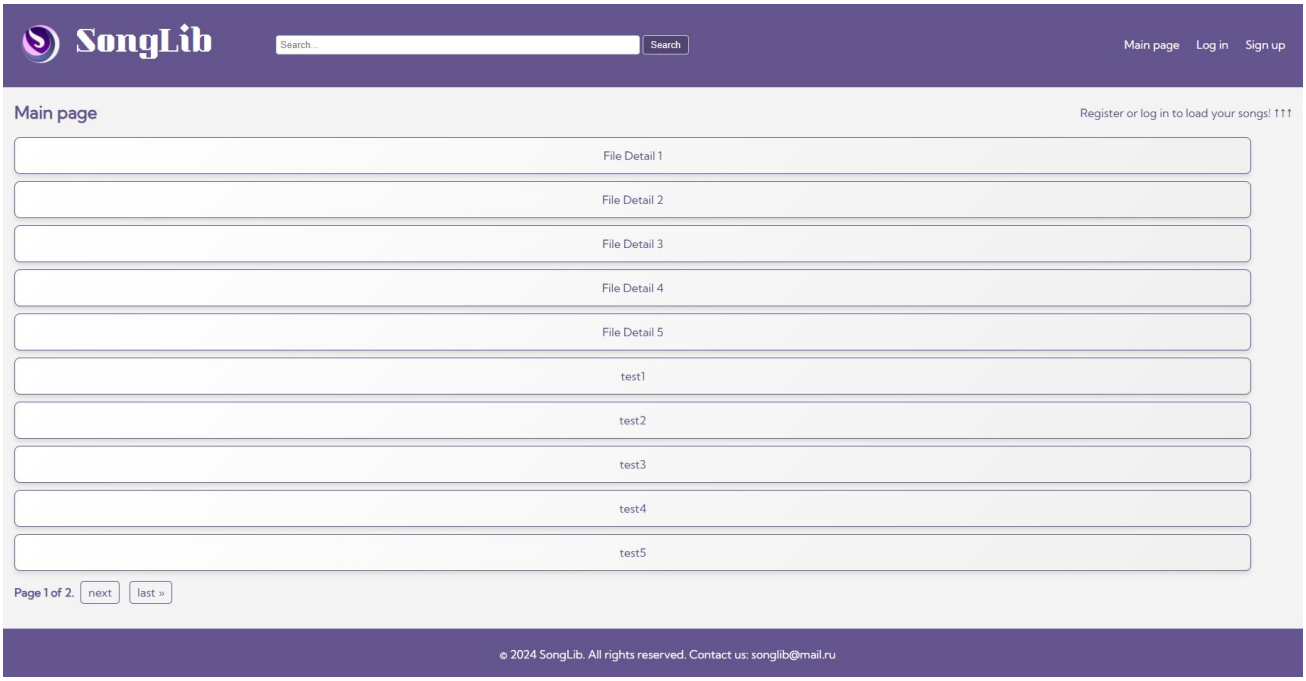

Рисунок 4 — Пример главной страницы от лица незарегистрированного пользователя.

Пример страниц входа и регистрации можно рассмотреть на рисунках 5, 6.

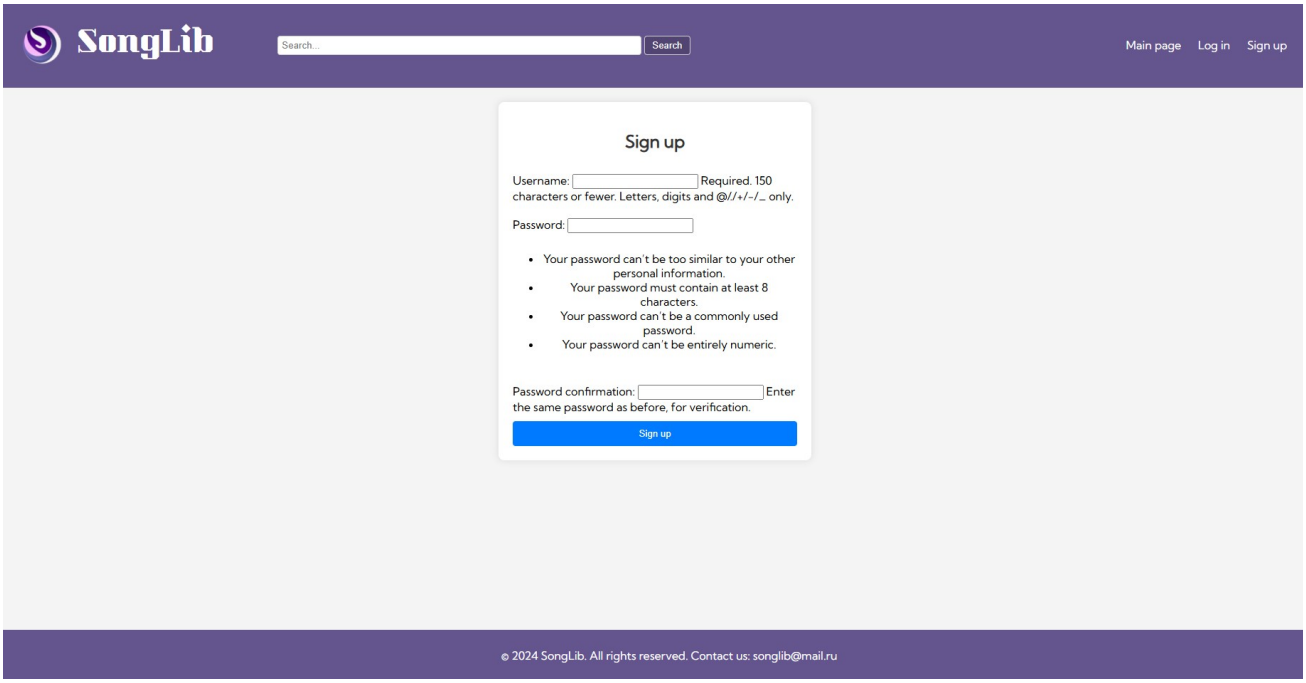

Рисунок 5 — Пример страницы регистрации

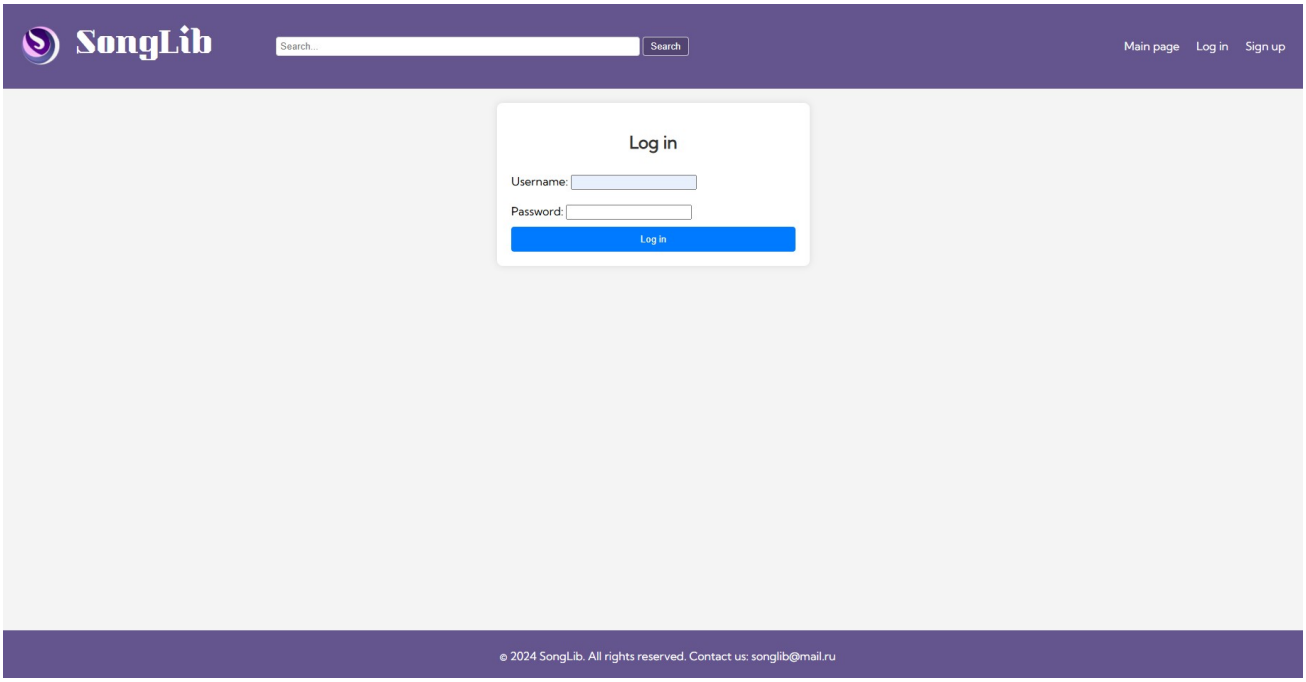

Рисунок 6 — Пример страницы входа

Выход же из аккаунта будет осуществляться на странице профиля. Пример представлен на рисунке 7.

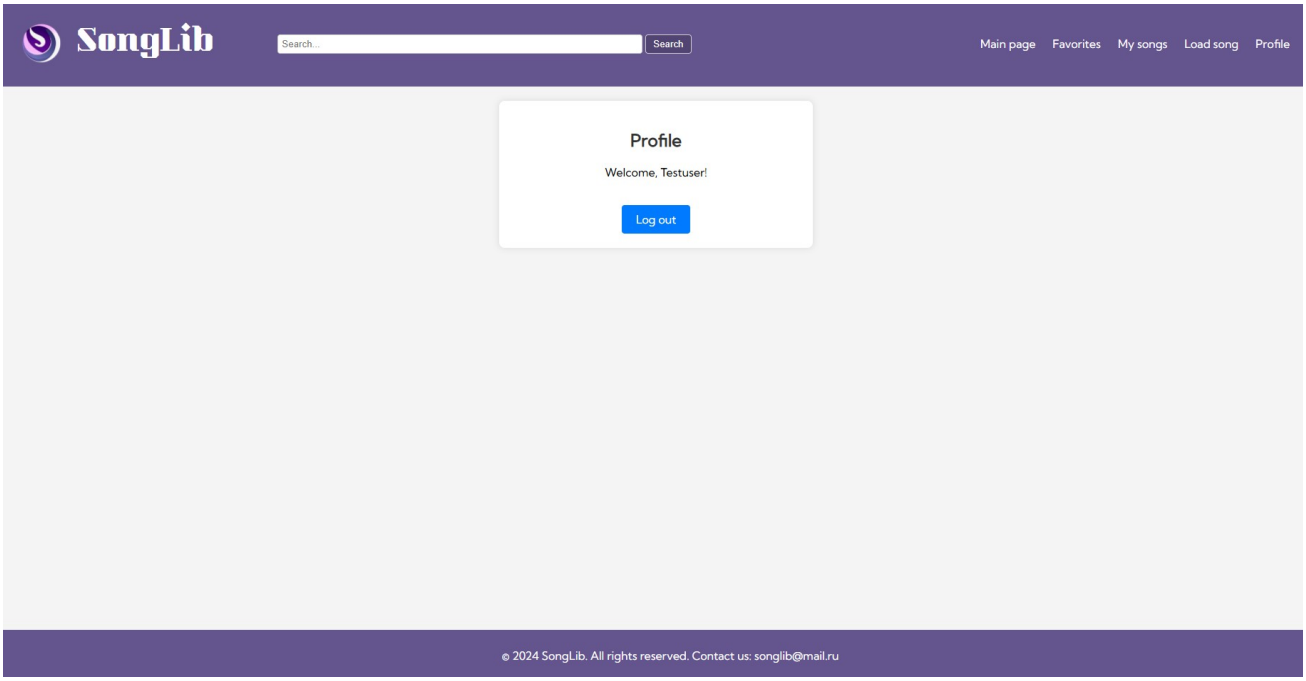

Рисунок 7 — Пример страницы профиля

Пример страницы загрузки файла с возможностью добавления его в избранные и скачивания представлен на рисунке 8. Также на этой странице можно увидеть имя пользователя, загрузившего файл, дату и время загрузки по UTC. Для примера загруженного файла взяты собственноручно написанные ноты с помощью программы guitar pro 8 в формате PDF. Переход на страницу просмотра какого либо файла осуществляется по нажатию кнопки нужного файла на главной странице. Страницы просмотра каких либо других файлов выглядят аналогично.

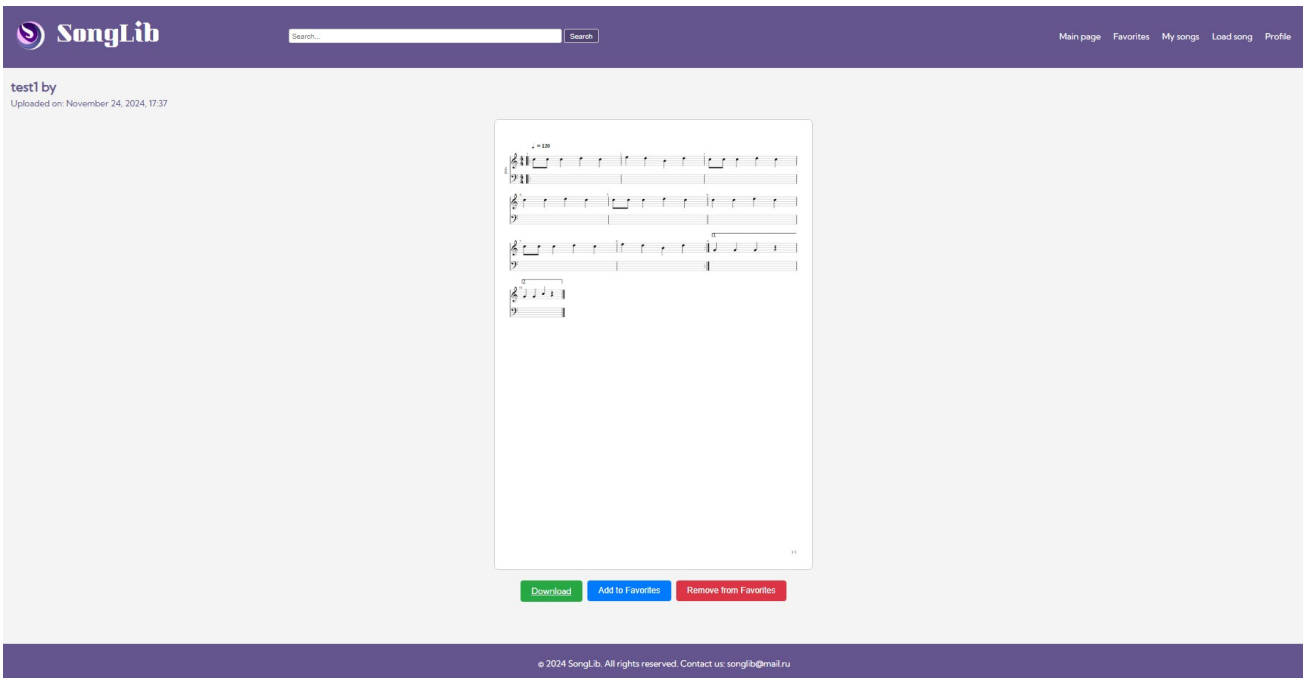

Рисунок 8 — Пример страницы просмотра файла

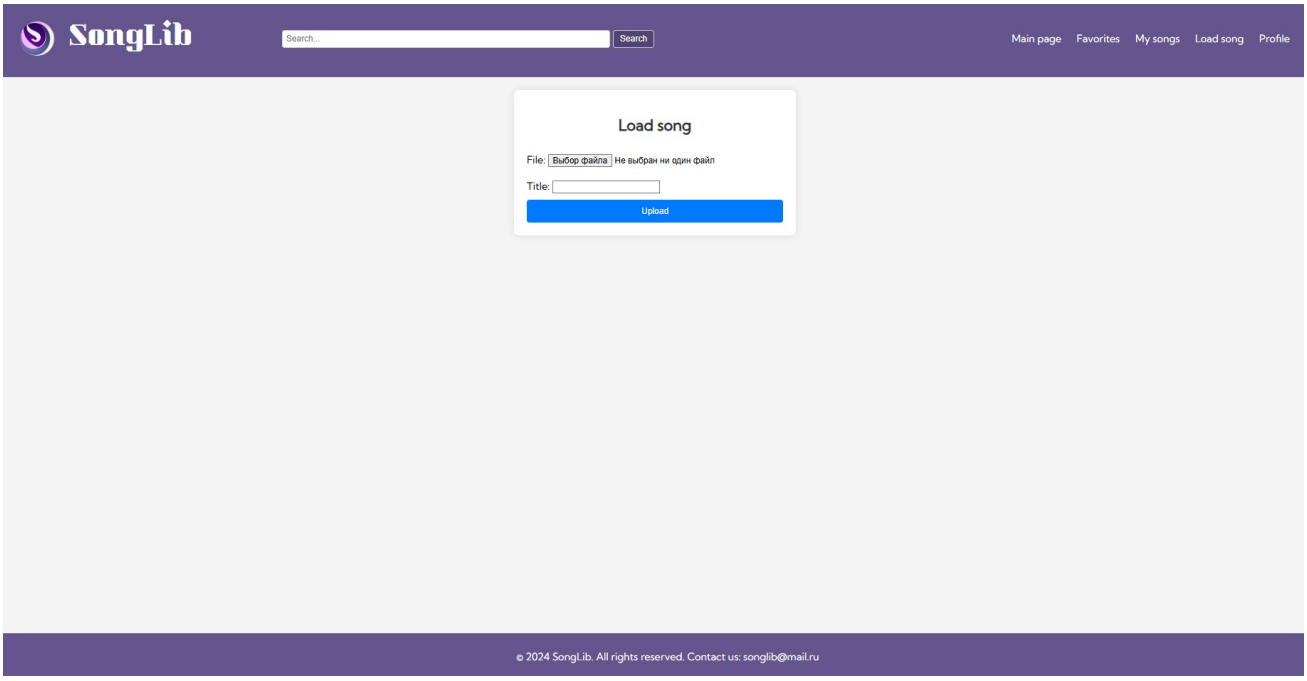

Рисунок 9 — Пример страницы загрузки песни с формой

Остальные страницы были сформированы по такому же принципу, как и выше описанные на рисунках 3-9.

Второй этап (реализация поставленных задач в коде):

Для написания кода использовался редактор Visual Studio Code (VS Code) под операционную систему Windows 10. Проект решено было написать на языке программирования Python с использованием Django. Также использовались и другие языки, например HTML, CSS и JavaScript, использующиеся для страниц и стилей.

Ниже будет кратко описан процесс написания кода для проекта.

Сперва был создан проект, и была проведена его настройка. Было создано приложение main. Приложение main было добавлено в INSTALLED_APPS в файле settings.py. Пример представлен на листинге 5.

Листинг 5 — Пример добавления приложения main в settings.py.

```
INSTALLED_APPS = [
  'main',
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
]
```

Следующим шагом была успешно настроена база данных. Были созданы модели для загрузки файлов и добавления их в избранные в models.py, после чего были применены миграции. Примеры создания моделей представлены на листингах 1, 2.

Следующим шагом была создана форма для загрузки файлов в forms.py. Пример представлен на листинге 6.

Листинг 6 — Пример добавленной формы для загрузки файлов в forms.py.

```
# Форма для загрузки файлов

class UploadFileForm(forms.ModelForm):
  class Meta:
    model = UploadedFile
    fields = ['file', 'title']

  def clean_file(self):
    """
    Валидация файла на стороне формы.
    """
    file = self.cleaned_data.get('file')
    if not file.name.endswith('.pdf'):
      raise forms.ValidationError('Only PDF files are allowed.')
    return file
```

Также, как можно заметить, было решено добавлять как можно больше комментариев для упрощения дальнейшей работы с кодом.

Следующим шагом стало создание нужных представлений в views.py, например: представление для главной страницы, для загрузки файлов, для просмотра деталей файла (сама страница открытого файла), для добавления в избранные и удаления оттуда и для самой страницы избранных файлов. Примеры добавления представлений представлены на рисунках 7-10.

Для реализации возможности загрузки файлов на сайт также были внесены нужные изменения в код остальных файлов.

Листинг 7 — Пример добавления представления для страницы деталей файла.

```
# Страница деталей файла

def file_detail(request, file_id):
  file = get_object_or_404(UploadedFile, id=file_id)  # Получаем файл по ID или
возвращаем 404
  return render(request, 'file_detail.html', {'file': file})  # Рендерим шаблон с
данными
```

Листинг 8 — Пример добавления представления для главной страницы.

```
# Главная страница

def home(request):
  files = UploadedFile.objects.all()  # Получаем все загруженные файлы
  paginator = Paginator(files, 10)  # Показываем 10 файлов на странице
  page_number = request.GET.get('page')  # Получаем номер текущей страницы
  page_obj = paginator.get_page(page_number)  # Получаем объект страницы
  return render(request, 'home.html', {'page_obj': page_obj})  # Рендерим шаблон с
данными
```

Листинг 9 — Пример добавления представления для страницы загрузки файлов.

```
# Страница загрузки файлов

@login_required
def load_song(request):
  if request.method == 'POST':
    form = UploadFileForm(request.POST, request.FILES)
    if form.is_valid():
      uploaded_file = form.save(commit=False)
      uploaded_file.user = request.user
      uploaded_file.save()
      return redirect('home')
  else:
    form = UploadFileForm()
  return render(request, 'load_song.html', {'form': form})
```

Листинг 10 — Пример добавления представления для страницы избранных файлов.

```
# Страница избранных файлов

@login_required
def favorites(request):
  favorites = FavoriteFile.objects.filter(user=request.user)  # Получаем избранные файлы
пользователя
  paginator = Paginator(favorites, 10)  # Показываем 10 файлов на странице
  page_number = request.GET.get('page')  # Получаем номер текущей страницы
  page_obj = paginator.get_page(page_number)  # Получаем объект страницы
  return render(request, 'favorites.html', {'page_obj': page_obj})  # Рендерим шаблон с
данными
```

Следующим шагом стало создание нужных маршрутов в urls.py Примеры представлены на листинге 11.

Следующим шагом стало создание уже заранее продуманных шаблонов. Так как было решено использовать шаблонизатор, в первую очередь был создан базовый шаблон base.html. Пример представлен на листинге 12. Далее были написаны хедер, футер и все заранее продуманные страницы на html.

Листинг 11 — Пример добавления маршрутов в urls.py.

```
urlpatterns = [

  path('', views.home, name='home'),  # Главная страница
  path('load-song/', views.load_song, name='load_song'),  # Страница загрузки файлов
  path('search/', views.search, name='search'),  # Страница поиска
  path('file/<int:file_id>/', views.file_detail, name='file_detail'),  # Страница
деталей файла
  path('register/', views.register, name='register'),  # Страница регистрации
  path('login/', views.login_view, name='login'),  # Страница входа
  path('logout/', views.logout_view, name='logout'),  # Страница выхода
  path('profile/', views.profile, name='profile'),  # Страница профиля
  path('favorites/', views.favorites, name='favorites'),  # Страница избранных файлов
  path('add_to_favorites/<int:file_id>/', views.add_to_favorites,
name='add_to_favorites'),  # Добавление в избранное
  path('remove_from_favorites/<int:file_id>/', views.remove_from_favorites,
name='remove_from_favorites'),  # Удаление из избранного
  path('my_files/', views.my_files, name='my_files'),  # Страница моих файлов
  path('delete_file/<int:file_id>/', views.delete_file, name='delete_file'),  # Удаление
файла
]
```

Листинг 12 — Пример добавления базового шаблона base.html.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}SongLib{% endblock %}</title>
  <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>
<body>
  <header>
    {% include 'includes/header.html' %}
  </header>

  <main class="content">
    {% block content %}
    {% endblock %}
  </main>

  <footer>
    {% include 'includes/footer.html' %}
  </footer>

  <script src="{% static 'js/main.js' %}"></script>
</body>
</html>
```

Следующим шагом стала реализация регистрации. Для этого были созданы нужные представления, шаблоны, и были внесены соответствующие изменения в код. Пример представлений представлен на листинге 4.

Все изменения в коде публиковались в специально созданном для этого публичном репозитории на сайте github.com, так что можно проследить историю обновления кода, перейдя по следующей ссылке: SwogiK/SongLib: A platform for analyzing musical compositions online.

2. РЕЗУЛЬТАТЫ И ОБСУЖДЕНИЕ

2.1. Функциональное тестирование

Было проведено функциональное тестирование сайта, и была сформирована таблица тестов. Примеры некоторых тестов представлены на таблице 1.

Таблица 1 — Примеры некоторых тестов.

№ теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод
1	Проверка перехода на страницу «Main page»	Нажатие кнопки «Main page» на хедере	Произойдёт переход на страницу «page»	Переход произошёл	Переход на страницу «Main page» работает по плану
2	Проверка отображения файлов на странице с деталями файлов	Нажатие кнопки с названием файла на странице «Main page»	Произойдёт переход на страницу с деталями файла, и он будет отображаться	Переход произошел, файл отображается	Всё работает по плану
3	Проверка работы авторизации на странице «Registration»	Нажатие кнопки «register» на странице «Registration» после заполнения всех полей в соответствии	Пользователь будет зарегистрирован, если все требования по его созданию выполнены. В любом другом случае	Пользователь создан, если в заполнении формы нет ошибок. Если же ошибки есть, то программа выдает	Авторизация работает правильно

		с требованиями	программа выдаст ошибку	ошибку.	
4	Проверка загрузки файлов на странице «Load song»	Заполнение формы в соответствии с требованиями и нажатие на кнопку «load song» на странице «Load song»	Файл будет загружен, если он сохранён в формате PDF, и если все поля заполнены корректно. В любом другом случае программа выдаст ошибку.	Файл загружен, если в заполнении формы нет ошибок. Если же ошибки есть, то программа выдает ошибку.	Загрузка файлов работает в соответствии с ожиданиями
5	Проверка работы страницы «Favorites»	Нажатие кнопки «Favorites» на хедере и просмотр страницы	Страница откроется. В избранном должны находиться добавленные туда файлы, если их туда добавляли.	Добавленные файлы присутствуют на открывшейся странице	Страница «Favorites» отображает добавленные на неё файлы корректно

Также были проведены различный другие аналогичные тесты, для которых не хватило места в таблице.

2.2. Оценочное тестирование

Также было проведено оценочное тестирование. В результате было опрошено несколько людей, которым было предложено поставить оценку удобства использования и эксплуатации сайта по шкале от 1 до 10. Примеры их оценок представлены на таблице 2.

Таблица 2 — Результаты оценочного тестирования

№ пользователя	Удобство использования	Удобство эксплуатации
1	10	9
2	7	8
3	8	9
4	9	7
5	9	9
Средняя оценка:	8,6	8,4

2.3. Результаты

Получилось успешно реализовать все поставленные задачи. Был разработан сайт для разбора музыкальных композиций. Был реализован широкий функционал, включающий загрузку файлов на сайт, добавление их в избранное, возможность регистрации и много других функций.

На полноценную реализацию проекта ушло около двух месяцев. Для достижения цели создания сайта пришлось преодолеть немало трудностей, таких как проблемы с настройкой Visual Studio Code, проблемы с настройкой проекта Django. В общем пришлось довольно много изучать различной литературы по дизайну, Python и Django. В результате получилось довольно сильно углубиться в тему создания веб-приложений на Python с помощью Django и разобраться в настройке этого фреймворка.

В результате получилось сделать удобный в использовании, как показывает таблица 2, сайт для простого разбора музыкальных композиций онлайн. Все поставленные цели и задачи были достигнуты.

ВЫВОДЫ

Был создан сайт с возможностью регистрации аккаунта, позволяющий максимально удобно и просто разбирать те или иные музыкальные композиции путём просмотра файлов с их устройством в формате PDF, которые могут загружаться зарегистрированными пользователями на сам сайт. Была реализована возможность поиска среди загруженных на сайт файлов. Была реализована возможность добавлять файлы в избранные и также возможность просмотра добавленных текущим пользователем файлов. Также была реализована возможность скачивания интересующего пользователя файла по нажатии кнопки и возможность удаления файла только самим загрузчиком. Также созданный сайт имеет множество преимуществ перед иностранными конкурентами, например, перед songsterr.com: Созданный сайт полностью бесплатный и не имеет никаких подписок, у него максимально простой интерфейс, так что каждый пользователь может с лёгкостью в нём разобраться, и приятный, расслабляющий дизайн. В результате было создано полноценное импортозамещение сайта songsterr.com. Теперь каждый желающий может загрузить разбор любимой композиции на сайт или же попробовать найти его файл на сайте, если он уже загружен кем то.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

- Силин, П. А. "Проектирование и разработка веб-приложений." М.: Издательство, 2020.
- Баранов, С. В. "Основы работы с Django." М.: Издательство, 2021.
- Степанов, И. А. "Методы тестирования программного обеспечения." М.: Издательство, 2017.
- Шабанов, Д. Ю. "Дизайн пользовательского интерфейса." М.: Издательство, 2020.
- "Django for Beginners: Build Websites with Python and Django" by William S. Vincent
- "Two Scoops of Django: Best Practices for Django 3.x" by Daniel Roy Greenfeld and Audrey Roy Greenfeld
- "Django Official Documentation" (https://docs.djangoproject.com/)
- "Mastering Django: Core" by Nigel George
- "HTML and CSS: Design and Build Websites" by Jon Duckett
- "Don't Make Me Think: A Common Sense Approach to Web Usability" by Steve Krug