**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет „Компютърни системи и технологии”

Катедра „Информационни технологии в индустрията”

Специалност: Информационни технологии в индустрията

**ДИПЛОМНА РАБОТА**

*Тема: Разработка на уеб услуга за музика*

Студент: Александър Ивайлов Иванов

Фак. No: 501217031

Научен ръководител: гл. ас. д-р Галя Павлова

София, 2021

**СЪДЪРЖАНИЕ**

[2. Въведение в уеб системи 4](#_Toc73908448)

[2.1 История 4](#_Toc73908449)

[2.2 Предимства 5](#_Toc73908450)

[2.3 Езици за реализиране на уеб приложения 7](#_Toc73908451)

[**2.3.1 HTML** 8](#_Toc73908452)

[**2.3.2 CSS** 10](#_Toc73908453)

[**2.3.3 JavaScript** 16](#_Toc73908454)

[3. Проектиране на уеб услугата 20](#_Toc73908455)

[3.1 Технологии за реализиране на уеб услугата Soundcloud 21](#_Toc73908456)

[**3.1.1 Java EE** 21](#_Toc73908457)

[**3.1.2 Tomcat** 22](#_Toc73908458)

[**3.1.3 HTTP протокол** 22](#_Toc73908459)

[**3.1.4 Spring framework** 28](#_Toc73908460)

[**3.1.5 Maven** 29](#_Toc73908461)

[**3.1.6 JPA(Hibernate) и JDBC** 30](#_Toc73908462)

[**3.1.7 MySQL** 31](#_Toc73908463)

[**3.1.8 MVC (Model-View-Controller)** 32](#_Toc73908464)

[4. Програмна реализация 33](#_Toc73908465)

[5. Ръководство за използване 35](#_Toc73908466)

[5.1 User guide 35](#_Toc73908467)

[**5.1.1 User Controller** 35](#_Toc73908468)

[**5.1.2 Playlist Controller** 36](#_Toc73908469)

[**5.1.3 Comment Controller** 37](#_Toc73908470)

[**5.1.4 Song Controller** 37](#_Toc73908471)

[6. Заключение 39](#_Toc73908472)

[7. Използвани източници 40](#_Toc73908473)

**1. Увод**

Предмет на тази дипломна работа е проектирането и разработването на back-end частта на уеб-базирано приложение/услуга за слушане на музика. То може да бъде използвано в Интернет при наличието на уеб браузър или като самостоятелно приложение от локален компютър. За реализацията ще използваме допълнителен софтуерен продукт за симулирането на такъв браузър на локално ниво, наречен Postman. Услугата може да се разглежда като аналог на популярното приложение Soundcloud, което е безплатно за ползване, или по-популярното, но платено такова, Spotify.

То ни предоставя възможност за слушане на милиони песни без да е нужно да ги съхраняваме наше устройство, стига да имаме постоянна връзка с Интернет. Услугата осигурява лесен начин за търсене, слушане, харесване и сортиране на песни и плейлисти, следване на други потребители, както и качване на наши песни и създаване на собствени плейлисти. Допълнителна функционалност е писането и харесването/нехаресването на коментари под всяка една песен.

За реализацията на уеб услугата са използвани технологии, съвместими с Java, а именно Spring Framework, Hibernate, JPA, Tomcat Server, MySQL, както и HTTP протокола за безопасна комуникация в Интернет. Използваната среда за разработка е IntelliJ IDEA върху Windows OS.

# **2. Въведение в уеб системи**

## **2.1 История**

Всичко започва с появата на уеб услугата в интернет (1990г.). В самото начало на виртуалната мрежа съдържанието и уеб страниците са били статични - представлявали са само HTML файлове. По-скоро брошури, на които един потребител поставя ръчно някакъв текст. В следващите години на уеб (~1993г.) към брошурата вече са се добавяли и картинки. Към този момент съдържанието, заедно с презентацията му са били едно цяло – HTML. Едва през 1995г. се появява специализираният за презентацията език CSS (Cascading Style Sheets), чрез който се описва форматирането и стила на HTML елементите. По същото време (1995-1997г.) се появява и възможността за динамично генериране на съдържанието в уеб страниците чрез програмни и скриптови езици като Perl, Python, PHP, ASP, които работят на сървъра.[ 4. - Какво е CMS (Content Management System)? (superhosting.bg)]

През последвалите години се появят още основни уеб технологии, променящи статичната среда на уеб такава. Една от тях е DOM (Document Object Model) или Документен Обектен Модел (1997-1998г.), с който съдържанието на HTML документа се поставя в дървовидна логическа структура от отделни обекти. А тези обекти, или части от съдържанието, вече може динамично да се променят чрез JavaScript(1995г.) и Ajax (1999г.) заявки, които не изискват презареждане на уеб страницата. .[ 4. - Какво е CMS (Content Management System)? (superhosting.bg)]

Дотук развитието на уеб технологиите преобръща статичната брошура в динамична и интерактивна уеб страница. Съдържанието и визуалното му представяне се генерират динамично, спрямо потребителя и неговите предпочитания и взаимодействие със страницата.

Интерактивността на уеб сайтовете подхранва взаимодействието на потребителите с тях. Но освен страхотна презентация, за да се задържи интересът и подтикне взаимодействието, сайтът вече трябва да предоставя уместно, качествено и най-вече полезно съдържание. А това от своя страна налага нуждата от добавяне и редовно обновяване на такова съдържание. В ранните години на уеб това би било непосилна задача за един единствен човек, който трябва да обновява ръчно HTML кода на десетките страници в сайта, ежедневно. В създаването на съдържание за сайта вече се включват повече хора, отговарящи за съответната информация – авторите да добавят публикации и помощни статии, маркетинг специалистите да обновяват промоциите, човешки ресурси да добавят обява за работа и други. .[ 4. - Какво е CMS (Content Management System)? (superhosting.bg)]

И тук е моментът, в който една система за управление на уеб съдържанието става жизнено необходима. Една система, която да предоставя методи за лесно управление на съдържанието в сайта от много потребители.

Още през 1995-1997г. се появяват първите системи за управление на съдържанието, които са предимно за корпоративни и бизнес клиенти и са със затворен код, специална разработка.

По това време (1997-1998г.) в софтуерната индустрия вече се говори за Open Source (отворен код), а Netscape публикуват изходния код на своя софтуерен пакет Netscape Communicator. Малко след това е сформирана и инициативата Open Source Initiative, която насърчава създаването на софтуер с отворен код. Отворен код означава, че притежателят на авторските права на изходния код предоставя правата за обучение, промяна и разпространение на софтуера на всекиго и за всякакви цели. .[ 4. - Какво е CMS (Content Management System)? (superhosting.bg)]

И в началото на 2000г. започват да се появяват и първите системи за управление на уеб съдържанието (WCMS), които са с отворен код, като Drupal 2000г., WordPress 2003г., Joomla 2005г. Най известните системи с отворен код, специализирани за онлайн търговия, се появяват малко по-късно Magento 2008г., OpenCart 2010г., WooCommerce 2011г. .[ 4. - Какво е CMS (Content Management System)? (superhosting.bg)]

## **2.2 Предимства**

Уеб системите имат голям набор от предимства пред десктоп базираните системи, за които е нужно да бъдат инсталирани или разработени за конкретна операционна система и трябва да имат стриктни хардуерни изисквания, за да се подсигури тяхното правилно функциониране. Тази хардуерна зависимост както и mainframe terminal applications представлява неудобство за някои потребители, а и допълнителни разходи за набавянето на нужните изисквания за изпълнението им.[ 3. Уеб базирана система - предимства пред десктоп (dgsoft.eu)]

Уеб базираните системи предлагат единна база данни, което води до централизирана архитектура на уеб системите и позволява съхранението на всички данни на едно място. Това предотвратява например проблемите с консолидирането на данните от различните офиси на компанията или различни инсталации на съответната система (ако например се инсталира отделна база данни във всеки обект, в който се използва софтуерът). [ 3. Уеб базирана система - предимства пред десктоп (dgsoft.eu)]

Благодарение на единната база данни е възможно да получавате информация в реално време за състоянието на компанията. Единната база данни позволява и много хора да работят с един и същи запис – например актуална ценова информация, задължения на клиенти и др. Eдин потребител може да вкарва поръчките от клиенти, а друг да обработва стоката от склада, намиращ се в друг град, например. [ 3. Уеб базирана система - предимства пред десктоп (dgsoft.eu)]

Улесняват се процесите на инсталация, настройка и поддръжка. В общия случай инсталацията, настройката и поддръжката на една системасе разделя на клиентска и сървърна. Когато една система е уеб базирана, клиентската машина (работните места) не се нуждае от каквато и да било инсталация. [ 3. Уеб базирана система - предимства пред десктоп (dgsoft.eu)]

## **2.3 Езици за реализиране на уеб приложения**

Скриптовите езици бързо се превръщат в програмни езици с общо предназначение, поради тяхната функционалност. Те се използват масово там, където времето на разработчиците е по-ценно от времето за изпълнение на програмата. С тях се пишат програми, предназначени за вграждане в Web-страници. Тези програми не се компилират, а се интерпретират от браузъра при зареждането на съответната страница на клиентския компютър.

С помощта на стандартните средства на скриптовия език и на обектите, дефинирани от DOM, програмистът може да получи достъп до всеки елемент от Web-страницата и динамично да промени неговото съдържание и стилово оформяне.

Благодарение скриптовите езици за програмиране се постигат разнообразни динамични ефекти като показване и скриване на съдържание, придвижване и позициониране на обекти, анимация, създаване на големите Интернет приложения и автоматизиране.

За да се разбере по-добре същността на скриптовите езици е необходимо да се изходи от второто им наименование – езици за сценарии. Сценарият представлява последователност от операции, които потребителят може да извършва на компютъра. Сценариите обикновено се интерпретират, а не компилират. Езиците като Java, например, „минават през компилиране до байт(машинен) код, който е разбираем за компютъра и може да бъде изпълнен след подаването му към процесора. Скриптовите езици не се компилират, а интерпретират. Има „програми“, наречени интерпретатори, които изпълняват командите на дадена програма една след друга. Основната разлика също е, че при интерпретируемите езици, грешките биват откривани по време на изпълнение, докато при комплириуемите езици още по време на компилация компилаторът проверява кода за синтактиктични грешки. [2. DBorisova\_010\_ITSr.pdf (unibit.bg) – стр. 7]

Приложна програма сценарий или скрипт е програма, която автоматизира някаква задача, която без сценария потребителя трябва да извърши ръчно. Тъй като сценариите се интерпретират от изходния код динамично при всяко изпълнение, те се изпълняват значително по-бавно от готови програми, транслирани в машинен код на етап на компилиране. Динамично изпълними скриптовите езици четат инструкциите от файла на програмата на минимални блокове и изпълняват тези блокове без да четат останалия код. [2. DBorisova\_010\_ITSr.pdf (unibit.bg) – стр. 7]

Предварително компилирани скриптовите езици – прочитат програмата в началото, компилират я цялата в машинен код или някакъв вътрешен формат и след това я изпълняват.

Какви са предимствата на скриптовите езици? Скриптовите езици притежават по-сложен инструментариум и поддържат по-прогресивни техники на програмиране. Това съкращава времето за разработка на програми и ги прави по ефективни. Скриптовите езици позволяват бързо да се откриват грешки в програмата и да се реализират доработки на кода без да се изчаква приключване на компилацията. [2. DBorisova\_010\_ITSr.pdf (unibit.bg) – стр. 7]

### **2.3.1 HTML** [2. DBorisova\_010\_ITSr.pdf (unibit.bg) – стр. 7, 8]

HTML (HyperText Markup Language) или език за маркиране на хипертекст е основният език за описание на уеб страници. HTML е стандарт в Интернет, а правилата се определят от международния консорциум W3C. Текущата версия на стандарта е HTML5.

Описанието на документа става чрез специални елементи или маркери, които се състоят от етикети или тагове (HTML tags). HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница, като заглавия, цитати, раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки и имат отварящ таг <h1> и затварящ таг </h1>.

HTML е основната технология, контролираща това, което уеб браузърът показва на екрана. Основните категории команди на езика HTML, включени във версиите 3.2 и 4.01 са предназначени за: специфициране на стилови формати и управление на текстовия поток (FlowControl); включване на графични изображения (Images); създаване на хииервръзки (Links); интегриране на аудио с външни графични обекти (Sound and Maps); вмъкване на видео таг (Video) и на youtube клипове директно от тяхната страница чрез тага (Iframe); създаване на интерактивни формуляри (Forms); разделяне на документа на отделни полета (Frames); включване на външни приложения (Applet), написани на езика Java; осъществяване на връзка с външни информационни структури (CGI-script). HTML е език за маркиране, който уеб браузърите използват, за да интерпретират и създават текст, изображения и други материали. Неизпълнената характеристика за всяка точка се определя в браузъра, като могат да бъдат променяни или засилвани с допълнителни дизайнерски страници, наречени Cascading Style Sheets (CSS).

В повечето случаи HTML кодът е написан в текстови файлове и се хоства на сървъри, свързани към Интернет.

Тези файлове съдържат текстово съдържание с маркери – инструкции за браузъра за това как да се показва текстът. Предназначението на уеб браузърите е да могат да прочетат HTML документите и да ги превърнат в уеб страници. Браузърите не показват HTML таговете, а ги използват, за да интерпретират съдържанието на страницата.

Основното предимство на HTML е, че документите, оформени по този начин, могат да се разглеждат на различни устройства, а не само на екрана. Документът може да бъде правилно оформен и върху монитора на персонален компютър.

Създаването на HTML-базирана уеб страница може да се извърши с помощта на обикновен текстов редактор. Този начин изисква познаване на HTML тагове, така че те да бъдат интегрирани в текста, който ще се показва на страницата. Също така, често срещани са по-приятелски настроените инструменти, които не изискват от потребителя да притежава познания по HTML. Специализирани инструменти за създаване на HTML страници са някои от продуктите на Microsoft, FrontPage, Notepad, Notepad++, Sublime Text, NetBeans и други.

Пример 1. Обикновен HTML код

<!DOCTYPE html>

<html>

<head>

<title>Заглавие,което се визуализира в таба на браузъра</title>

</head>

<body>

<h1>Заглавие</h1>

<p>Параграф, в който може да се публива статия</p>

</body>

<html>

**Семантични елементи в HTML5:**

* Елемент <header> - представлява заглавие на елемент или група от елементи. Тага <header> позволява използването на няколко заглавия в един документ.
* Елемент <nav> - дефинира навигацията на сайта. Той е предназначен за големи блокове от навигационни връзки. Не всички връзки трябва да са в <nav> елемента, а само главното навигационно меню.
* Елемент <article> - представлява предмет/статия. Съдържанието в <article> трябва да има смисъл само по себе си и да може да се разпространява независимо и отделно от останалата част на сайта. Използва се най-често за публикация във форум; публикация в блог; новини; коментари и др.
* Елемент <section> е тематично групирано съдържание, обикновено със заглавие. Този елемент позволява влагане на нова сегмантична структура във вече съществуващ елемент.
* Елемент <aside> - представя съдържание което е странично от основното съдържание на страницата. Съдържанието в <aside> се счита за неважно и може да бъде пропуснато, ако сайта се възпроизвежда на устройство с малък екран като телефон.
* Елемент <footer> - може да съдържа информация за автора на страницата; навигация на сайта; авторските права върху използваните материали и връзки към други (подобни) публикации.

### **2.3.2 CSS** [2. DBorisova\_010\_ITSr.pdf (unibit.bg) – стр. 10-14]

CSS (Cascading Style Sheets) е език за описание на стилове, който се използва основно за описване на представянето на документ, написан на език за маркиране (HTML, XML). Официално спецификацията на CSS се поддържа от W3C (World Wide Web Consortium). CSS позволява да се определя как да изглеждат елементите на една HTML страница –

шрифтове, размери, цветове, фонове и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности. CSS е удобен начин за форматиране на текст, шрифтове, списъци, таблици и др.

Всяко CSS правило има две основни части: селектор и една или повече декларации. Ако трябва да сме още по-прецизни, всяка декларация се състои от комбинацията свойство и стойност. Синтаксисът е следния:



*Фиг.1. CSS правило*

От лявата страна на свойството се намира селекторът – това е частта от правилото, указваща за кой елемент от документа трябва да се приложат стиловете.

В примера са избрани всичките елементи на body. От дясната страна на селектора се намира декларацията, която представлява комбинация от свойството и стойност. Свойството е стилов атрибут, на който се задава стойност. Всяко свойство има стойност. CSS декларацията винаги завършва с точка и запетая, а групите от декларации са заградени от фигурни скоби.

Най-голямото улеснение, което предлага CSS е свързано с контрола на голям набор HTML документи, като контролирането на външния вид на страниците може да става по няколко начина.

Пример 2. CSS код с добавени стилове в </head> частта

<!DOCTYPE html>

<html>

<head>

<title>CSS като отделен файл </title>

<meta

http-equiv=

Content-Type

content="text/html; charset=UTF-8" />

<link type="text/css"/>

</head>

<body>

rel="stylesheet"

href="external.css"

<h1>Заглавие</h1>

<p >Параграф, в който може да се публива

статия</p>

</body>

<html>

Пример 3. CSS код с добавени стилове в </head> частта.

<!DOCTYPE html>

<html>

<head>

<title> вграден CSS файл </title>

<meta http-equiv= Content-Type content="text/html;

charset=UTF-8" />

<style type=”text/css”>

</style>

</head>

<body>

<h1>Заглавие</h1>

<p >Параграф, в който може да се публива статия</p>

</body>

<html>

Пример 4. CSS код със стилове в самите HTML елемент

<!DOCTYPE html>

<html>

<head>

<title> вграден CSS файл </title>

<meta

http-equiv=

Content-Type

content="text/html; charset=UTF-8" />

<style type=”text/css”>

</style>

</head>

<body>

<h1>Заглавие</h1>

<p style=”text-align: left;”>Параграф,

в който може да се публива статия</p>

</body>

<html>

При наличие на повече от един стил, специфициран за HTML елемент, ще се използва този, който има най-висок приоритет (четири е най-високият), както следва:

* + - 1. Стилът на браузъра по подразбиране.
      2. Външните стилове.
      3. Вътрешните стилове.
      4. Стиловете, вградените в самите HTML елементи.

Първоначално HTML етикетите са били създадени, за да определят съдържанието на документа. Или иначе казано, идеята била следната: таговете трябвало да “казват”: “Това е заглавие”, “Това е параграф”, “Това е таблица”, използвайки синтаксис от рода <h1> (за големи заглавия), <p> (за параграф). Постепенно с добавянето на нови спецификации към HTML и с увеличаване на изискванията към визията на сайта, ставало все по-трудно да се създават страници само с него.

Друго предимство на CSS е, че спестява работа и по-точно писане на код. Тъй като можем във външен файл да съхраним повтарящите се форматирания и така да не пишем всеки път атрибутите за тях. Освен това, по този начин при евентуална промяна на изгледа на едно форматиране, се налага само промяна в съответния стил. Например, ако имаме 10 страници с шрифт Arial, размер 2, цвят черен, и искаме да променим цвета, но не сме задали стил, то ще се наложи ръчно да го направим във всичките страници поотделно. Задавайки стил за конкретното форматира в отделен файл, трябва само да се поправи там – на едно единствено място.

**Синтаксисът** на CSS е съставен от три части: селектор (selector), характеристика (property) и стойност (value).

**Селекторът** е нормален HTML елемент (таг), който трябва да се промени, характеристиката е атрибутът за смяна, като всяка характеристика си има стойност.

Пример 5.

selector {property: value}

Характеристиката и стойността са отделени с двоеточие и са заградени с фигурни скоби:

Пример 6.

body {color: black}

Ако стойността представлява низ съдържащ няколко думи – необходимо е да се огради с кавички:

Пример 7.

p {font-family: "sans serif"}

Ако трябва да се специфицира повече от една характеристика, е необходимо да се разделят с “;”.

Пример 8.

p {text-align:center;color:red}

Селекторите с еднакви характеристики могат да се групират по между си и така да се спести писане на код. Това става като се разделят със запетая. В следващия пример са групирани всички header елементи. След прилагането на стила съответния текст ще бъде оцветен в зелено:

Пример 9.

h1,h2,h3,h4,h5,h6 { color: green }

CSS притежава още едно важно свойство, така нареченият клас Селектор (class selector). С него може да се зададат различни стилове за един и същи HTML елемент в уеб страницата. Ако например имаме два параграфа, единият от тях е с дясно подравняване а другият с централно, с клас селектор това би изглеждало така:

Пример 10.

p.right {text-align: right}

p.center {text-align: center}

Друга полезна опция в CSS e id селекторът. Той се различава от класа селектор, по това че последния може да се приложи върху няколко елемента върху страницата, а id селекторът е приложим само за един. Той трябва да е уникален в рамките на уеб страницата. Долу е дефиниран стил, който свързва елемента <p> с id стойността “paragraph1”

Пример 11.

p# paragraph1{text-align: center; color: red }

Добра практика е като се пише код да се оставят коментари, каквато възжможност притежава и CSS. Коментарът започва с "/\*" и завършва с "\*/". Всичко между тях се игнорира от браузъра.

**2.3.3 JavaScript**

**Javascript** е скриптов език за програмиране, създаден първоначално от *Netscape Communications Corporation* и предназначен за създаване на итерактивни Web страници. Базата за създаване и развитие на езика е една спецификация, носеща името **ECMA-262**. Предназначението на ***Javascript*** е да улесни динамичната обработка на HTML документи при клиента. Скрипт кодът се включва като част от HTML кода. Изпълнението на скрипта става след интерпретация от навигатора на клиента при изтегляне на страницата от сървъра. [5. Уроци по Java Script | Реферат от 'Информационни технологии (referati.org) стр. 1]

Динамично управление на графика, формуляри, хронологичния списък на страниците са сред най-често използваните приложения на скрипт код, написан на ***Javascript***.

***Javascript*** е първият скриптов език за програмиране в среда Web и вече не е единствен. С подобни функции и възможности е ***VBScript*** *(Visual Basic Script)*, създаден от *Microsoft*.

Сходството в наименованията между езиците ***Java*** и ***Javascript*** не ги прави еднакви, нещо повече те са различни в много отношения. Не бива да се смята, че ***Javascript*** е някакво подмножество на езика ***Java***, те просто са два различни езика за програмиране.

Подобно на всеки друг скриптов език чрез ***Javascript*** се създава скрипт-код, който подлежи на интерпретиране. [5. Уроци по Java Script | Реферат от 'Информационни технологии (referati.org) стр. 1]

**Javascript и Java**

Различията между ***Javascript*** и ***Java*** в най-общи линии се заключават в следното:

**Таблица 1**. [5. Уроци по Java Script | Реферат от 'Информационни технологии (referati.org) стр. 1]

|  |  |
| --- | --- |
| **Javascript** | **Java** |
| Код интегриран в HTML страница. | Кодът не се интегрира в HTML страница (applet). |
| Кодът се интерпретира от навигатора в момента на изпълнение. | Първичният код се компилира преди изпълнение. |
| Програмният код е упростен, но и с много ограничения. Обектно-ориентиран. | Сложен обектно-ориентиран език за програмиране от високо ниво, почти без ограничения. |
| Достъп до обектите на навигатора. | Липсват средства за достъп до обектите на навигатора. |

***Javascript и HTML***

Приложението на ***Javascript*** изисква предварителни познания и по HTML. Написаният код на ***Javascript*** се вмъква директно в кода на HTML страницата по следния начин:

Пример 12. [5. Уроци по Java Script| Реферат от 'Информационни технологии (referati.org) стр. 1]

|  |
| --- |
| **<html>**  **<head>**  **<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">**  **<title>Javascript</title>**  **</head>**  **<body>**  **<p>Текстов параграф в HTML код.</p>**  **<p>**  **<script language="JavaScript">**  **<!-- hide from old browsers**  **document.write("Текст, изведен на екран чрез JavaScript код!")**  **// -->**  **</script>**  **</p>**  **<p>Текстов параграф в HTML код.</p>**  **</body>**  **</html>** |

JavaScript кодът се вмъква в HTML документа между двойката елементи **<SCRIPT>** и **</SCRIPT>.** Този елемент е въведен в стандарта HTML 3.2 и HTML 4.01. Когато срещне тагът **<SCRIPT>**, браузърът разбира, че трябва да спре интерпретирането на HTML кода и да започне да обработва скрипта, намиращ се между **<SCRIPT>** и **</SCRIPT>**. Този скрипт не е задължително да бъде написан на JavaScript. Има и други езици за  писане на скриптове, например VBScript. Затова когато се пише отварящият таг за скрипт, трябва да се укаже на браузъра на какъв език ще бъде скрипта. Ако този език е JavaScript, трябва да се напише **<SCRIPT LANGUAGE="JavaScript">.** [6. Уроци по JavaScript | Реферат от 'Информационни технологии (referati.org) стр. 1]

Код на ***Javascript*** може де се вмъква динамично в момента на интерпретиране на HTML страница чрез следната команда:

|  |
| --- |
| **<SCRIPT SRC="javascript\_file.js" TYPE="text/JavaScript">** |

където, файлът **javascript\_file.js** съдържа код на ***Javascript***.

Основните **характеристики** на езика ***Javascript*** се заключават в следното:

Няколко основни типа данни:

* + Цифров тип, без ясно разграничаване между цели и реални;
  + Символен тип;
  + Логически тип;
  + Нулев тип;

• Слаба типизация в езика - една променлива може във всеки един момент да промени своя тип;

• Синтактична близост на езика с езика за програмиране С, почти същите оператори и инструкции;

• Езикът е основан на недоразвит, начален обектен модел, липсва понятието клас и не поддържа унаследяване;

• Няколко базови обекта за обработка на дати, таблици и математически пресмятания;

• Ограничени възможности за манипулиране на входа и изхода, и почти никакви за въздействие с мрежови операции;

**Елементи** на езика **Javascript** са следните:

• Лексика • Коментари • Типове данни • Променливи • Ключови думи

[6. Уроци по Java Script | Реферат от 'Информационни технологии (referati.org) стр. 2]

Като език за програмиране, JS поддържа **обектно-ориентираното програмиране**

и е един от малкото езици, които позволяват да създадете приложение във всяка една възможна област - от сайтове и мобилни приложения до сървърни и desktop приложения.

Най - популярните среди за интерпретация на JavaScript са уеб браузърите, които използвате всеки ден - Chrome, Firefox, Internet Explorer и т.н. Когато се зарежда даден уеб сайт е много вероятно той да съдържа JavaScript файлове, които ще се изпълнят при отварянето на сайта и ще направят изживяването при разглеждането на сайта по-приятно и динамично. Твърде вероятно е ако в сайта има падащи менюта, анимации, регистрация на потребител, дразнещи реклами, то те да са реализирани именно чрез използването на езика JavaScript.

Често се чува изразът, че даден код се "изпълнява на клиента" - това ще рече, че JavaScript кодът се изпълнява в browser-a на клиента, който играе ролята на клиент, или приемник. За да има приемник, трябва да има и предавател. В технологичния свят тези предаватели се наричат сървъри. Можем да си представим сървърите като едни много мощни компютри, до които имат достъп много хора. Всички сайтове стоят на подобни сървъри - т.е. файловете, които помагат на сайта да изглежда такъв, какъвто го виждаме - картинки, текстове, JavaScript файлове, се намират на конкретен сървър. Нашият browser (клиент) се свързва към сървъра, на който се намира сайта, и сървърът изпраща обратно необходимите файлове, за да се визуализира пред нас. Може да се представи, че browser-а е радиото в колата, на което се слуша любимата програма, а сървърът е сградата, откъдето се излъчва това предаване, макар и комуникацията между двете да протича по различен начин.

Другият популярен интерпретатор за JavaScript e NodeJS. Може да се представи като приложение, което инсталираме на компютъра си и той започва да разбира от JavaScript по същия начин, по който и web browser-a разбира. По този начин е възможно да се изпълнява JavaScript код директно на компютъра, без необходимостта от browser. Както се спомена по-рано, сървърите са просто по-мощни компютри. Те започват да разбират от JavaScript по същия начин, по който и компютрите - просто им се инсталира NodeJS.

# **3. Проектиране на уеб услугата**

В настоящата глава се разглеждат различните аспекти на Web-програмирането, неговите предимства и недостатъци, както и причините за неговата популярност.

Всеки потребител в Интернет е виждал Web-базирани e-mail системи като mail.yahoo.com, mail.bg и abv.bg. Те са чудесни примери за Web-приложения.

Както стана ясно, Web-приложенията представляват услуги, които работят на някакъв Web-сървър и предоставят на потребителите Web-базиран интерфейс, който се визуализира от Web-браузърите им. Комуникацията между потребителските Web-браузъри и Web-приложенията се основава на заявки и отговори и се извършва по протокол HTTP. Когато потребителят напише адреса на някое Web-приложение, неговият Web-браузър изпраща на съответния Web-сървър заявка за достъп до това Web-приложение и получава динамично генериран отговор във вид на HTML или друг формат, който браузърът разбира.

**Архитектура на Java-базираните Web-приложения**

Java-базираните Web-приложения могат да бъдат разделени на две части:

- сървърска част – представлява съвкупност от Java компоненти и JSP страници, които обработват получените от потребителя данни и в зависимост от тях динамично генерират HTML документи, CSS стилове и JavaScript код.

- клиентска част – представлява съвкупността от динамично генерираните HTML документи, CSS и JavaScript код, които се визуализират от Web-браузъра на потребителя и изграждат потребителския интерфейс на приложението.[ 1. Разработка на Web-приложения с Java | Реферат от 'Информационни технологии - Страница 11 (referatite.org)]

Разглеждайки Web-приложенията по този начин, те могат да бъдат определени като многопотребителски клиент-сървър приложения, предназначени за работа в Интернет.

Когато, обаче, се разработва по-сложно програмно приложение, се използват множество готови програмни елементи и класове, които образуват специализираните програмни среди. Най-широко разпространение в момента получава специализираната програмна среда, разработена от Sun Microsystems J2EE, с която е разработено и моето уеб приложение.

За реализирането на моето приложение съм използвал Spring Framework - технологична рамка с отворен код (open source), които улеснява изключително много работата в създаването на Java уеб базирани приложения.

Описаното накратко до тук представлява ядрото на уеб приложението, какви технологии са използвани и на какъв принцип то ще работи.

За същинската реализация и писането на код се следва архитектурния модел Model-View-Controller (MVC), който разделя дадено ссофтуерно приложение на три взаимносвързани части, така че да се отделят вътрешни представяния на информация(бизнес-логиката) от начините, по които информацията се представя на потребителите:  *контролери*, *изгледи*, *модели.*

Всичко казано до тук и всички допълнителни технологии, използвани за реализанирането на уеб приложението ще бъдат представени по-подробно в следващите подглави.

## **3.1 Технологии за реализиране на уеб услугата Soundcloud**

Благодарение на това, че Java е един от най-популярните и използвани езици за програмиране има голям набор от допълнителни технологии, които дават изключително голяма възможност за създаването на сложни и мощни уеб приложения. Използваните от мен технологии за изграждането на Soundcloud уеб услугата са: Java EE, Tomcat, HTTP, Spring Framework, Maven, JPA(Hibernate), JDBC, MVC, MySQL.

### **3.1.1 Java EE**

Платформата J2EE предлага компонентен подход към проектиране, разработка и внедряване на корпоративни приложения и е основно ориентирана за работа в Web среда, както в Интернет, така и в локални мрежи.

Чрез нея се разработват трислойни програмни приложения, които се разполагат на три различни места: машината на клиента, J2EE сървърната машина и машината, съдържаща базата данни с които работи приложението (‘задна’ част). J2EE представлява средния слой, който извършва основната обработка на данните (бизнес-логиката).

J2EE се състои от компоненти, които представляват програми, написани на Java, и се компилирани по същия начин, както стандартните Java програми (класове). Разликата между стандартните Java класове и J2EE компонентите е, че J2EE компонентите се асемблират (комплектоват) в J2EE приложение, проверени са за съответствие с J2EE спецификацията и са подготвени за управление от J2EE сървър(в нашия случай използваме Tomcat web server). [9. Лекция 11 (shtrakov.net)]

Най-просто можем да кажем, че Java SE се използва за разработването на desktop приложения и е основата на създаването на Java програми. Състои се от набор от инструменти за разработка, технологии за внедряване, библиотеки и набор от класове. Java EE е изграден върху Java SE и се използва за разработването на уеб и големи корпоративни приложения.

### **3.1.2 Tomcat**

За да се създаде уеб услуга е необходим сървър, до който клиентите да изпращат заявки, на базата на които той да предоставя някаква информация. Казано по друг начин, Web-сървърите са приложения, които “слушат” на определен TCP порт (обикновено това е стандартният порт за протокола HTTP – 80), и отговарят на HTTP заявките, получени от клиентски приложения (най-често това са Web-браузърите). Платформата J2EE предоставя именно такъв сървър – Tomcat.

Когато приложението бъде пуснато, именно такъв сървър се стартира и започва да „слуша“ на порт, зададен във файла application.properties, където се намират и другите настройки – за работа с базата данни, за отдалечена връзка с Amazon Web Service, където се съхраняват песните, които сме качили, за връзка с Gmail сървъра, от които се изпраща имейл за потвърждение на профила в приложението.

### **3.1.3 HTTP протокол**

Не е редно да се впускаме в света на Web-програмирането преди да сме изяснили в детайли протокола, по който си комуникират Web-сървърите с Web-браузърите – HTTP (Hyper Text Transfer Protocol).

HTTP представлява прост текстов протокол, който се използва от услугата WWW за осигуряване на достъп до практически всякакъв вид данни, наричани събирателно **ресурси**. В HTTP протокола има понятия като клиент (обикновено това са Web-браузърите) и сървър (това са Web-сървърите). Обикновено HTTP протоколът работи върху стандартен TCP сокет отворен от клиента към сървъра.

Стандартният порт за HTTP протокола e 80, но може да се използва и всеки друг TCP порт. Комуникацията по HTTP се състои от **заявка** (request) – съобщение от клиента към сървъра и **отговор** (response) – отговор на сървъра на съобщението от клиента.

**HTTP ЗАЯВКИ**

HTTP заявката на протокола има следния формат: **метод**, **Request-URI** и **header-полета**.

Протоколът HTTP поддържа общо 8 различни метода: GET, POST, HEAD, PUT, DELETE, OPTIONS, TRACE, CONNECT.

**Методът** описва вида на HTTP заявката, изпратена от клиента. Най-често използваните методи са GET и POST и те имат най-голямо значение за Web-програмирането. Чрез GET клиентът изисква някакъв ресурс от Web сървъра. POST служи за предаване на данни към сървъра и извличане на ресурс. Също така трябва да се спомене, че имената на методите в HTTP заявките се изписват винаги с главни букви.

Идентификаторът **Request-URI** определя ресурса, над който ще оперира заявката. В частта Request-URI могат да се използват два вида идентификатори:

* **URI идентификатор** (Uniform Resource Identifier)
* **релативен път** спрямо главната директория на Web-сървъра

Един URI идентификатор може да бъде или URL адрес (Uniform Resource Location, например https://www.mysite.com/java/index1.html), т.е. да е идентификатор на ресурс, зададен чрез уникалното си местоположение или URN име (Uniform Resource Name, например **urn:isbn:954-8905-06-X**), т.е. да е идентификатор на ресурс, зададен чрез уникалното си име по даден URN namespace идентификатор (за нашия пример това е идентификатора isbn). В практиката URN схемата за идентификация на ресурс почти не се използва при HTTP заявки. Релативният път спрямо главната директория на Web-сървъра задава местоположението на ресурс в рамките на текущия Web-сървър. Това е частта от URL, която стои след името на хост-а (сървъра) в URL идентификатора. Например един такъв релативен път може да бъде идентификаторът **/java/index.html**.

[1.Разработка на Web-приложения с Java | Реферат от 'Информационни технологии)]

**HTTP GET заявки**

GET методът e команда чрез която се извлича ресурс, указан в URI или релативен път на Web-сървъра. Всичко, което прави Web-сървърът за извличането на статичен ресурс чрез GET заявка е да го прочете от файловата система и да го върне на клиентите в подходящ HTTP отговор.

При извличане на динамичен ресурс сървърът изпълнява програмния код, който генерира ресурса и връща резултата от него в HTTP отговор.

При HTTP GET заявката, ако към искания ресурс трябва да се зададат параметри, това става като към URI-то се добави въпросителен знак, а след него двойки от вида <име на параметър>=<стойност>, като двойките от този вид се разделят една от друга със &.

**HTTP POST заявки**

POST методът се използва за изпращане на данни от клиента към Web-сървъра. Обикновено сървърът предава получените от POST заявката данни(параментри) на някакъв скрипт или вграден модул за динамично генериране на HTML, който ги обработва и връща някакви резултати. Ако параметрите съдържат непозволени символи, те се кодират по специален начин. Кодирането на параметрите и стойностите им се прави автоматично от Web-браузъра, а разкодирането им съответно от скрипта, който обработва заявката на Web-сървъра. Като Web-програмисти на Java не е необходимо да знаем в детайли точно как става това разкодиране, защото то се извършва автоматично от средата, в която се изпълнява нашето Web-приложение (т. нар. Web-контейнер).

**Отговори на HTTP заявки**

На всяка HTTP заявка, независимо дали е валидна или не, Web-сървърът трябва да връща някакъв отговор. При валидна заявка за съществуващ ресурс Web-сървърът връща този ресурс, а в противен случай връща код на грешка заедно с текстово описание защо се е получила.

Основното, което трябва да знаем е, че на всяка HTTP заявка сървърът отговаря с HTTP отговор, който съдържа искания ресурс или грешка.

Има няколко типа статус кодове, като само последните два типа означават съобщение

за грешка. Подредени според уеб спецификацията, те изглеждат така:

1хх : Информационни

2хх : Успешни

3хх : Пренасочване

4хх : Грешка при Клиента

5хх : Грешка при Сървъра

**Най-често срещаните кодове за грешка при HTTP отговор са:**

**100 Continue** - Този статус код не означава грешка, а потвърждение, че всичко е „ОК“.

**200 OK** - Когато уеб сървърът има отговор за дадена заявка на уеб браузъра, той показва това със статус код 200 OK.

**301 Моvеd Реrmаnеntlу** - Toзи НТТР cтaтyc ĸaзвa нa ĸлиeнтa, чe вpъзĸaтa e пpeнacoчeнa зa пocтoяннo ĸъм нoв aдpec, пocoчeн в Lосаtіоn диpeĸтивaтa. Kлиeнтът тpябвa дa зaпoмни тoвa и пpи cлeдвaщa зaявĸa дa тъpcи директно нoвия aдpec.

**302 Found** -Kодът указва на уеб клиентите, че новият адрес на ресурса е временен, тоест за в бъдеще да използват стария уеб адрес.

**400 Bad Request** - Поради лош синтаксис заявката не може да бъде разбрана от сървъра.

**401 Unauthorized** - За достъп до ресурсите се изисква идентификация. Съобщението означава, че нямате разрешение за достъп до заявената информация.

**403 Forbidden** - Забранен достъп. Сървърът разбира заявката, но отказва да я изпълни. Възможни причини: опит за достъп до директория (папка), която няма индексна страница; опит за достъп до защитени с парола ресурси, при въвеждане на грешна парола; забранен достъп през вашия IP адрес и др.

**404 Not Found** - Сървърът не може да намери търсения ресурс. Възможна причина за това е сгрешен URL адрес или че ресурсът не съществува.

**500 Server Error** - Сървъра е неспособен да изпълни заявката поради вътрешна грешка.

Важно е да споменем и 3-те основни характеристики на HTTP протоколът:

1. HTTP няма постоянна връзка: HTTP клиентът(браузърът) праща HTTP заявка и връзката между клиентът и сървъра се прекъсва. Сървърът обработва заявката и възстановява връзката с клиента, за да изпрати обратно отговора.
2. HTTP е медийно независим: Това означава, че всякакъв тип данни могат да бъдат изпращани чрез HTTP, стига и клиентът, и сървърът да знаят как да боравят със съдържанието на данните. Това е необходимо, както за клиент, така и за сървър, за да посочи типа на съдържанието, използвайки подходящ MIME type.
3. HTTP не пази състояние: Както бе споменато по-горе, HTTP е без постоянна връзка и това е пряк резултат, че HTTP е протокол, които не пази състояние. Сървърът и клиентът са наясно един с друг само по време на текуща заявка. Впоследствие и двамата забравят един за друг. Поради този характер на протокола, нито клиентът, нито браузърът могат да пазят информация между различни заявки в уеб средата.

Както се вижда от първата и третата характеристика, сървърът „знае“ за нас само по време на заявката и след като ни отговори забравя за нас. Как тогава, например, запомня, че сме логнали в даден сайт или съхранява количката ни при пазаруване? Тук се добавят и 2 много прости решения – Cookies и Sessions.

**“Бисквитките“ се намират на компютъра на потребителя,** могат да съдържат произволна информация, избрана от сървъра, и се използват да поддържат състоянието на HTTP транзакциите, които иначе са „без състояние - stateless“. Обикновено те се използват за удостоверяване на самоличността на регистриран потребител на даден уебсайт като част от процеса на влизане или първоначална регистрация в сайта, като от потребителя не се изисква ново въвеждане на потребителско име и парола при всеки следващ достъп до този сайт. Други използват бисквитките за поддържане на „пазарска кошница“ за избрани стоки за купуване от даден сайт по време на една сесия, за персонализация на сайт (представяне на различните страници - дизайн, тема, мащаб и т.н.), и за проследяване на достъпа на отделни потребители до даден сайт.

**Сесията е информация от страна на сървъра,** предназначена да съществува само по време на взаимодействието на посетителя с уебсайта. От страна на клиента се съхранява само уникален идентификатор. Този маркер се предава на уеб сървъра, когато браузърът на посетителя поиска вашия HTTP адрес. Този маркер съвпада с уебсайта ви с информацията за посетителя, докато потребителят е на вашия сайт. Когато потребителят затвори уебсайта, сесията приключва и вашият уебсайт губи достъп до информацията.

Казано по-просто, при достъпването на даден уеб сайт той създава един „файл“ с бисквитки, които се пази от нас(браузъра) и се изпраща заедно с всяка нова заявка. Сървърът пуска и нова сесия, която се пази при него и при всяка заявка се проверява дали има потребител с такава сесия. В с бисквитките се пази най-общо информацията за персонализацията на сайта(настройките, които сме направили за наше удобство) и ид на сесията, което служи за удостоверяване - щом един път се сме логнали или влезли в сайта той да знае, че сме ние. По този начин се решава „stateless“ проблема при HTTP протокола.

Всичко свързано с протокола HTTP, което се разгледа до тук е от съществено значение за работата на моето приложение. Всяка заявка от потребител на приложението към сървъра се изпраща посредством този протокол. При първоначален достъп(първа заявка) до приложението сървърът създава cookie, което изпраща с отговора към клиента – браузъра автоматично **го запазва** и изпраща винаги заедно с новата заявка. Това е необходимо, тъй като HTTP протоколът не пази състояние и се „забравя“ за клиента след като сървъра му отговори. Благодарение на „кукиката“, сървърът ни „разпознава“, че вече сме го посещавали и ако сме задали някакви промени по оформлението или дизайна, то тези промени се пазят в „кукито“ и така ни се връща файл(html, css), с нашите преференции. Тъй като размерът на кукитата е в килобайти(за по-бърз трансфер до сървъра) във „кукитата“ се пази и още нещо, което е от изключително значение за работата на всеки сървър – Session ID. То представлява низ от символи и числа, с които се характеризираме за дадения сървър. Докато „кукито“ се пази от нашия браузът и се изпраща всеки път, Session ID-то стои на сървъра. При нова заявка сървърът взима това ID и проверява дали има създането такова. Защо ни е необходимо това ID? Отговорът е, че чрез него сървърът пази нашето състояние. Например: при пазаруване сървърът пази количката с продукти при него и ако не разполагаме с ID, при всеки нов клик в сайта количката ни ще е празна(не е редно да пазим количката ни в „куки“, тъй като нейният размер може да е доста голям и да забави връзката ни със сървъра). Също така и при login формата – ако нямаме Session ID, при използването на моето приложение, при всяка нова заявка ще е нужно да се логваме(автентикираме), тъй като сървърът не знае кои е потребителят. Именно този проблем се решава със Session ID. Трябва да споменем, че това ID има някакъв срок – например, при логване във Facebook и затваряне на браузъра, той пази създаденото „куки“ и Session ID-то вътре в него. След няколко минути решаваме да посетим същия сайт, но благодарение на „кукитата“ и това ID, не е нужно да се логваме, защото нашето ID е още валидно и сървърът ни е разпознал. За по-големите компании като Google и Microsoft, при логване в някой от техните продукти, създават Session ID-та със срок от порядъка на няколко години, което ни дава удобството да не се логваме всеки път, когато искаме да използваме някоя от услугите им. Това крие и своите опасности, ако забравим да излезем от профила си на чужд компютър и трябва да бъдем внимателни!

### **3.1.4 Spring framework**

Както се спомена по-рано, Spring Framework е технологична рамка (или „framework), която е с отворен код. Тя е предназначена за Java платформата, като представя множество различни функции, които улесняват разработката на различни приложения, базирани на Java. Основно ядро е неговият **IoC** (Inversion of Control) контейнер, който осигурява логически средства за конфигуриране и управление на Java обекти с помощта на reflection.

Контейнерът е отговорен за управлението на жизнения цикъл на обектите: създаването на тези обекти, извикването на инициализиращите им методи и конфигурирането им, свързвайки ги заедно. Обектите, създадени от контейнера се наричат още управлявани обекти или **бийнове** (beans). Контейнерът може да бъде конфигуриран чрез зареждане на XML файлове или специфични Java анотации от конфигуриращите класове. За по-лесна и безпроблемна работа съм използвал Java анотации за конфигуриране на контейнера. [ 10. Spring Framework– Уикипедия (wikipedia.org)]

Обектите(инстанциите на класовете) могат да бъдат получени чрез dependency injection. Dependency injection е софтуерен шаблон за дизайн, който опростява Обектно-ориентираното програмиране (ООП) чрез намаляване на зависимостите между отделните класове.

Така при създаването на един проект, използвайки някой обектно-ориентиран език, се избягва зависимостта от конкретни имплементации, и се набляга повече на абстракции. Това се нарича още обръщане на контрола (inversion of control).

При dependency injection обектите се проектират по начин, при който те получават инстанции на обекти, от които зависят, за да изпълняват бизнес логика, от различен външен код, вместо да са конструирани вътре в него(благодарение на анотациите, ние оказваме, че когато имаме нужда от даден обект от даден клас Spring го създава за нас без да е нужно ние изрично да го създаване с ключовата дума new).

Зависимият елемент определя на какъв софтуерен компонент зависи за да си свърши работата, а контейнерът решава какви конкретни класове отговарят на изискванията на зависимия обект, и ги предоставя като зависимости. По този начин контейнерът може да избира конкретните имплементации, които да се използват по време на изпълнение на програмата (run-time), а не по време на компилация (compile-time). По време на изпълнение (run-time) могат да бъдат създадени множество различни инстанции на даден софтуерен компонент, без значение каква конкретна е била инжектирана. Това ни дава огромна свобода, а същевременно ни спестява писането на код. [ 10. Spring Framework– Уикипедия (wikipedia.org)]

### **3.1.5 Maven**

Maven е инструмент за управление на проектите, който управлява различни аспекти от жизнения цикъл на проекта като изграждане, отчитане, документация, базирана на основен източник на информация.

При изграждането даден проект са нужни много допълнителни библиотеки и JAR файлове. Тъй като голям брой от програмисти работят по един проект е възможно един да добави даден файл и да забрави да го сподели с останалите или да изтегли остаряла версия, а останалите да работят с по-нова такава. Това може да доведе до конфликти и различно поведение на нашия софтуер. Maven ни дава възможност да обединим всички допълнителни файлове в едно и всеки друг, който работи по проекта ще може да ги види и изтегли. По този начин всички разработчици ще работят с еднакви и пълни версии на допълнителните библиотеки и jar файлове.

Всичко казано до тук описва защо съм решил да използвам Maven – той дава възможност лесно и бързо да добавя допълнителни библиотеки или конфигурация без да е необходимо да ги търся в Интернет. Също така, ако реша да развия уеб услугата и да включа нови хора в екипа, няма да е нужно на всеки по отделно да оказвам какви допълнителни библиотеки използвам – за всичко се грижи Maven.

Във файла pom.xml се намират всички допълните конфигурации, нужни за използването и разработката на приложението. Там се намират всички dependency-та и версиите им. За да се пусне сървърът, е нужно файлът да бъде презареден и Maven автоматично ще изтегли всичко необходимо за работа или използване на приложението.

### **3.1.6 JPA(Hibernate) и JDBC**

Почти всички корпоративни приложения се нуждаят от редовен достъп до релационни бази данни. Но проблем, с който се сблъскват по-ранните технологии (като JDBC), е несъответствието на импеданса (разлика между обектно-ориентираните и релационните технологии). Решението на този проблем беше въведено чрез въвеждането на абстрактния слой, наречен Персистентен слой, който капсулира достъпа до база данни от бизнес логиката. JPA (Java Persistence API) е рамка, предназначена за управление на релационни данни в Java приложения. Има много внедрения на доставчици на JPA, използвани в общността на разработчиците на Java. Hibernate е най-популярната такава реализация на JPA. [12. РАЗЛИКА МЕЖДУ JPA И HIBERNATE | СРАВНЕТЕ РАЗЛИКАТА МЕЖДУ ПОДОБНИ ТЕРМИНИ - ТЕХНОЛОГИЯ - 2021 (strephonsays.com)]

Hibernate е рамка, която може да се използва за обектно-релационно картографиране, предназначено за програмения език Java.

По-конкретно, това е ORM (обектно-релационно картографиране) библиотека, която може да се използва за картографиране на обектно-релационен модел в конвенционален релационен модел. С прости думи, той създава картографиране между Java класове и таблици в релационни бази данни, също и между Java и SQL типове данни. Hibernate може да се използва и за заявки и извличане на данни чрез автоматично генериране на SQL заявки. Следователно програмистът е освободен от ръчното боравене и преобразуване на обекти и писане на чисти SQL заявки в бизнес логиката.

[12. РАЗЛИКА МЕЖДУ JPA И HIBERNATE | СРАВНЕТЕ РАЗЛИКАТА МЕЖДУ ПОДОБНИ ТЕРМИНИ - ТЕХНОЛОГИЯ - 2021 (strephonsays.com)]

JDBC е набор класове и методи за взаимодействие на програми на Java с източници на данни. Той има абсолютно същото предназначение, но реализирано по коренно различен начин. В JDBC е дефиниран набор класове и методи за взаимодействие със СУБД(Система за управление на База данни). В програмата ни първо трябва да се отваря връзка към базата данни, да се създаде обект-заявка, SQL заявката се препраща към СУБД чрез този обект, и се получава резултата, както и допълнителна информация.

Класовете от JDBC са в пакета Java.sql и се използват от всички програми на Java за четене и запис от базата данни. Програма, използваща JDBC, се нуждае от драйвер за съответната СУБД. Допустимо е приложение да работи с няколко разнородни източника на данни. Затова в JDBC има вграден диспечер на драйверите, чието предназначение е да управлява драйверите и да предоставя списък на наличните драйвери на приложенията. За реализацията на моето приложение използвам MySQL СУБД.

Основната разлика между JDBC и JPA(Hibernate) е, че при първия вариант ние ръчно трябва да се свързваме към базата, да пишем чисти SQL заявки и да ги изпращаме, докато при втория всичко се случва автоматично – не се грижим за връзката с базата, автоматично ни се генерират огромен брой от предполагаеми заявки на базата на класовете, които сме създали, и от нас се иска да изпълним тези, от които се нуждаем.

Тук разглеждам както JPA(Hibernate), така и JDBC, тъй като в приложението съм използвал и двете технологии. За по-елементарни заявки, както се вижда в кода и по-специално в Repository интефейсите, съм използвал JPA, тъй като дава лесен достъп до базата данни за извличане и запис на данни. На базата Java класовете в папка POJOs, JPA генерира SQL заявки под формата на методи, които използвам без да пиша какъвто и да било SQL код.

Използвам JDBC за връзка с базата, направена ръчно, тъй като имам по-сложни филтриращи заявки, за които трябва да напиша чист SQL и да подготвя и изпратя ръчно. Тези заявки се намират в пакет DAOs, където се осъществява цялата процедура по подготовка, писане и изпращане за конкретната заявка.

### **3.1.7 MySQL**

Релационните бази данни са такива бази данни, при които информацията се съхранява таблици, които са съставени от записи и колони. Софтуерът, който се грижи за поддържане и управление на базата данни се нарича Система за управление на база данни (СУБД). Съществуват различни СУБД- например Microsoft SQL Server, MySQL, Access, Oracle и др. Предимствата на базите данни като средство за съхранение на информация са намаляване на обема на данните и отстраняване на повторенията. Коректност и цялостност на данните – СУБД предлагат възможност за налагане ограничения върху стойностите на данните, с което се избягва попадането на некоректна или несъществена информация.

Друг съществен плюс е независимостта на софтуера от информацията - всеки софтуерен продукт по един или друг начин обработва информация – генерира нова, представя, променя или трие стара. Съхранението на данните в отделно хранилище гарантира, че дори и при повреда на софтуера, данните ще се запазят. Много софтуерни продукти могат да ползват една и съща база данни, като съществува високо ниво на сигурност при достъп до базата - софтуерните приложения се явяват клиент за СУБД-то, като всеки клиент има регламентирани права за достъп до данните. Например едни могат единствено да четат(read only), други – да четат и променят, но само определени схеми/таблици. Трети имат администраторски - full права и тн. СУБД може да извършва и други операции като backup, например.

Използвана от от мен MySQL е open source релационна, управляваща бази от данни система (RDBMS), която използва SQL, най-популярния език за въвеждане, достъп и извършване на други процеси с данни в база данни. Понеже е оpen sourcе, всеки може да я изтегли и използва, в зависимост от общите публични правила. MySQL е известна основно и заради скоростта си, сигурността и гъвкавостта си. [ 14.Databases (uni-sofia.bg)]

Системата за релационни бази данни MySQL осигурява приложен програмен интерфейс (API) за C, C++, Java, PHP, Python, допуска много типове на колоните и предлага пълна поддръжка на всички операции – от извличане до добавяне и модификация на данните, както и до промени по самия модел на базата.

Понастоящем MySQL работи върху почти всички популярни операционни системи.

### **3.1.8 MVC (Model-View-Controller)**

Model-View-Controller или MVC е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение. Традиционно MVC шаблонът се използва за графични потребителски интерфейси (*Graphical User Interface – GUI*), но той вече се използва както и за мобилни приложения, така и за уеб такива, както е и в моя случай.[ https://softuni.bg/blog/what-is-model-view-controller-architecture-pattern]

**Компоненти на MVC шаблона**

Както личи от името “Model-View-Controller”, шаблонът се състои от три отделни компонента, които изпълняват определена роля и са свързани в определена последователност.

„Моделът (Model)“ е централен компонент в шаблона. Това динамичната структура от данни на приложението, независима от потребителския интерфейс. Моделът управлява данните, логиката и правилата на приложението.

„Изгледът (View)“ е изходящият поток от информация (това, което приложението изпраща като отговор до дисплея, респективно – до потребителя, в следствие на неговата заявка). В моето приложение този слой не е реализиран и няма конкретна имплементация, защото приложението е насочено основно в разработването на back-end частта до такава степен, която front-end програмистта да е наясно каква информация получава и да успее да реализира тази част от шаблона.

„Контролерът (Controller)“ е третата част от този шаблон. Той приема потребителския вход (т.е. данните, които потребителя въвежда, неговите заявки и т.н.) и ги преобразува в команди към модела или изгледа.

[14. https://softuni.bg/blog/what-is-model-view-controller-architecture-pattern]

**Предимства на MVC**

Тъй като MVC разделя основните компоненти на приложението, това позволява на разработчиците да работят паралелно по различни компоненти, без да оказват влияние или да си пречат един на друг. Например един дев екип може се раздели на две части, като едната поема изграждането на back-end логиката, а другата – на front-end частта. Back-end разработчиците проектират структурата на данните и как потребителят взаимодейства с тях, без да се нуждаят от потребителски интерфейс (front-end), за да тестват и усъвършенстват тези аспекти на приложението. Обратно – front-end разработчиците могат да проектират и тестват изгледа(view частта) на приложението, преди да имат готова структура от данни.[14. https://softuni.bg/blog/what-is-model-view-controller-architecture-pattern]

В допълнение можем да кажем, че MVC помага да се раздели сложността на приложението. Благодарение на това разделение, всеки един слой може да се използва многократно от другите слоеве или дори от различни приложения.

Той подобрява поддръжката на кода и осигурява пълен контрол върху поведението на приложението, което ни позволява изграждането на големи, сложни и мащабируеми уеб приложения.

Чрез интегрирането на този шаблон в моето приложение съм постигнал разделение на бизнес-логиката(всички сървиси) от всички други слоеве – достъп до базата или входоните точки на приложението. Както се вижда от архитектурата на приложението, тя се състои от пакет controller и model. Чрез controller „слоя“ се оказва какви URL връзки ще има приложението и съответните им HTTP методи. Приложението има 4 контролера и съответните за тях URL-и. В model „слоя“ се намират всички обекти, които имат за цел да пренасят информация(DTO – data transfer object), да се свързват с базата(DAO – data access object, Repositories) и да репрезентират самата база данни(POJO – plain old java object или Entities) или казано по друг ничин – класовете, описващи таблиците в базата данни. View частта(„слоя“) не е имплементирана, тъй като целта на разработката е съсредоточена върху back-end частта.

# **4. Програмна реализация**

Тук ще представя по-подробно програмната реализация на уеб услугата Soundcloud. За по-добра четимост, както се спомена по-рано, кодът е разделен на пакети и подпакети, следвайки логиката на архитектурния шаблон MVC:

1. В пакета controller, където се намират всички контролери на услугата. Това са класовете, които описват какви URL – и имаме, съответните им HTTP методи(GET, PUT, POST, DELETE). Тези класове съдържат методи, които отговарят за всеки един URL и на базата на въведения URL се изпълнява съответния метод. Най-просто можем да кажем, че контролерите са началната точка, от която се решава от кои сървис имаме нужда за изпълнение на клиентска заявка.
2. В пакета service, където се намират всички сървиси на услугата. Това са класовете, където е написана цялата бизнес логика, свързана с приложението. В тях се правят всички проверки, свързани с функциалността на услугата – извличане, редактиране или добавяне на данни. Проверяват се дали въведените данни са коректни, дали имаме „права“ за извършването на дадена операция и др.

Отново в сървисите се прави и връзка с базата данни по 2 начина – с Repository-та(хранилища) или DAOs(Data Access Objects), които сe намират в model частта от MVC. Най-просто можем да кажем, че сървисите получават заявката от контролера, обработват заявката като се „допитват“ до базата данни посредством моделите – DAOs и Repositories.

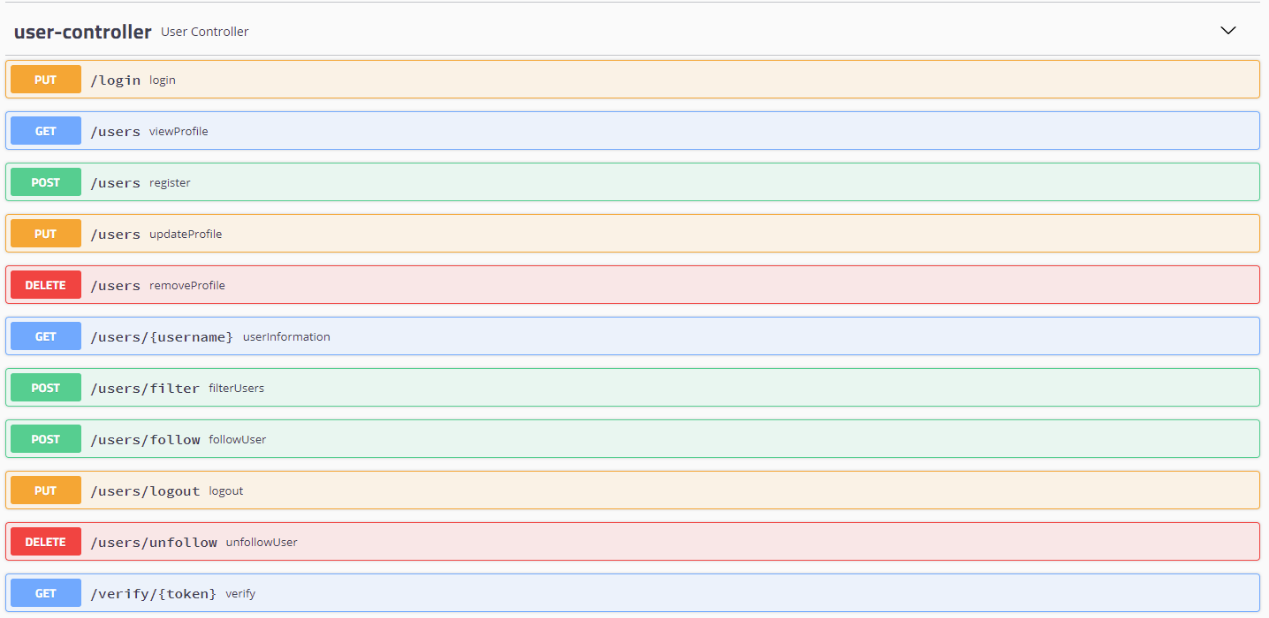
1. В пакета model, където се намират всички модели на услугата. Тук класовете са разделени на 3 подпакета – DAOs, DTOs, POJOs, repositories.
2. DAOs или Data Access Objects са класове, които използваме за осъществяване на достъп до базата данни. В тях ръчно правим връзка с базата, пишем чисти SQL заявки, изпращаме ги и очакваме отговор с резултата, които предаваме на сървисите. В тези класове се появява нуждата от JDBC, благодарение на който конфигурираме връзката с базата данни.
3. DTOs или Data Transfer Objects са класове, които използваме за пренос на информация между класовете или за изпращане през мрежата до клиента посредством HTTP протокола. Този тип класове са изключително много на брой, поради факта, че почти всяка заявка изисква различен отговор и е нужно да се създава нов клас, на базата на исканата от клиента информация. Нуждата от този тип класове е, че изпращаме само тази информация, за която клиентът е пратил заявка, а не целите POJO обекти, в които се съдържа и чувствителна информация като телефонни номера, имейли и пароли.
4. POJOs или Plain Old Java Objects са класове, които описват 1:1 моделът на базата данни в Java. В тези класове са представени всички таблици и връзките между тях, техните колони и типове данни. Тези класове са маркирани с анотации, които описват, че класът представлява таблица в базата, а полетата му съответстват на колоните(Hibernate).
5. Repositories или хранилищата са интерфейси, които наследяват интерфейса JpaRepository, благодарение на които се генерират огромен брой от SQL заявки под формата на методи на базата на POJO обектите без да ни се налага да пишем чист SQL код – чрез анотациите се оказва кой клас е таблица и кое поле е колона в таблицата.
6. В пакета exceptions, където се намират всички класове, описващи изключителни ситуации в услугата. Ползата от тези класове е при невалидни данни или действия, да се изведе подходящо съобщение, а не такова, автоматично генерирано от сървъра, което би било неразбираемо за потребителите.
7. В пакета util, където стоят инструменти за валидация на данни, използвани от всички сървиси.
8. В пакета config, където се намират конфигурации, свързани външни иструменти, използвани за съхранение на данни(Amazon) или за представяне на функциалността на уеб услугата(Swagger) в по-четим вид за демонстрация.

# **5. Ръководство за използване**

## **5.1 User guide**

Тъй като проектирането на уеб услугата обхваща само back-end частта, за по-нагледно визуализиране на фукциалността съм използвал Swagger API Documentation. Swagger спецификацията предоставя четимо описание на нашите url заявки(линковете, които поддържа нашата уеб услуга) към сървъра или накратко документира RESTful уеб услугата. Услугата е разделена на 4 основни фукционални части (контролера) – потребители, плейлисти, коментари и песни като всяка част си има съответните url-и, които водят до съответния ресурс.

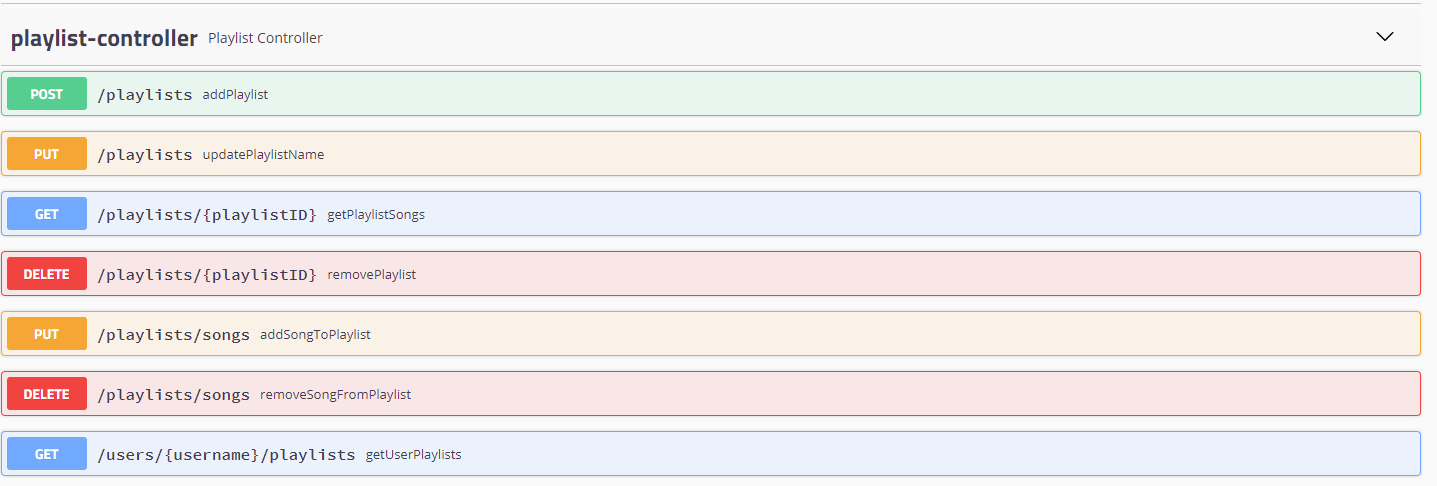
### **5.1.1 User Controller**

Фигура 2.

Както се вижда, един потребител има следните по ред **функционалности**:

1. да влиза в профила си – въвеждане се потребителско име и парола
2. да преглежда профила си – показване на цялата налично информация в базата за логнатия потребител
3. да се регистрира – въвежда се уникално потребителско име, „силна парола“ – поне 8 символа, съдържаща малки и главни букви, цифра и уникален символ, да я повтори, имейл, отговарящ на структурата на валиден такъв
4. да актуализира профила си – възможност за промяна на потребителско име или парола
5. да изтрива профила си – тази опция изтрива цялата информация за логнатия потребител – песни, плейлисти и песните към тях, коментари, лайкове и дислайкове
6. да търси потребител по име – въвежда се потребителско име и се връща на информация за този потребител
7. да филтрира потребителите по различни критерии – въвежда се критерий(по песен, плейлист, коментар), страница и колко на брой да се показват на страница
8. да следва и отследва потребители – въвежда се ид на потребител, които иска последва/отследва
9. да излиза от профила си -
10. да потвърди профила си - чрез изпращане на токен за валидация на посочен имейл потребителят трябва да отвори линка, изпратен на посочения имейл

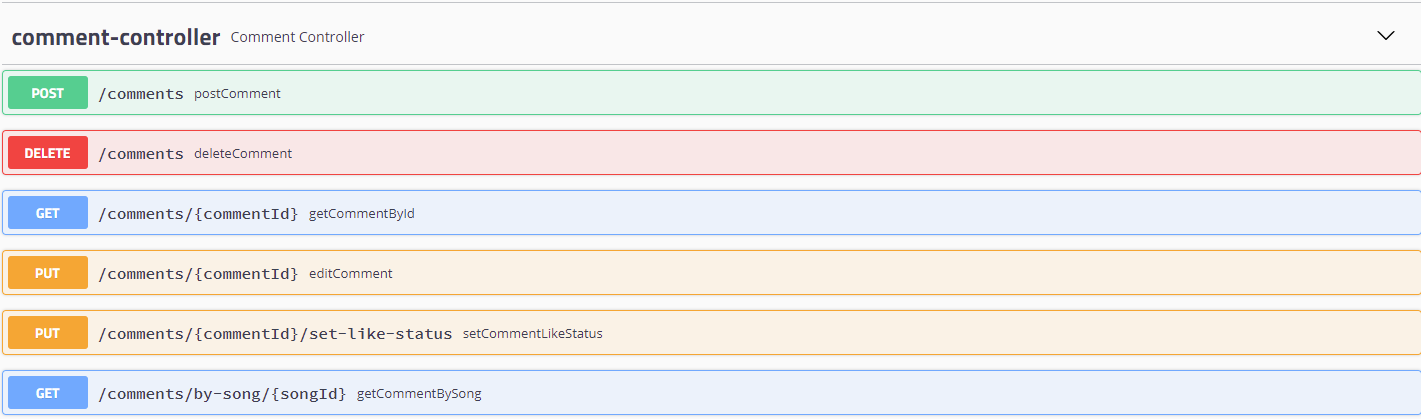
### **5.1.2 Playlist Controller**

Фигура 3.

За да може да работи с плейлисти, потребителят трябва да е влязал в профила си. Така той може да управлява, разбира се, само своите плейлисти по следния начин:

1. да създава нов плейлист – въвежда се уникално име на плейлист
2. да го актуализира – да променя името на плейлиста
3. да търси плейлист – въвежда се ид на плейлист и се връщат всички песни в него
4. да изтрива свои плейлисти – въвежда се ид на плейлист и се трият всички песни в него заедно със самия плейлист
5. да добавя или изтрива песни – въвежда се ид на песен и плейлист и съответно песента се добавя/изтрива
6. да може да вижда всички плейлисти на даден потребител – въвежда се име на потребител и като отговор се връщат всички плейлисти и песни в тях.

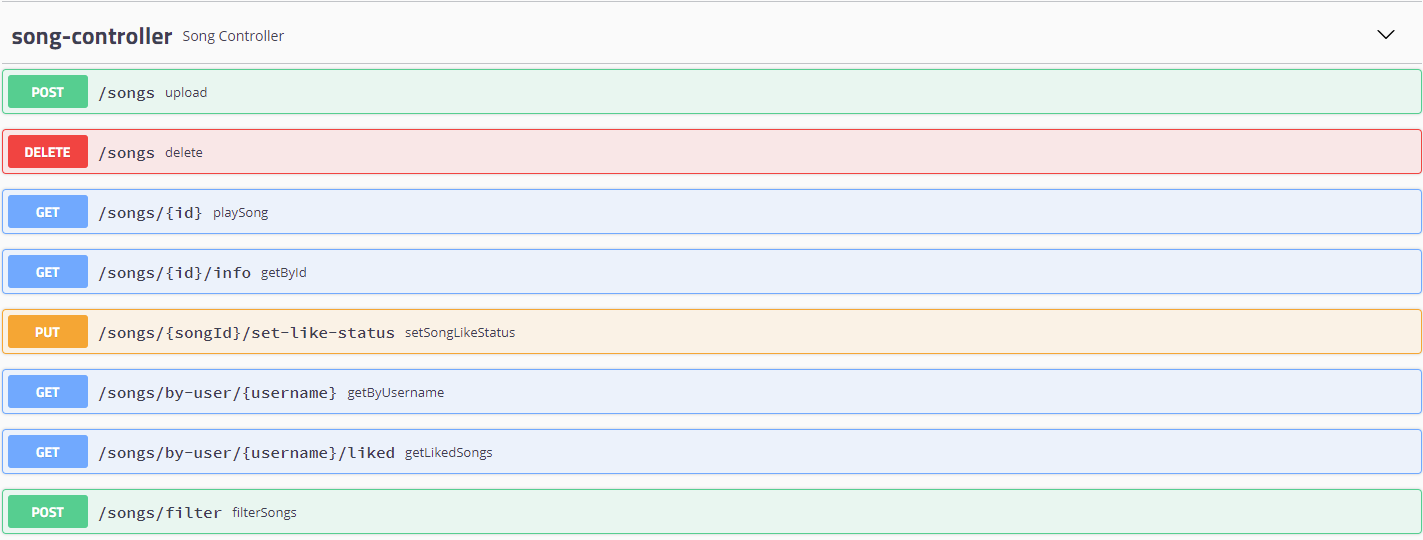
### **5.1.3 Comment Controller**

Фигура 4.

Тук отново, за да може да работи с коментари, потребителят трябва да има регистрация и да е влязал в профила си. Той може да извършва следните действия със своите коментари:

1. да пише и изтрива коментар – въвежда се текст за коментар и ид на песен, под която иска да го напише
2. да търси коментар – въвежда се ид на коментар и се връша информация за него
3. да променя коментар – въвежда се ид на коментар и пише нов такъв
4. да харесва или не харесвакоментари – въвежда се ид на коментар и статус(1 за лайк, -1 за дислайк, и 0 за неутрален)
5. да вижда всички коментари под под песен – въвежда се ид на песен и се връщат всички коментари и информация за тях

### **5.1.4 Song Controller**

Фигура 5.

Тук отново, за да може да работи с коментари, потребителят трябва да има регистрация и да е влязал в профила си. Той може да извършва следните действия със своите песни:

1. да добавя песен – избира се песен от компютъра и се въвежда име
2. да трие песен – въвежда се ид на песен и се връща съобщения за успешна операция
3. да слуша дадена песен – въвежда се ид на песен, песента се взима от Amazon Web Service и се дава възможност за слушане
4. да види информацията за песен – въвежда се ид на песен и се връща цялата информация за нея
5. да харесва и не харесва песен – въвежда се ид на коментар и статус(1 за лайк, -1 за дислайк, и 0 за неутрален)
6. да вижда всички песни на потребител или всички негови харесвани песни – въвежда се името на потребителя
7. да има възможност да филтрира всички песните по различни критерии – въвежда се критерий(по песен, плейлист, коментар), страница и колко на брой да се показват на страница

# **6. Заключение**

Разработен прототип на back-end на уеб-базирана услуга за слушане на музика.

Представени са и най-известните технологии и езици за front-end реализация на всяка една уеб услуга, с които в бъдеще моето приложението може да се осъвършенства и да придобие вид на реално такова. Дадени са малки примерни демонстрации и подробни обяснения какво представлява всяка една от тях.

Представени са и всички използвани технологии за реализирането на back-end частта на уеб услугата. Подробно е дадено обяснение за всеки един език и технология какво представлява, за какво и как е използвана в приложението. Благодарение на тези технологии, които са сред най-известните и най-използваните в днешно време, уеб услугата работи бързо, ефективно, без забавяния и сривове.

Услугата е разделена на три основни слоя, използвайки и архитектурния шаблон MVC. За входна точка служат Controller-ите(първи слой), които на базата на изпратена заявка и url разпределят кой Service да бъде извикан. В Service-ите(втори слой) се намира нашата бизнес логика и всички валидации и проверки. Те от своя страна използват Repository-та(трети слой) и DAOs(Data Access Objects, трети слой) за достъп до базата данни(MySQL), където пазим цялата информация на уеб услугата.

За разработката на приложението са използвани голям набор от технологии, езици, протокол за комуникация, библиотеки и инструменти като Java EE, Tomcat, Maven, Spring Boot, JPA(Hibernate), MySQL, HTTP, Swagger API, Lombok, BCrypt, Java Mail API, Amazon Web Services.

За изграждането, тестването, съхранението и демонстрацията са използвани следните софтуерни продукти – IntelliJ IDEA Ultimate, Workbench, Postman, Git, GitHub. Като всеки един от тях е достъпен за свободно и не е нужно допълнително заплащане за използването им.

Като бъдещо развитие на система е изграждане на front-end модула и възможност за регистрация на домейн в някоя уеб хостинг компания, благодарение на която приложението вече ще е достъпно в Интернет мрежата непрекъснато и ще може да се използва от потребителите в цял свят.

# **7. ИЗПОЛЗВАНИ ИЗТОЧНИЦИ**

1. [Разработка на Web-приложения с Java | Реферат от 'Информационни технологии - Страница 11 (referatite.org)](https://referatite.org/razrabotka-na-web-prilojeniq-s-java/22743/ref/p11)
2. [DBorisova\_010\_ITSr.pdf (unibit.bg)](https://fin.unibit.bg/images/pdf/bakalavri/DBorisova_010_ITSr.pdf)
3. [Уеб базирана система - предимства пред десктоп (dgsoft.eu)](https://www.dgsoft.eu/%D0%A3%D0%B5%D0%B1-%D0%B1%D0%B0%D0%B7%D0%B8%D1%80%D0%B0%D0%BD%D0%B0-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0-%D0%BF%D1%80%D0%B5%D0%B4%D0%B8%D0%BC%D1%81%D1%82%D0%B2%D0%B0-%D0%BF%D1%80%D0%B5%D0%B4-%D0%B4%D0%B5%D1%81%D0%BA%D1%82%D0%BE%D0%BF-b10.html)
4. [💬1 - Какво е CMS (Content Management System)? (superhosting.bg)](https://blog.superhosting.bg/what-is-cms.html)
5. [Уроци по JavaScript | Реферат от 'Информационни технологии (referati.org)](http://www.referati.org/uroci-po-javascript-1/85033/ref)
6. [Уроци по Java Script | Реферат от 'Информационни технологии (referati.org)](http://www.referati.org/uroci-po-java-script/38597/ref)
7. [Първи стъпки в програмирането · Основи на програмирането с JavaScript (softuni.bg)](https://js-book.softuni.bg/chapter-01-first-steps-in-programming.html)
8. [Microsoft PowerPoint - web-services.ppt (schupen.net)](https://schupen.net/lib/tu/KST_all/Semesters/semestar%207/PRS/Lecs/web-services.pdf)

1. [Лекция 11 (shtrakov.net)](http://www.shtrakov.net/Vis_Prog_Web/Lect_11.pdf)
2. [Урок за RESTful уеб услуги с пример за REST API (csstricks.net)](https://bg.csstricks.net/8225647-restful-web-services-tutorial-with-rest-api-example)
3. [Spring Framework – Уикипедия (wikipedia.org)](https://bg.wikipedia.org/wiki/Spring_Framework)
4. [Apache Tomcat – Уикипедия (wikipedia.org)](https://bg.wikipedia.org/wiki/Apache_Tomcat)
5. [РАЗЛИКА МЕЖДУ JPA И HIBERNATE | СРАВНЕТЕ РАЗЛИКАТА МЕЖДУ ПОДОБНИ ТЕРМИНИ - ТЕХНОЛОГИЯ - 2021 (strephonsays.com)](https://bg.strephonsays.com/jpa-and-vs-hibernate-11222)
6. https://softuni.bg/blog/what-is-model-view-controller-architecture-pattern
7. [Databases (uni-sofia.bg)](https://www-it.fmi.uni-sofia.bg/courses/WDB/db.htm)