Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Основы конструирования программ

Пояснительная записка

к курсовой работе

на тему

ВЕРСТКА И РАЗРАБОТКА САЙТА-ПЛЕЕРА

С ПОДБОРОМ МУЗЫКИ

Выполнила: студент группы 814301

Станкевич Анастасия

Проверила: Мельникова Е.А.

Минск, 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc40711079)

[1 ОПИСАНИЕ ПРОЕКТА 6](#_Toc40711080)

[1.1 Серверная часть 6](#_Toc40711081)

[1.2 Клиентская часть 12](#_Toc40711082)

[1.3 Текущая структура сайта beafs 12](#_Toc40711083)

[2. ОБОСНОВАНИЕ ТЕХНОЛОГИЙ 15](#_Toc40711084)

[2.1 HTML5 15](#_Toc40711085)

[2.2 CSS3 17](#_Toc40711086)

[2.3 JavaScript 19](#_Toc40711087)

[2.4 JQuery 20](#_Toc40711088)

[2.5 Python3 Django 21](#_Toc40711089)

[2.6 SQLite3 24](#_Toc40711090)

[3 ИНСТРУМЕНТАРИЙ 25](#_Toc40711091)

[3.1 Atom 25](#_Toc40711092)

[3.2 Figma 25](#_Toc40711093)

[3.3 Coggle 26](#_Toc40711094)

[3.4 FlowMapp 27](#_Toc40711095)

[3.5 dbdiagram.io 27](#_Toc40711096)

[3.6 Github 27](#_Toc40711097)

[4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ 28](#_Toc40711098)

[4.1 MVC в Django 28](#_Toc40711099)

[4.2 Схема алгоритма 29](#_Toc40711100)

[4.3 Диаграмма последовательностей 29](#_Toc40711101)

[4.4 Диаграмма состояний 29](#_Toc40711102)

[5 ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ 31](#_Toc40711103)

[5.1 Шаблоны проектирования 31](#_Toc40711104)

[5.1 Шаблон проектирования итератор 36](#_Toc40711105)

[5.2 Шаблон проектирования Состояние 38](#_Toc40711106)

[ЗАКЛЮЧЕНИЕ 42](#_Toc40711107)

[ИСТОЧНИКИ 43](#_Toc40711108)

[ПРИЛОЖЕНИЕ А 44](#_Toc40711109)

[ПРИЛОЖЕНИЕ Б 45](#_Toc40711110)

# РЕФЕРАТ

БГУИР КР 1-40 05 01-10 013 ПЗ

Скурко, И.А. Программа восстановления расфокусированного изображения / И.А. Скурко. – Минск: БГУИР, 2019. – 45 с.

Пояснительная записка 45 с., 17 рис., 4 источника, 2 приложения

DJANGO, ШАБЛОНЫ, ШАБЛОНЫ ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ, ШАБЛОН MVC, ШАБЛОН MTV, ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ, САЙТ, ВЕБ-РАЗРАБОТКА, ВЁРСТКА САЙТА

Предмет: сайт с плеером.

Объект: шаблоны для написания сайта.

Цель: вёрстка и разработка сайта-плеера.

Методология проведения работы: в процессе решения поставленных задач использованы шаблонизатор Django, шаблон MVC(MTV), шаблоны проектирования практических задач Итератор и Состояние.

Результаты работы: выполнен анализ сайтов с подобной тематикой; использована база-данных для хранения пользователей и песен; применён шаблон Состояние и Итератор для отображения данных пользователя и аудиозаписей сайта; построен плеер для проигрывания аудиозаписей из базы данных и прохождения плейлиста; спроектирован графический пользовательский интерфейс сайта; внедрена проверка формы регистрации со стороны клиента и со стороны сервера..

Модуль предназначен для генерирования плейлиста композиций, формирования пользовательски плейлистов.

Область применения результатов: досуг, поиск новой музыки.

# ВВЕДЕНИЕ

На данный момент сервисов для прослушивания музыки огромное множество. Среди них гигант Spotify, Yandex.Music с огромной базой русской музыки, «андерграундный» SoundCloud, Genius с текстом песен, Apple.Music.

Все они отличаются смелым использованием новых технологий, включая анализ большого количества данных (Spotify, Yandex.Music), развитый внешний интерфейс (Genius). К сожалению, ко всем из них подходит правило «музыка, текст, бесплатность — одно зачеркнуть». Моя курсовая направлена на изучение данных сервисов и создание сайта с музыкой и текстом.

Цели:

1. Изучить шаблон проектирования MVC.
2. Познакомиться с индустрией веб-разработки и современными техническими решениями.
3. Написать сайт с музыкальным плеером и регистрацией.
4. Создать базы данных для пользователей и аудиозаписей.
5. Применить к своему проекту принципы ООП.

Задачи:

1. Спроектировать модель сайта.
2. Создать дизайн в Figma.
3. Написать сайт на Django и JavaScript (JQuery):
   1. написать представление приложений;
   2. смоделировать базы данных пользователей, песен и плейлистов;
   3. прописать адресы на сайте и соединить с представлениями.

# ОПИСАНИЕ ПРОЕКТА

## 1.1 Серверная часть

Для создания сервера веб-сайта был использован «каркас» Django, который он создаёт автоматически, после чего можно было заполнить параметры сайта, urls, модели, представления и шаблоны.

Алгоритм создания следующий:

1. Использовать djangdo-admin  для создания папки проекта, шаблонов остальных файлов, и скрипта для управления проектом (**manage.py**).
2. Использовать  **manage.py** для создания одного или нескольких приложений.
3. Зарегистрировать в настройках эти приложения, чтобы использовать их в проекте.
4. Настроить маршруты url адресов для каждого из приложений.

Создание проекта

Создание проекта начинается с создания папки для моего сайта beafs, а после и самого проекта.

*beafs/*

**manage.py**

*beafs/*

    settings.py

    urls.py

    wsgi.py

Подпапка проекта  *beafs* это ключевая директория проекта:

* **settings.py** содержит в себе все настройки проекта. Здесь регистрируются приложения, задаётся размещение *статичных* файлов, настройки базы данных и так далее.
* **urls.py** задаёт ассоциации url адресов с представлениями. Несмотря на то, что этот файл может содержать *все* настройки url, обычно его делят на части, по одной на приложение, как будет показано далее.
* **wsgi.py** используется для налаживания связи между вашим Django приложением и веб-сервером. Вы можете воспринимать его, как утилиту.
* Скрипт **manage.py** используется для создания приложений, работы с базами данных и для запуска отладочного сервера.

Создание приложения entry.

Cледующая команда есть создание приложения *entry*, который будет размещён внутри папки beafs (команду необходимо выполнять из папки, в которой находится **manage.py**).

python3 manage.py startapp entry

Эта команда создаёт новую папку и наполнит её файлами различных частей приложения. Большинство файлов названы, исходя из их назначения (например контроллеры (views) должны находится во **views.py**, модели в **models.py**, тесты в **tests.py**, настройки административной части в **admin.py**, регистрация приложения в **apps.py**) и уже содержат некоторый шаблонный код для работы с вышеназванными объектами.

Обновлённая директория должна выглядеть следующим образом:

*beafs/*

  manage.py

*beafs/*

***entry/***

**admin.py**

**apps.py**

**models.py**

**tests.py**

**views.py**

**\_\_init\_\_.py**

***migrations/***

Кроме перечисленных выше файлов были созданы:

Папка *migrations* используется, чтобы хранить"миграции" — файлы, которые позволяют вам автоматически обновлять базу данных по мере изменения моделей.

**\_\_init\_\_.py** — пустой файл для того, чтобы Django и Python распознавали папку как Python модуль и позволяет нам использовать его объекты внутри других частей проекта.

Регистрация папки с приложением.

После создания приложения, нужно зарегистрировать его в проекте, чтобы различные утилиты затрагивали его своим действием (например при добавлении моделей в базу данных). Приложения регистрируются добавлением их названий в список INSTALLED\_APPS в настройках проекта(**settings.py**).

В файле **beafs/beafs/settings.py** было изменено:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

**'catalog.apps.CatalogConfig',**

]

Новая строка указывает на файл конфигурации приложения (CatalogConfig), который был создан в **/beafs/entry/apps.py** , когда было создано приложение.

Настройка базы данных.

На этом шаге обычно указывают базу данных для будущего проекта — имеет смысл использовать для разработки и размещённого в Сети одну и ту же базу данных, по возможности, чтобы исключить различия в поведении.

Для сайта была использована база данных SQLite, потому что не предпологается большое количество одновременных запросов на неё, а также потому, что для её не надо настраивать. База данных уже настроена в **settings.py** (подробная информация указана ниже):

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

Другие настройки проекта.

В файле присутствует две настройки, которые не нужно менять сейчас: SECRET\_KEY. Это секретный ключ, который используется Django для поддержки безопасности сайта. При раскрытии этого ключа в процессе разработки кому-либо, то необходимо будет его сменить при размещении проекта на сервер.

DEBUG. Включает подробные сообщения об ошибках, вместо стандартных HTTP статусов ответов. Должно быть изменено на False на сервере, так как эта информация очень много расскажет взломщикам.

Подключение URL-адреса.

При создании сайта, был создан файл сопоставления URL (**urls.py**) в корне проекта. Хотя можно использовать его для обработки всех URL адресов, более целесообразно подключать отдельные файлы сопоставлений для каждого приложения.

Открытый файл **beafs/beafs/urls.py** :

"""

beafs URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/1.10/topics/http/urls/

Examples:Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: url(r'^$', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: url(r'^$', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.conf.urls import url, include

2. Add a URL to urlpatterns: url(r'^blog/', include('blog.urls'))

"""

from django.urls import path

from django.contrib import admin

urlpatterns = [

path('admin/', admin.site.urls),

]

URL соотношения хранятся в переменной urlpatterns, которая является списком функций path(). Каждая path() функция или ассоциирует шаблон URLс контроллером(views) или же его с другим таким списком (во втором случае, первый URL становится "базовым" для других, которые определяются в дочернем списке). Список urlpatterns инициализирует список функции, которая, например, соотносит *admin/* с модулем admin.site.urls , который содержит собственный файл-соотноситель.

Были добавлены строки, приведённые ниже в низ файла **urls.py** , чтобы добавить новый элемент в список urlpatterns. Этот элемент содержит url() который направляет запросы с URL catalog/ к модулю  catalog.urls (файл с относительным путём **/catalog/urls.py**).

# Используйте include() чтобы добавлять URL из каталога приложения

from django.urls import include

from django.urls import path

urlpatterns += [

    path('catalog/', include('catalog.urls')),

]

После чего можно настроить корневой URL сайта : поменять (например 127.0.0.1:8000) на URL 127.0.0.1:8000/entry/; это единственное приложение. Чтобы это использовать, понадобится специальная функция (RedirectView), которая принимает первым параметром новый относительный URL на который следует перенаправлять (/entry/) когда указанный в функции url() адрес  соотносится с адресом запроса (корневой URL, в данном случае).

Были добавлены следующие строчки, тоже в конец файла:

# Добавьте URL соотношения, чтобы перенаправить запросы с корневового URL, на URL приложения

from django.views.generic import RedirectView

urlpatterns += [

path('', RedirectView.as\_view(url='/catalog/', permanent=True)),

]

В самом конце нашего URL соотносителя, можно включить размещение статических файлов.

# Используйте static() чтобы добавить соотношения для статических файлов

# Только на период разработки

from django.conf import settings

from django.conf.urls.static import static

urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

Был создан файл **urls.py** внутри папки **entry**, и вставлен следующий код, чтобы определить (пустой) urlpatterns. Сюда мы будем добавлять наши URL соотношения, по мере разработки сайта.

from django.urls import path

from . import views

urlpatterns = [

]

Подобным образом были добавлены приложение main и папки static, templates , media.

## 1.2 Клиентская часть

Все html-странички хранятся в папках templates, Django сам их находит по названию. Так как он включает всё содержимое таких папок в одну конечную папку Templates, внутри папок templates были созданы ещё папки с названием проекта.

У Django есть свои шаблоны для более структурного расположения html-кода, которые к тому же позволяют программисту не копировать код. Например, можно создать wrapper.html и помещать туда html-код нужной странички, не прописывая header и footer. Можно подключать отдельные кусочки, например, player.html для отображения плеера.

Статичные файлы, такие как картинки, иконки, css- и java-script- файлы хранятся в папках static.

Для медиа-файлов предусмотрена внешняя папка media, куда пользователи могут загружать свои картинки, музыку, используя ранее прописанные базы данных.

## 1.3 Текущая структура сайта beafs

├── beafs

│ ├── \_\_init\_\_.py

│ ├── pycache

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── db.sqlite3

├── entry

│ ├── \_\_init\_\_.py

│ ├── pycache

│ ├── admin.py

│ ├── apps.py

│ ├── forms.py

│ ├── migrations

│ ├── models.py

│ ├── static

│ │ └── entry

│ │ ├── css

│ │ │ ├── style.css

│ │ │ └── util.css

│ │ ├── fonts

│ │ ├── js

│ │ │ ├── script3.js

│ │ │ └── scriptsimple.js

│ │ └── vendor

│ ├── templates

│ │ └── entry

│ │ ├── log\_in.html

│ │ ├── sign\_up.html

│ │ ├── sign\_up2.html

│ │ └── wrapper.html

│ ├── tests.py

│ ├── urls.py

│ └── views.py

├── main

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── forms.py

│ ├── migrations

│ ├── models.py

│ ├── static

│ │ ├── beafs

│ │ │ ├── css

│ │ │ └── js

│ │ ├── main

│ │ │ └── style2.css

│ │ ├── music

│ │ │ ├── cover-song.jpg

│ │ │ ├── cover-song2.jpg

│ │ │ └── style.css

│ │ ├── player

│ │ │ ├── icons

│ │ │ ├── jquery-1.10.2.min.js

│ │ │ ├── splice5.js

│ │ │ └── style.css

│ │ ├── profile

│ │ │ ├── ava.jpg

│ │ │ ├── cover-heart.jpg

│ │ │ ├── cover-music.jpg

│ │ │ └── style3.css

│ │ └── story

│ │ ├── ashley.jpg

│ │ └── style.css

│ ├── templates

│ │ ├── main

│ │ │ ├── includes

│ │ │ │ ├── footer.html

│ │ │ │ ├── player.html

│ │ │ │ └── search.html

│ │ │ └── wrapper.html

│ │ ├── music

│ │ │ └── content.html

│ │ ├── profile

│ │ │ └── content.html

│ │ └── story

│ │ └── content.html

│ ├── tests.py

│ ├── urls.py

│ └── views.py

└── manage.py

# ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

## HTML5

HTML5  — последняя версия стандарта HTML.

Новая версия языка HTML, с новыми элементами, атрибутами и новым поведением, или набор технологий, позволяющий создавать разнообразные сайты и Web-приложения.

Семантика

1. Секции и контуры в HTML5. Контурные и секционные элементы в HTML5:  <section>,  <article>, <nav>, <header>, <footer>, <aside> and <hgroup>.
2. Использование HTML5 audio и video <audio> и <video> элементы вставляют и позволяют управлять мультимедиа контентом.
3. Формы в HTML5. Взгляд на улучшение форм в HTML5: API валидации, несколько новых атрибутов, новые значения для аттрибута type тега
4. <input> и новый элемент <output>.

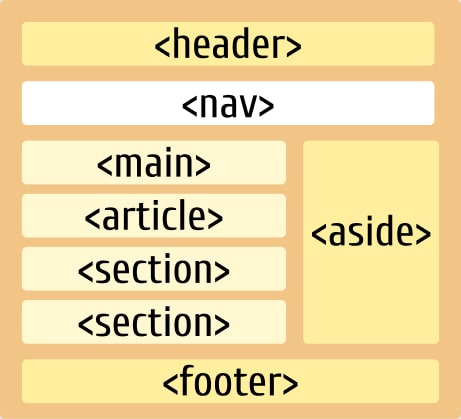


Рисунок 2 Семантические элементы

1. Новые семантические элементы. Кроме секций, медиа и форм, множество новых тегов, такие как <mark>,  <figure>,  <figcaption>,  <data>,  <time>, <output>,  <progress> , <meter>, увеличено количество валидных HTML5 элементов.
2. MathML позволяет вставлять математические формулы.
3. HTML5-совместимый парсер. Анализатор, который превращает байты HTML документа в DOM, был расширен и точно определяет поведение, чтобы даже в случае неверного HTML, исход был предсказуемым и одинаков во всех HTML5-совместимых браузерах.

Связь

1. Web Sockets. Позволяет создать постоянное соединение между страницей и сервером и обмениваться данными через него.
2. Server-sent события. Позволяет серверу отправлять события клиенту, а не по классической парадигме, где сервер может передавать данные только в ответ на запрос клиента.
3. WebRTC. Эта технология, где RTC создает возможость общения в реальном времени, позволяет подключаться к другим людям и контролировать видеоконференции непосредственно в браузере, без необходимости плагинов и внешний приложений.

Мультимедиа

Использование HTML5 audio и video. <audio>  и  <video>  элементы вставляют и позволяют управлять мультимедиа контентом.

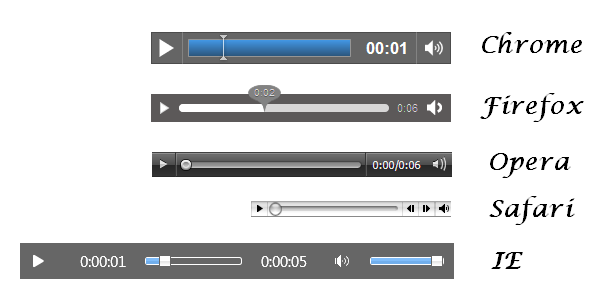


Рисунок 3 <audio> в различных браузерах

Графика и эффекты

SVG. Основанный на XML формат векторных изображений, который может быть непосредственно вставлен в HTML.

Cтилизация

1. CSS был расширен, чтобы дать возможность стилизировать элементы наиболее оптимальным способом. Его часто называют CSS3, хотя CSS больше не является монолитной спецификацией и различные модули, не все на уровне 3: некоторые на уровне 1, а некоторые на уровне 4, с промежуточными уровнями.
2. Новые способы стилизирования фона.
3. Новая возможность задать тень элемента, используя box-shadow или установление множественных фонов.
4. Лучшие границы.
5. Не только изображения можно использовать для стилизирования границы, используя border-image или его длинные формы записи, а скруглить уголки можно свойством border-radius.

Анимация

 CSS Переходы анимируют изменение состояния элемента или CSS Анимации для анимации частей страницы без запуска событий, теперь можно контролировать мобильные элементы на странице сайта.

Улучшение типографии

Авторы могут лучше контролировать типографию. Например, они могут контролировать text-overflow и перенос слов, а также тень текста и его декорированиe. Имеют возможность загрузить и применить другой шрифт правилом @font-face.

Новые презентационные макеты

Для того, чтобы улучшить гибкость дизайна, добавили: CSS мульти-колоночный макет и CSS отзывчивый блочный макет. [1]

## CSS3

В отличие от HTML, который служит для определения структуры и семантики содержимого, CSS отвечает за его внешний вид и отображение. К примеру, с помощью CSS можно изменять шрифт, цвет, размер, межстрочный интервал, разделять содержимое на колонки, а также добавлять анимацию и другие декоративные элементы. Структурировать информацию на странице помогает «display: flex;» .

Назначение CSS Flexbox

CSS Flexbox предназначен для **создания гибких макетов**. С помощью этой технологии можно очень просто и гибко расставить элементы в контейнере, распределить доступное пространство между ними, и выровнять их тем или иным способом даже если они не имеют конкретных размеров.Flexbox можно использовать как для CSS разметки целой страницы, так и её отдельных блоков.

Псевдоэлементы ::after и ::before

::after создаёт псевдоэлемент после содержимого элемента. Этот псевдоэлемент работает только совместно со стилевым свойством content, которое определяет содержимое для вставки. Не у всех элементов есть собственное содержимое, поэтому ::after не даёт результата для <img>, <input> и ряда других элементов. ::before аналогичен ::after, но вставляет псевдоэлемент до содержимого элемента.

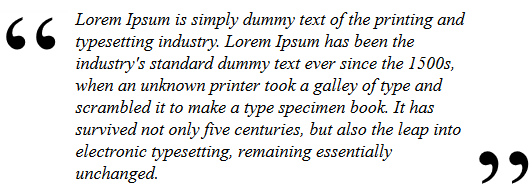


Рисунок 4 Цитата с псевдоэлементами

Блок с цитатой. ::before и ::after прекрасно подходят для оформления цитат, в частности, вывода увеличенных кавычек, которые показывают пользователям, что перед ними цитата.

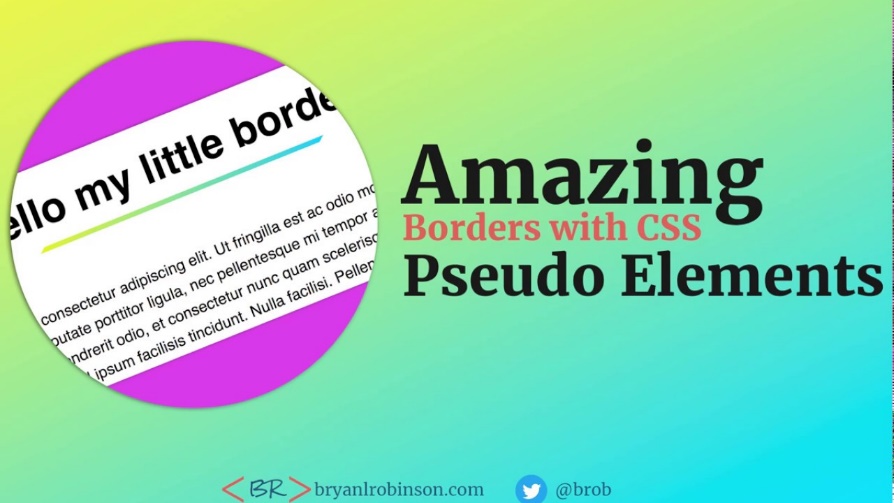


Рисунок 5 Статья про меняющие цвет, мигающие границы

Data-атрибуты в HTML5

До создания HTML5 работать с атрибутами в HTML элементах было неудобно. Приходилось использовать такие атрибуты, как**rel**или**class**. А некоторые разработчики даже создавали свои атрибуты. HTML5 предоставил возможность использования своих**data**атрибутов. Теперь довольно легко можно сохранять дополнительные данные стандартными средствами.

Как же работают дата-атрибуты? Дата атрибуты хранят в себе какие-то данные, заданные разработчиком. Они состоят из приставки data- и части, задаваемой верстальщиком (по спецификации допускаются только символы нижнего регистра и дефисы). Элемент может содержать в себе любое количество дата-атрибутов.

Данные из дата-атрибутов можно использовать в скриптах или в стилях. [1]

## JavaScript

Изначально JavaScriptбыл создан, чтобы«сделать веб-страницы живыми»*.* Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript». Разные движки имеют разные «кодовые имена». Например:

* V8 – в Chrome и Opera.
* SpiderMonkey – в Firefox.
* «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Языки «над» JavaScript

Синтаксис JavaScript подходит не под все нужды. Разные люди хотят иметь разные возможности.Это естественно, потому что проекты разные и требования к ним тоже разные.Так, в последнее время появилось много новых языков, которые транспилируются (конвертируются) в JavaScript, прежде чем запустятся в браузере.

Современные инструменты делают транспиляцию очень быстрой и прозрачной, фактически позволяя разработчикам писать код на другом языке, автоматически преобразуя его в JavaScript «под капотом».

Примеры таких языков:

1. CoffeeScript добавляет «синтаксический сахар» для JavaScript. Он вводит более короткий синтаксис, который позволяет писать чистый и лаконичный код.
2. TypeScript концентрируется на добавлении «строгой типизации» для упрощения разработки и поддержки больших и сложных систем. Разработан Microsoft.
3. Flow добавляет свою типизацию. Разработан Facebook.
4. Dart выделяется среди них, так как имеет собственный движок, работающий вне браузера (например, в мобильных приложениях). Первоначально был предложен Google, как замена JavaScript, но на данный момент необходима его транспиляция для запуска так же, как для вышеперечисленных языков. [2]

## JQuery

Следование одному и тому же шаблону: нахождение одного или несколько элементов страницы, а затем выполнение над ними некоторых действий, таких как скрытие, изменение размеров, положения, прозрачности и.т.д. — привело к созданию JQuery. Автор разработал библиотеку таким образом, что наиболее общие задачи становятся тривиальными. Например, вот так, с помощью функции $() из библиотеки, можно находить элементы на странице по различным параметрам:

|  |  |
| --- | --- |
| $("div") | вернет все div-элементы на странице. |
| $(".someBlock") | вернет все элементы с классом someBlock. |
| $("#content") | вернет элемент с идентификатором content. |
| $("#content2 div.someBlock") | вернет #content2 div.someBlock. |
| $("div:odd") | вернет div-элементы под нечетными номерами. |
| $("[value = 5]") | вернет все элементы с атрибутом value, равным 5. |

После выбора элементов, можно незамедлительно приступить к манипуляциями над ними, jQuery предоставляет для этого широкий ассортимент методов. Наличие только этих возможностей уже сделало jQuery незаменимой технологией. Однако эта библиотека позволяет делать гораздо больше.

## Python3 Django

Django — фреймворк для веб-разработкина Python

Django — это программный каркас с богатыми возможностями, подходящий для разработки сложных сайтов и веб-приложений, написанный на языке программирования Python.

Один из основных принципов фреймворка —  DRY (don't repeat yourself). Веб-системы на Django строятся из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из заметных архитектурных отличий этого фреймворка от некоторых других, например Ruby on Rails. Также, в отличие от многих других фреймворков, обработчики URL в Django конфигурируются явно, при помощи регулярных выражений, а не автоматически задаются из структуры контроллеров.

Django проектировался для работы под управлением Apache (с модулем mod\_python) и с использованием  PostgreSQL  в качестве базы данных. В настоящее время, помимо PostgreSQL, Django может работать с другими СУБД: MySQL (MariaDB), SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere и Oracle.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (Templates). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Первоначально разработка Django велась для обеспечения более удобной работы с новостными ресурсами, что достаточно сильно отразилось на архитектуре: фреймворк предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, lamoda и др.

****Некоторые возможности Django:****

1. ORM, API доступа к БД с поддержкой транзакций;
2. встроенный интерфейс администратора, с уже имеющимися переводами на многие языки;
3. диспетчер URL на основе регулярных выражений;
4. расширяемая система шаблонов с тегами и наследованием;
5. система кеширования;
6. интернационализация подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
7. «generic views» — шаблоны функций контроллеров;
8. авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и прочие;
9. система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
10. библиотека для работы с формами (наследование, построение форм по существующей модели БД);
11. встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение.

Некоторые компоненты фреймворка между собой связаны слабо, поэтому их можно достаточно просто заменять на аналогичные. Но с некоторыми (например, с ORM) это сделать не очень просто. Кроме возможностей, встроенных в ядро фреймворка, существуют пакеты, расширяющие его возможности.

На базе Django разработано много готовых решений, распространяемых под свободной лицензией, среди которых системы для управления интернет-магазинами, универсальные системы управления содержанием, а также более узконаправленные проекты.

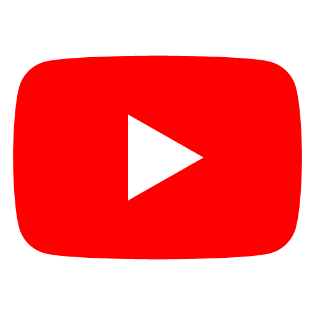
  

Рисунок 6 Сайты брендов, написанные на Django

## SQLite3

SQLite — это встраиваемая библиотека в которой реализовано многое из стандарта SQL 92. Её притязанием на известность является как собственно сам движок базы, так и её интерфейс (точнее его движок) в пределах одной библиотеки, а также возможность хранить все данные в одном файле. Позиция функциональности SQLite где-то между MySQL и PostgreSQL. Однако, на практике, SQLite нередко оказывается в 2-3 раза (и даже больше) быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер».

Всё это, собранное в один пакет, лишь немногим больше по размеру клиентской части библиотеки MySQL, является впечатляющим достижением для полноценной базы данных. Используя высоко эффективную инфраструктуру, SQLite может работать в крошечном объёме выделяемой для неё памяти, гораздо меньшем, чем в любых других системах БД. Это делает SQLite очень удобным инструментом с возможностью использования практически в любых задачах возлагаемых на базу данных.



Рисунок 7 Логотип SQLite

# ИНСТРУМЕНТАРИЙ

## Atom

Настраиваемый текстовый редактор с активным сообществом и большим количеством тем, подсветок синтаксиса, сниппетов, настроек, а также с возможностью совместной работы Teletype и пользованием Github не выходя из приложения.

В Atom есть встроенная подсветка, кроме неё были поставлены:

* atom-beautify — приводит html, css, javascript к красивому виду;
* autoclose\_html\_plus — автоматически создаёт закрывающий тег;
* autocomplete-paths — предлагает существующие имена при печатании;
* emmet — набор удобных сочетаний клавиш при работе с html, css, js;
* file-icons — добавляет иконки рядом с именем файла;
* pigments — раскрашивает hex-цвета соответствующим цветом.

## Figma

Веб-приложение для ui/ux-дизайна сайта. Можно посмотреть стили css каждого элемента и расстояния между ними.

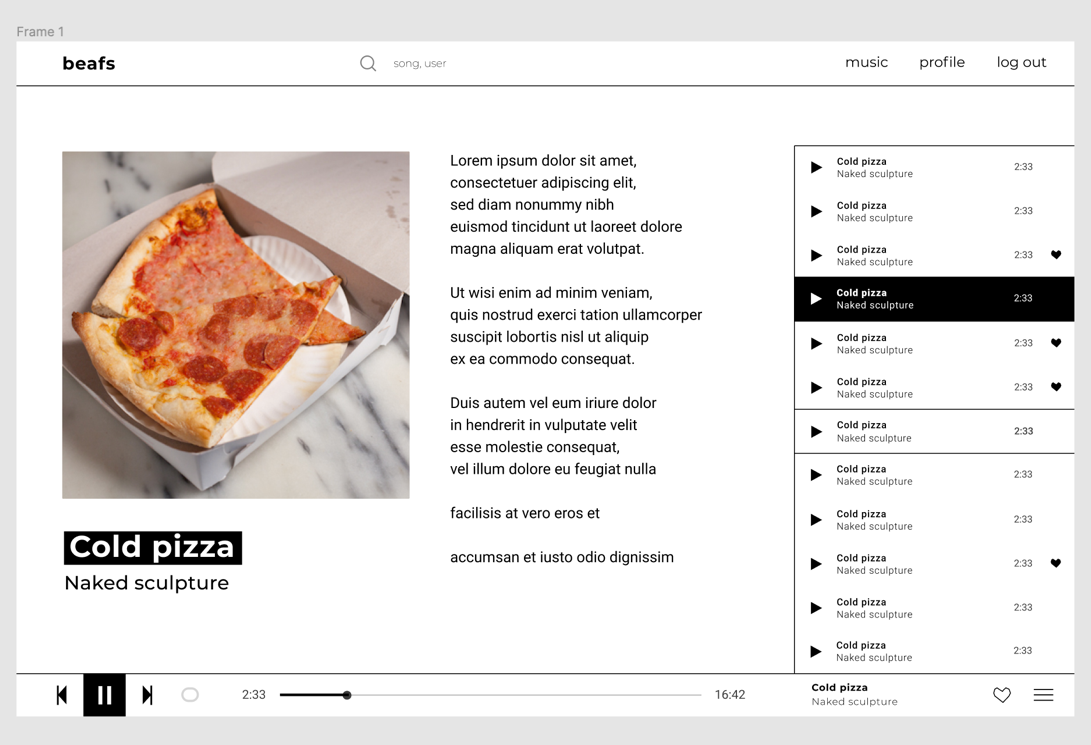


Рисунок 8 Дизайн странички с музыкой

## Coggle

Mind-карта, доступная для совместной работы нескольких человек. Наглядная и красивая.

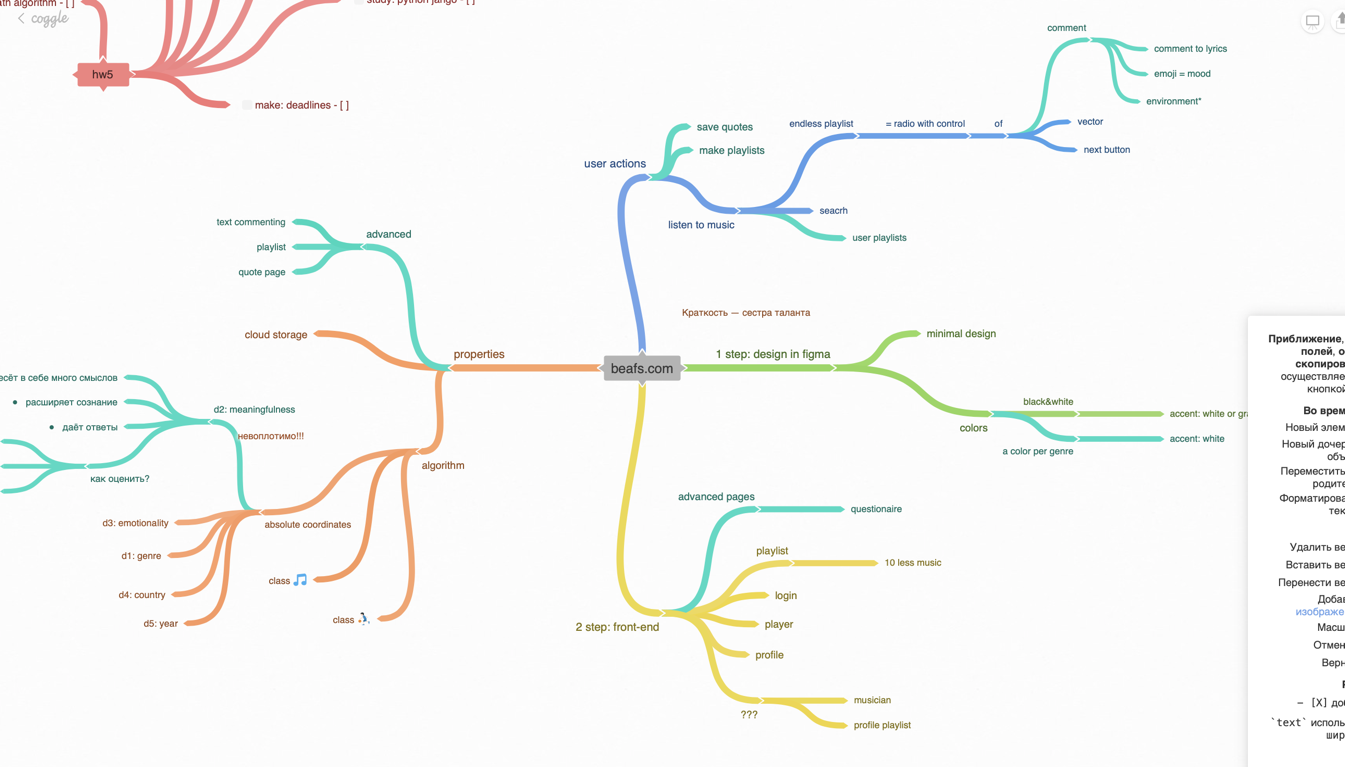


Рисунок 9 Разноцветная карта Coggle

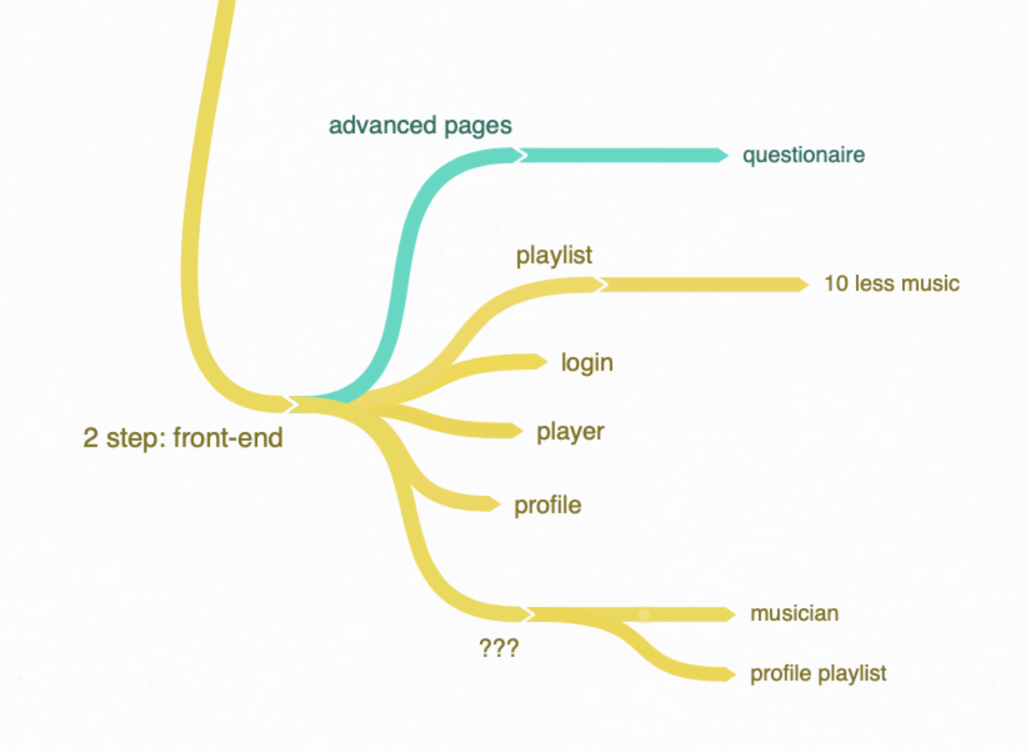


Рисунок 10 Ветка front-end в Coggle

## FlowMapp

Визуальный редактор построения сайта.

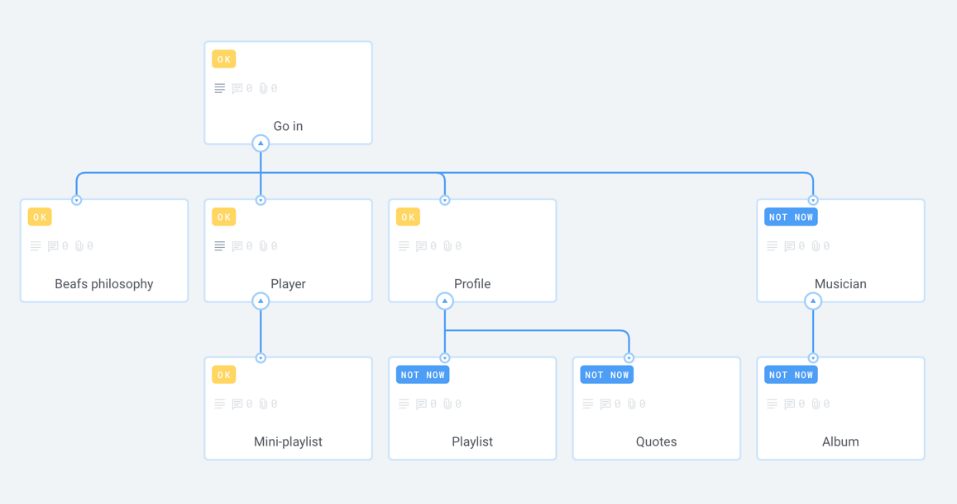


Рисунок 11 Структура сайта в FlowMapp

## dbdiagram.io

Визуальный редактор построения баз данных, основанный на sql-подобном языке.

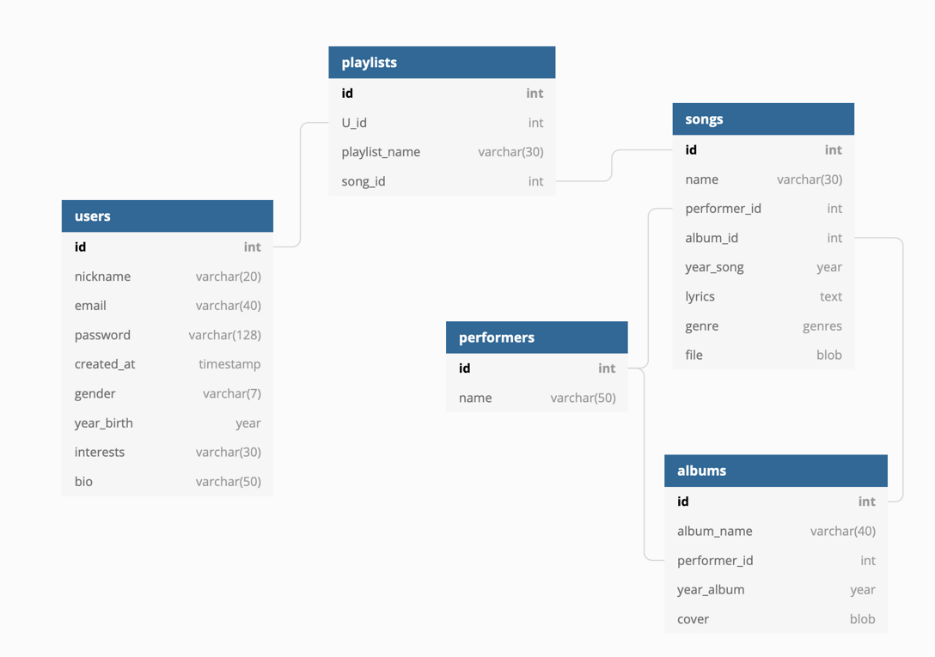


Рисунок 12 База данных в dbdiagram.io

## Github

Репозиторий для проекта.

Ссылка: https://github.com/chaoticReason/beafs

# АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ

## MVC в Django

Django поощряет слабую связанность и строгое отделение различных компонентов приложения друг от друга. Следование этой идеологии позволяет без труда вносить изменения в одну часть приложения, не затрагивая остальные. В совокупности три части – логика доступа к данным, бизнес-логика и логика отображения – составляют шаб­лон проектирования, который иногда называют Модель-Представление-Контроллер (Model-View-Con­troller – MVC). Здесь «модель» относится к уровню доступа к данным, «представление» – к той части сис­темы, которая выбирает, что отображать и как отображать, а «контроллер» – к части сис­темы, которая в зависимости от введенных пользователем данных решает, какое представление использовать, и при необходимости обращается к модели.

Примерное соответствие между буквами M, V, C и концепциями Django:

M – часть, касающаяся доступа к данным; соответствует уровню работы с базой данных в Django;

V – часть, касающаяся решения о том, что и как отображать, соответствует представлениям и шаб­лонам;

C – часть, которая передает управление некоторому представлению в зависимости от того, что ввел пользователь, реализована самим фреймворком с помощью конфигурации URL, которая говорит, какую функцию представления вызывать для данного URL.

Поскольку буква «C» реализована самим фреймворком, а самое ин­тересное происходит в моделях, шаб­лонах и представлениях, то по­ этому Django стали называть MTV-фреймворком, где:

M означает «Model» (модель), то есть уровень доступа к данным. На этом уровне сосредоточена вся информация о данных: как получить к ним доступ, как осуществлять контроль, каково их поведение, каковы отношения между данными.

T означает «Template» (шаб­лон), уровень отображения. Здесь принимаются решения, относящиеся к представлению данных: как следует отображать данные на веб-странице или в ином документе.

V означает «View» (представление), уровень бизнес-логики. На этом уровне расположена логика доступа к модели и выбора подходящего шаб­лона (или шаб­лонов). Можно сказать, что это мост между моделями и шаб­лонами. [3]

## Схема алгоритма

Смотреть Графические документы, Схема алгоритма обработки запроса в профиль.

Здесь рассматривается программа, отвечающая за представление и обработку запросов к странице. Она находится в папке views.py.

Программа обрабатывает запрос. Создаёт контекст, передавая пустую форму для изменения пароля, объект пользователя, плейлисты пользователя и сообщение об ошибке. В случае, если метод запроса POST, то обрабатывается пришедшая форма, несколько типов ошибок. Далее, если есть ошибки, программа редактирует контекст, изменяя значение сообщения об ошибках. Если их нет, обновляет и сохраняет объект пользователя с новым паролем. Именно views.py указывают, какое представление (Template) запускать.

## Диаграмма последовательностей

Смотреть Графические документы, Диаграмма последовательностей. Здесь рассматривается работа формы регистрации. Присутствуют две проверки содержимого формы: клиентская и серверная. Клиентская ­— jquery-файл. Серверная — обрабатывает форму после её отправки. В алгоритме выше рассмотрено, каким именно образом.

## Диаграмма состояний

На следующей страничке отображена диаграмма состояний формы входа. После запроса к форме входа, она подключается и ожидает действий пользователя. Когда все поля заполнены, форму можно отправить по активировавшейся ссылке.



Рисунок 13 Диаграмма состояний

# 5 ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

## **5.1 Шаблоны проектирования**

Шаблон проектирования,  или паттерн , в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Шаблоны проектирования — это не шаблоны анализа, это не описания стандартных структур (например, связанных списков). Это не определенные разработки приложений или фреймворков. По сути, шаблоны проектирования — это «описания взаимодействующих объектов и классов, предназначенных для решения общей проблемы проектирования в определенном контексте».

Иными словами, шаблоны проектирования предоставляют обобщенное, многократно применяемое решение проблем программирования, с которыми мы сталкиваемся каждый день.

Шаблоны проектирования — это не готовые классы или библиотеки, которые можно просто применить к вашей системе. Это не конкретное решение, которое можно преобразовать в исходный код. Их можно использовать для решения проблемы в различных конкретных ситуациях.

Шаблоны проектирования помогают ускорить разработку, потому что они уже проверены временем, и со стороны разработчика требуется лишь их выполнить. Шаблоны проектирования не только ускоряют процесс разработки программного обеспечения, но и представляют сложные понятия в более простой форме. Однако нужно быть острожным и не использовать шаблоны проектирования там, где это не нужно. Помимо теории, мы также приведем наиболее абстрактные и простые примеры шаблонов проектирования.

На данный момент существует 23 шаблона проектирования, которые по их назначению можно разделить на три категории:

* Порождающие шаблоны: используются для создания объектов, которые можно отделять от их системы реализации.
* Структурные шаблоны: используются для формирования больших объектных структур между множеством разрозненных объектов.
* Поведенческие шаблоны: используются для управления алгоритмами, отношениями и обязанностями между объектами.

Ниже приведен полный список шаблонов проектирования.

Порождающие шаблоны

* **Abstract Factory** (Абстрактная фабрика) - позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение.
* **Builder** (Строитель) - используется для отделения процесса конструирования сложного объекта от его представления, так что в результате одного и того же конструирования могут получаться различные объекты. Этот паттерн очень похож на абстрактную фабрику, но в нем акцентируется пошаговое конструирование объекта - в отличие от фабрики, где конструируется семейство классов.
* **Factory Method** (Фабричный метод) - предоставляет подклассам интерфейс для создания экземпляров некоторого класса.
* **Prototype** (Прототип) - используется для задания вида создаваемых объектов на основе объекта прототипа, от которого происходит передача внутреннего состояния (создаёт новые объекты путём копирования прототипа).

Структурные шаблоны

* **Adapter** (Адаптер) - предназначен для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.
* **Bridge** (Мост) - используется для отделения абстракции от ее реализации так, чтобы и то и другое можно было изменять независимо.
* **Composite** (Компоновщик) - используется для компоновки объектов в древовидные структуры для представления иерархий, позволяя одинаково трактовать индивидуальные и составные объекты.
* **Decorator** (Декоратор) - используется для динамического расширения функциональности объекта. Является гибкой альтернативой наследованию.
* **Facade** (Фасад) - представляет собой унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Паттерн фасад определяет интерфейс более высокого уровня, который упрощает использование подсистем.
* **Flyweight** (Приспособленец) - используется для уменьшения затрат при работе с большим количеством мелких объектов.
* **Proxy** (Прокси) - который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

Поведенческие шаблоны

* **Chain of responsibility** (Цепочка обязанностей) - служит для ослабления связи между отправителем и получателем запроса. При этом сам по себе запрос может быть произвольным.
* **Command** (Команда) - представляет собой действие. Объект команды заключает в себе само действие и его параметры.
* **Interpreter** (Интерпретатор) - решает часто встречающуюся, но подверженную изменениям, задачу.
* **Iterator** (Итератор) - представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов.
* **Mediator** (Медиатор) - обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга.
* **Memento** (Хранитель) - позволяет, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в это состояние.
* **Observer** (Наблюдатель) - создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.
* **State** (Состояние) - используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния.
* **Strategy** (Стратегия) - предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.
* **Template Method** (Шаблонный метод) - определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.
* **Visitor** (Посетитель) - описывает операцию, которая должна быть выполнена над каждым объектом из некоторой произвольной структуры.

Стратегия

Этот шаблон основан на алгоритмах. Вы инкапсулируете определенные виды алгоритмов, не позволяя классу клиента, ответственного за инстанциирование определенного алгоритма, узнать о фактической реализации. Пример:

<?php

interface OutputInterface

{

    public function load();

}

class SerializedArrayOutput implements OutputInterface

{

    public function load()

{

        return serialize($arrayOfData);

    }

}

class JsonStringOutput implements OutputInterface

{

    public function load()

{

        return json\_encode($arrayOfData);

    }

}

class ArrayOutput implements OutputInterface

{

    public function load()

{

        return $arrayOfData;

    }

}

Декоратор

Этот шаблон позволят добавлять новое или дополнительное поведение объекту во время выполнения, в зависимости от ситуации. Пример:

<?php

class HtmlTemplate

{

*// any parent class methods*

}

class Template1 extends HtmlTemplate

{

    protected $html;

    public function \_\_construct()

{

        $this->html = "<p>\_\_text\_\_</p>";

    }

    public function set($html)

{

        $this->html = $html;

    }

    public function render()

{

        echo $this->html;

    }

}

class Template2 extends HtmlTemplate

{

    protected $element;

    public function \_\_construct($s)

{

        $this->element = $s;

        $this->set("<h2>" . $this->\_html . "</h2>");

    }

    public function \_\_call($name, $args)

{

        $this->element->$name($args[0]);

    }

}

class Template3 extends HtmlTemplate

{

    protected $element;

    public function \_\_construct($s)

{

        $this->element = $s;

        $this->set("<u>" . $this->\_html . "</u>");

    }

    public function \_\_call($name, $args)

{

        $this->element->$name($args[0]);

    }

}

Реестр

Этот шаблон немного выделяется из общего списка, потому что он не является Порождающим шаблоном. Реестр это хэш, получить доступ к данным которого можно при помощи статических методов.

Фабрика

Это еще один очень популярный шаблон. Его название говорит само за себя: это класс, который подобен настоящей фабрике экземпляров объекта. Иными словами, предположим, что мы знаем, что существуют фабрики, производящие определенный тип продукта. Не важно, как фабрика производит этот продукт, но важно, что для каждой фабрики есть один универсальный способ запросить этот продукт.

## Шаблон проектирования итератор

В работе над сайтом мною был использован итератор. Это поведенческий шаблон проектирования, который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

Коллекции — самая распространённая структура данных, которую вы можете встретить в программировании. Это набор объектов, собранный в одну кучу по каким-то критериям.

Большинство коллекций выглядят как обычный список элементов. Но есть и экзотические коллекции, построенные на основе деревьев, графов и других сложных структур данных.

Но как бы ни была структурирована коллекция, пользователь должен иметь возможность последовательно обходить её элементы, чтобы проделывать с ними какие-то действия.

Но каким способом следует перемещаться по сложной структуре данных? Например, сегодня может быть достаточным обход дерева в глубину, но завтра потребуется возможность перемещаться по дереву в ширину. А на следующей неделе и того хуже — понадобится обход коллекции в случайном порядке.

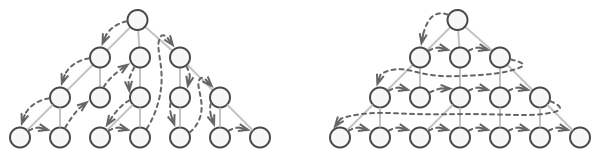


Рисунок 14 Разные способы обхода коллекции

Добавляя всё новые алгоритмы в код коллекции, вы понемногу размываете её основную задачу, которая заключается в эффективном хранении данных. Некоторые алгоритмы могут быть и вовсе слишком «заточены» под определённое приложение и смотреться дико в общем классе коллекции.

Идея паттерна Итератор состоит в том, чтобы вынести поведение обхода коллекции из самой коллекции в отдельный класс.

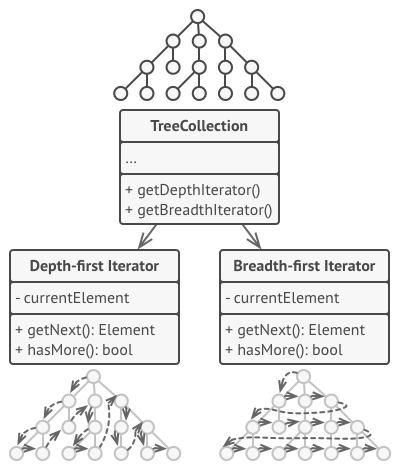


Рисунок 15 Итераторы

Объект-итератор будет отслеживать состояние обхода, текущую позицию в коллекции и сколько элементов ещё осталось обойти. Одну и ту же коллекцию смогут одновременно обходить различные итераторы, а сама коллекция не будет даже знать об этом

## Шаблон проектирования Состояние

Кроме итератора я также использовала. Состояние — это поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

Шаблон Состояние невозможно рассматривать в отрыве от концепции  машины состояний, также известной как  стейт-машина или конечный автомат.

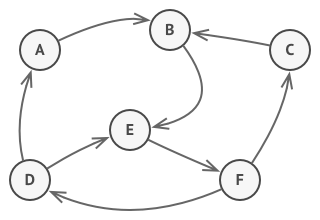


Рисунок 14 Конечный автомат

Основная идея в том, что программа может находиться в одном из нескольких состояний, которые всё время сменяют друг друга. Набор этих состояний, а также переходов между ними, предопределён и конечен. Находясь в разных состояниях, программа может по-разному реагировать на одни и те же события, которые происходят с ней.

Такой подход можно применить и к отдельным объектам. Например, объект Документ может принимать три состояния: Черновик, Модерация и Опубликован. В каждом из этих состоянии метод опубликовать будет работать по-разному:

* Из черновика он отправит документ на модерацию.
* Из модерации — в публикацию, но при условии, что это сделал администратор.
* В опубликованном состоянии метод не будет делать ничего.

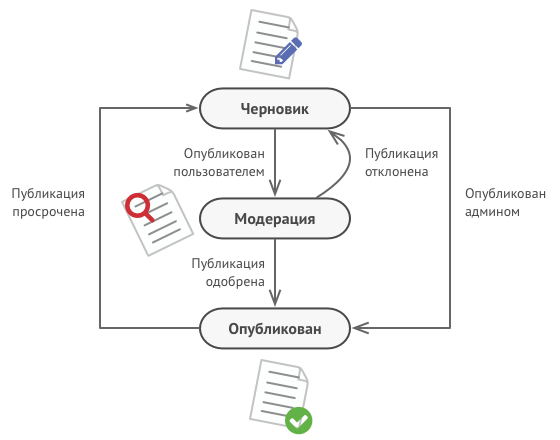


Рисунок 16 Возможные состояния документа

Машину состояний чаще всего реализуют с помощью множества условных операторов,  if либо switch, которые проверяют текущее состояние объекта и выполняют соответствующее поведение. Наверняка вы уже реализовали хотя бы одну машину состояний в своей жизни, даже не зная об этом. Как насчёт вот такого кода, выглядит знакомо?

**class** **Document** **is**

**field** state: string

*// ...*

**method** publish() **is**

switch (state)

"draft":

state = "moderation"

break

"moderation":

**if** (currentUser.role == 'admin')

state = "published"

break

"published":

// Do nothing.

break

*// ...*

Основная проблема такой машины состояний проявится в том случае, если в Документ добавить ещё десяток состояний. Каждый метод будет состоять из увесистого условного оператора, перебирающего доступные состояния. Такой код крайне сложно поддерживать. Малейшее изменение логики переходов заставит вас перепроверять работу всех методов, которые содержат условные операторы машины состояний.

Путаница и нагромождение условий особенно сильно проявляется в старых проектах. Набор возможных состояний бывает трудно предопределить заранее, поэтому они всё время добавляются в процессе эволюции программы. Из-за этого решение, которое выглядело простым и эффективным в самом начале разработки, может впоследствии стать проекцией большого макаронного монстра.

Решение

Паттерн Состояние предлагает создать отдельные классы для каждого состояния, в котором может пребывать объект, а затем вынести туда поведения, соответствующие этим состояниям.

Вместо того, чтобы хранить код всех состояний, первоначальный объект, называемый контекстом, будет содержать ссылку на один из объектов-состояний и делегировать ему работу, зависящую от состояния.

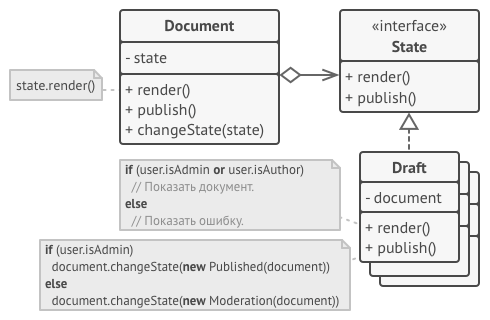


Рисунок 17 Состояние

Документ делегирует работу своему активному объекту-состоянию.

Благодаря тому, что объекты состояний будут иметь общий интерфейс, контекст сможет делегировать работу состоянию, не привязываясь к его классу. Поведение контекста можно будет изменить в любой момент, подключив к нему другой объект-состояние.

Очень важным нюансом, отличающим этот паттерн от Стратегии, является то, что и контекст, и сами конкретные состояния могут знать друг о друге и инициировать переходы от одного состояния к другому.

 Применимость

 Когда у вас есть объект, поведение которого кардинально меняется в зависимости от внутреннего состояния, причём типов состояний много, и их код часто меняется.

 Паттерн предлагает выделить в собственные классы все поля и методы, связанные с определёнными состояниями. Первоначальный объект будет постоянно ссылаться на один из объектов-состояний, делегируя ему часть своей работы. Для изменения состояния в контекст достаточно будет подставить другой объект-состояние.

Когда код класса содержит множество больших, похожих друг на друга, условных операторов, которые выбирают поведения в зависимости от текущих значений полей класса.

 Паттерн предлагает переместить каждую ветку такого условного оператора в собственный класс. Тут же можно поселить и все поля, связанные с данным состоянием.

 Когда вы сознательно используете табличную машину состояний, построенную на условных операторах, но вынуждены мириться с дублированием кода для похожих состояний и переходов.

 Паттерн Состояние позволяет реализовать иерархическую машину состояний, базирующуюся на наследовании. Вы можете отнаследовать похожие состояния от одного родительского класса и вынести туда весь дублирующий код. [4]

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой была изучена область веб-программирования, связанная с ООП, шаблонами MVC и проектирования практических задач, а также шаблонизатором Django. Сайт был написан на современных языках Django, JavaScript, JQuery.

На сайте можно зарегистрироваться и слушать музыку из базы данных, читая текст песни.

Выполнены поставленные цели и задачи:

Цели:

1. Изучить шаблон проектирования MVC.
2. Познакомиться с индустрией веб-разработки и современными техническими решениями.
3. Написать сайт с музыкальным плеером и регистрацией.
4. Создать базы данных для пользователей и аудиозаписей.
5. Применить к своему проекту принципы ООП.

Задачи:

1. Спроектировать модель сайта.
2. Создать дизайн в Figma.
3. Написать сайт на Django и JavaScript (JQuery):
   1. написать представление приложений;
   2. смоделировать базы данных пользователей, песен и плейлистов;
   3. прописать адресы на сайте и соединить с представлениями.

# ИСТОЧНИКИ

[1] Веб- документация  MDN  [Электронный  ресурс].  —

https://developer.mozilla.org/ru/docs/Web/

[2] Современный учебник JavaScript  [Электронный  ресурс].  —

https://learn.javascript.ru/intro

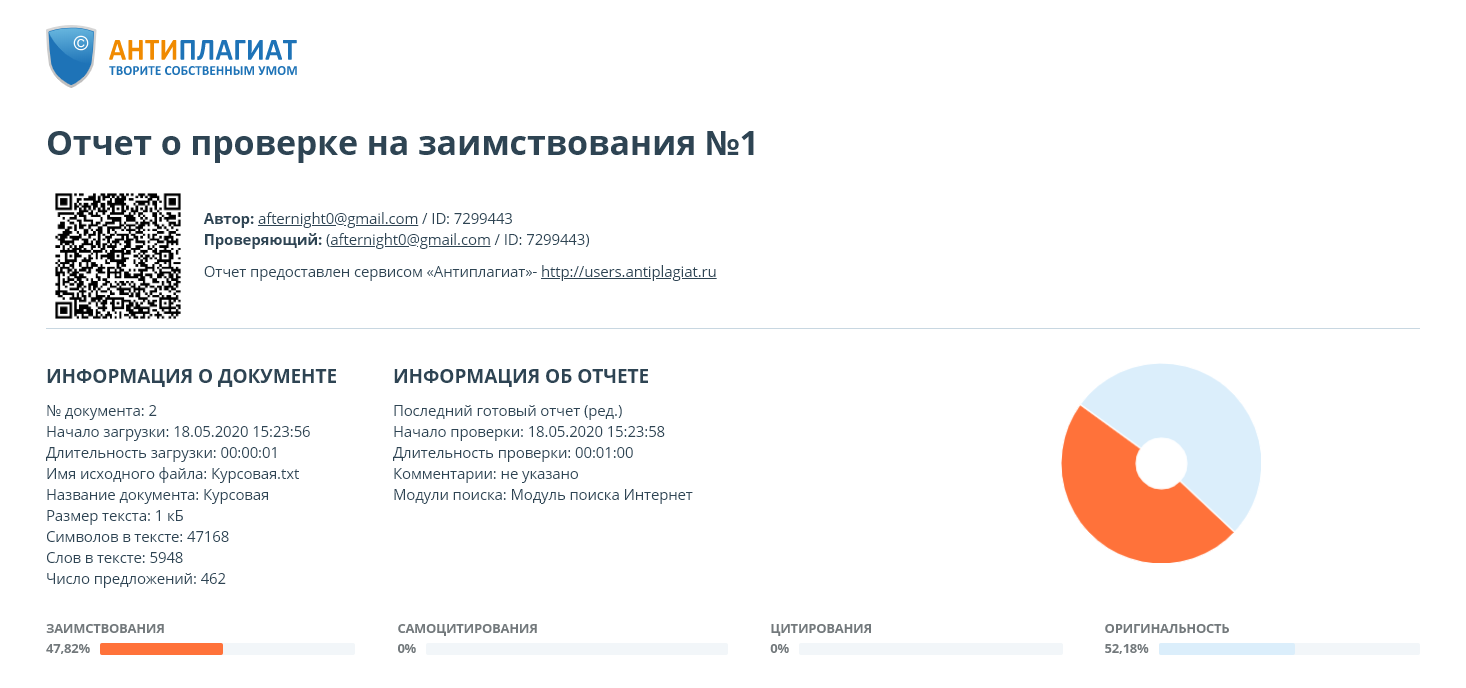
[3] Sheregeda [Электронный  ресурс].  —

http://sheregeda.github.io/

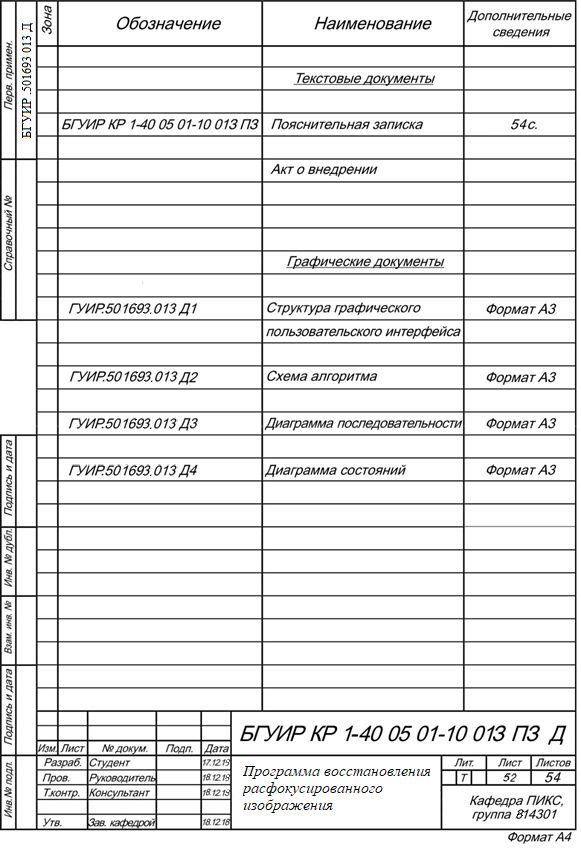
[4] Refactoring Guru [Электронный  ресурс].  —

https://refactoring.guru/ru/design-patterns/

# ПРИЛОЖЕНИЕ А



# ПРИЛОЖЕНИЕ Б



*18.05.20*

*18.05.20*

*18.05.20*

*18.05.20*

v

*46*

*50*

*46*

*Сайт-плеер beafs*

Сайт-плеер beafs