



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ
Факултет Компютърни системи и технологии
Катедра „Компютърни системи“

ДИПЛОМНА РАБОТА

за придобиване на образователно-квалификационна степен
„бакалавър“

на тема

**„Проектиране и разработка на уеб приложение за
града Неготино“**

Дипломант:

Димитар Аврамов

Фак. номер: 123216003

Група: 47

Специалност: КСИ

Научен ръководител:

доц. д-р Антония Ташева

София, 2020 г.

Задание

Декларация за авторство

Съдържание

Задание	2
Декларация за авторство.....	3
Съдържание.....	4
Увод	7
1. Анализ на темата. Преглед на подобни приложения. Цели и задачи.	8
1.0. Мотивация.	10
1.1. Анализ на тематичната област.....	12
1.2. Съществуващи решения	15
1.2.1. www.visitsofia.bg.....	15
1.2.2. www.varna.bg	18
1.2.3. www.burgas.bg	21
1.3. Цели и задачи	25
2. Обзор на използваните технологии	28
2.1. Java.....	28
2.2. Spring Boot.....	29
2.3. MySQL.....	31
2.4. HTML5	32
2.4. CSS.....	33
2.5. JavaScript	34
2.6. Thymeleaf.....	35
2.6. Apache Tomcat	35
2.7. Извод.....	35
3. Проектиране на приложението	37
3.1. Функционални и нефункционални изисквания.....	37
3.1.1. Функционални изисквания.....	37
3.1.2. Нефункционални изисквания.....	38
3.2 Архитектура	42
3.2.1 Цялостна архитектура	42
3.2.2 Архитектура според MVC шаблона.....	42
3.2.3 Как Spring имплементира MVC	45
3.2.4 Архитектура на уеб приложението.....	46
3.2.5 Архитектура на Spring Data.....	47
3.2.6 Архитектура и механизъм за автентикация и авторизация	48

3.3 Проектиране на базата данни.....	49
3.4 Работен процес на приложението.....	56
3.4.1 Начална страница - Без въвеждане на ендпойнт (/)	56
3.4.2 Новини - /news.....	56
3.4.3 Одобряване на изпратена новина - /news/approve	57
3.4.4 Изхвърляне на новина - /news/discard	57
3.4.5 Одобряване на всички изпратени новини - /news/approve-all	58
3.4.6 Отхвърляне на всички изпратени новини - /news/discard-all.....	58
3.4.7 Добавяне на новина - /news/add	58
3.4.8 Събития - /events	59
3.4.9 Одобряване на изпратено събитие - /events/approve	59
3.4.10 Изхвърляне на събитие - /events/discard	59
3.4.11 Одобряване на всички изпратени новини - /events/approve-all.....	59
3.4.12 Отхвърляне всички изпратени новини - /events/discard-all	59
3.4.13 Добри дела - /good-things.....	59
3.4.14 Одобряване на изпратено добро дело - /good-things/approve	60
3.4.15 Отхвърляне на добро дело - /good-things/discard.....	60
3.4.16 Одобряване на всички изпратени добри дело - /good-things/approve-all.....	60
3.4.17 Отхвърляне на всички изпратен добри дела - /good-things/discard-all	60
3.4.18 Дарения - /donations	60
3.4.19 Одобряване на изпратено дарение - /donations/approve	60
3.4.20 Отхвърляне на дарение - /donations/discard	60
3.4.21 Одобряване на всички изпратени дарения - /donations/approve-all	60
3.4.22 Отхвърляне на всички изпратен дарения - /donations/discard-all	60
3.4.23 Таланти - /talents	60
3.4.24 Одобряване на изпратен талант - /talents/approve.....	60
3.4.25 Отхвърляне на талант - /talents/discard.....	60
3.4.26 Одобряване на всички изпратени таланти - /talents/approve-all.....	61
3.4.27 Отхвърляне на всички изпратени таланти - /talents/discard-all	61
3.4.28 Регистриране на потребител - /register	61
3.4.29 Вход за потребители - /login.....	62
4. Програмна реализация.....	63
4.1 Реализация на базата данни	63
4.1.1 Пример 1 – Таблица потребители (users)	64
4.1.2 Пример 2 – Таблица роли (user_roles).....	65
4.1.3 Пример 3 – Таблица таланти (talents).....	65

4.2 Реализация на потребителската част	66
4.2.1 Потребителски роли	66
4.2.2 Сигурност на потребителите	68
4.2.3 Регистрация на нов потребител	68
4.2.4 Автентикация на потребители	70
4.2.5 Авторизация на потребители	71
4.3 Реализация на постовете в разните секции.....	72
4.3.1 Поставане от обикновени потребители (AJAX).....	72
4.3.2 Приемане/Отхвърляне на постовете от модераторите.....	75
4.3.3 Отхвърляне на съществуващи постове от админите	76
4.4 Реализация на намиране на топ 3 добри дела.....	77
4.5 Реализация на търсене на новина	78
4.6 Реализация на логовете (потребителски истории)	78
4.7 Реализация на организацията на потребителите.....	80
4.7.1 Промяна на ролята на определен потребител	81
4.7.2 Изтриване на определен потребител.....	82
4.8 Реализация относно информацията за вируса COVID-19	83
5. Ръководство на потребителя.....	86
Заключение	101
Литература	102
Приложения.....	103
1.Entities	103
2. User Model.....	106
3. UserRole Factory	107
4. Repositories	107
5. Services	108
6. Additional bussines logic.....	114
6. Controllers.....	117
7. Views(изгледи).....	125

Увод

Интернета като един глобален извор на информация все повече и повече става част от нашето ежедневие. Мощността, която той предлага стана основна и незаменима част от живота на всеки един човек. Уеб технологиите се развиват изключително бързо, предлагачи все по-напреднали услуги, качество и характеристики.

Идеята да се създават все по-добри и по-полезни сайтове е от голямо значение. Шансът да се улесни, подобри или замести някой процес от реалността е очакван с широко отворени ръце от страна на хората. Особено, когато се работи за развитието в града в който живеят. За това целта на текущата дипломна работа е именно разработка на уеб приложение за града Неготино.

Разгледани са повече подобни приложения, които имат идеята да представят особеностите на определен град. Някои от тях са: www.visitsofia.bg, www.varna.bg, www.burgas.bg. От направения анализ стана ясно, че всичките наблягат върху външното развитие на града, С демонстриране на главните особености на града дават информация, насочена към туристите и външния свят. Обаче, никой от тях не споменава вътрешното развитие на този град където основата са гражданите му. Не се предоставя възможност за свободата и гласа на народа, кой създава тези особености, кой ги поддържа тези особености, кой подобрява нещата в града, какво се случва в него и много други въпроси които представляват **вътрешно развитие на града**.

Забелзвайки основно тези недостатъци, задаваме си предизвикателство да създадем платформа, която главно ще се води от потребителите (гражданите на Неготино) и където всеки ще може да сподели какво е създал за целта на вътрешното развитие на общината.

За разработката на уеб приложението са избрани следващите технологии: Spring framework-а на програмния език Java, Thymeleaf, HTML, CSS, JavaScript, MySQL. Съобразението от тези неща позволява ползването на едни от най-популярните уеб технологии имплементирани архитектурата MVC (Model-View-Controller). Извършено и описано е проектирането относно всички спецификации, които предлага шаблона MVC и едно уеб приложение на Spring. Цялото това ще се състои от следните модули: изгледи, контролери, услуги, репозиторита, модели и ентитита.

Реализацията се състои в това да съществуват секции за отделните начини по които един град може да се развива вътрешно. Напредък, в който акцента ще се слага основно на потребителите и свежестта която носят в себе си. Платформа, която дава възможност за една по-добра и по-качествена солидарност, поддържане и помагане помежду гражданите и като цяло развиване на нещата бързо и едноставно. Като поставените цели са успешно реализирани и възможно е бъдеще развитие с увеличаване на областите, които обхваща.

1. Анализ на темата. Преглед на подобни приложения. Цели и задачи.

С течение на годините технологиите се развиват ежедневно все повече и повече. Броят на софтуерните продукти нараства непрекъснато. Нови и нови системи излизат във всеки един момент, с което даже и умишлено добре направени статистики, които да анализират бройката на нови продукти, трудно би се съобразили за точната бройка. С всеки нов проект, с всяка нова функционалност, идеите стават все по-големи и все по-примамливи. Пазарът е винаги гладен за нови възможности. Хубавите намерения да се създаде продукт който улеснява, помага или върши голяма работа на определен брой хора е добре дошъл в света, който с всеки нов ден е все по-напред и по-напред. Същността на продукта се намира в това, какво искаме да създадем и на кои хора може да „улесним живота“. Така, в последните няколко години според мои лични наблюдения все повече тези софтуерни продукти са предоставяни чрез уеб технологии. Възможностите на интернета стават все по-огромни, даже можем да кажем че без интернет сме „никой и нищо“. В последно време дори и изискванията на десктоп приложения намаляват и се заменят с такива в уеб платформа.

Микса от тези неща представлява един голям набор от функционалности, които могат да бъдат достъпни и ползвани. Някои от тези уеб приложения са по-прости, докато пък някои по-сложни. Някои от тези страници са насочени само да представляват общата функционалност на приложение със всичките негови спецификации. Те са само един вид „снимка“, която потребителите могат само да виждат и без да си взаимодействат с нея. Имат фиксиран брой на страници, които имат специфично оформление. Този изглед на уеб страниците примерно имат новините, спортните страници, сайтовете на университетите, основните и средните училища, министерства и много други. При тях не съществуват основните потребители и тяхната функционалност. Но, все пак може да имат отредени специфични профили като админ, модератор, които имат възможност да променят текста или изгледа на уеб страницата. Служат за статичната информация да се предаде в публичния свят и да е видима за всичките посетители. Напоследък все по-популярни са сайтове, където потребителите генерират съдържанието им. Те имат различни видове профили, които се разделят примерно на админи, модератори, обикновени потребители или пък гости. Основната разлика с първия вид софтуер е, че тук потребителите играят ролята да изнесат публичната информация в сайта или пък да достъпват функционалностите на уеб приложението за свои лични нужди. Такива сайтове са примерно Facebook, там може да видим, че потребителите са тези които водят сайта и създават един виртуален свят. Почти цялата информация е създадена точно от профилите му. Да имат приятели, да споделят снимки, видеа, статуси, линкове, да организират събития, да играят игри, да пуснат предаване на живо, да пускат анкети, да създават вътрешни сайтове и какво ли още не. Също така потребителите имат възможността да харесват или коментират на

споделените неща на приятелите им или на други публични постове. Въпреки това, те имат и възможността да си общуват с останалите потребители без това да е видимо за публичната част на сайта.

Както казват по-старите, винаги трябва да правим живота по-лесен и по-сложните проблеми да ги цепим на колкото е възможно по-малки и по-прости. Винаги трябва да гледаме да направим живота си по-едноставен и да сме гъвкави относно време и ресурси. Така ще имаме възможността за по-бързо и по-качествено развитие.

Подобна работа се случва в областта на уеб технологиите. В днешно време интернета е неделима част от нашето ежедневие. Това води до нуждата от разработване на уеб приложения от всякакво естество. Почти всички уеб сайтове имат основната дейност да споделят информации и решават проблеми относно улесняване на хората в определено нещо. Като за пример може да вземем сайтовете като Amazon който помагат хората да могат да купуват неща бързо и едноставно, банковите услуги за проверка на статуси, извършване на преводи и други подобни работи, сайтовете като Wikipedia за научни цели, социалните мрежи за социализиране и т.н. Всичките тези и много много други са създадени с една основна цел – да сервират на хората една идея по-напред към достъпа до нещата.

Така, в момента една малка част от съвкупността на тези приложения е от типът на портал за определен град. Примерно за столицата на България съществуват повече сайтове, които са умишлено създадени да подпомагат на хората да се запознаят по-добре със София. Нормално, подобни сайтове съществуват за почти всеки град в България и света. Тук отново идва основната идея, това да улесняват живота на техните граждани и туристите. Всичките сайтове от такъв тип се стремят да изкарат най-хубавите места които могат да се посетят в града, специалитети на ресторантите, история на града, култура, спорт, снимки, видеа и много други. Всичките тези неща са много хубави и много полезни. Примерно ако предположим че искаме да пътуваме в Бургас и сме граждани на друг град или даже и друга държава, как ще знаем какво да посетим, какво да видим, къде да си починем, или къде може да минем най-ефективно времето си. Разбира се, ще намерим решение, но трябва да попитаме повече хора, да се обаждаме на повече телефони и т.н. Тук идва моментът когато едно уеб приложение за този град може да ни даде много. След това и да искаме повече да навлезнем в някоя особеност на града, информацията, ще знаем за какво точно да се разпитаме и информацията от отделните хора ще ни дойде допълнителна.

Към момента за съжаление не съществува нито един уеб сайт за града Неготино, което за мен представлява основния мотив и предизвикателство.

1.0. Мотивация.

Връщам се във времето още когато бях дете във родния ми край Неготино. Малък град от 20 000 жители, където всеки се познава със всекиго. Още от малък започнах да се занимавам със спорт, поточно шах и баскетбол. Бях особено щастлив и горд когато бях част от отбора на клуба по шах на Неготино. Заедно, успяхме да станем 2-ри на държавното първенство в Македония. Сещам се и за успеха, който до момента ми е най-драг, 3-то място на държавното първенство за младежи до 16 години. Също така в тези млади години, когато основната област ми беше спорта бях част и от отбора на Неготино по баскетбол, който ми беше приоритет около 7 години. Но, тук не е време и място за моите успехи и падове. Трудих се много, но винаги имаше по-добри. Винаги имаше **таланти** и хора които бяха просто над мен. Възхищавах им се. Исках, много исках, да станат успешни и да представляват нашия град на държавно, па и европейско ниво защото го заслужаваха. Винаги, като тим дали в баскетболния или този в шах, пожелавах да напредваме като отбор. Всеки път, не знам защо, но от малък имах чувство че заедно сме по- силни, че това е правилното. Заедно учихме, тренирахме, бяхме заедно и в добро и лошо и си споделяхме всичкото. Едноставно отбора беше един вид „**фамилия**“. Не знам защо, но още тогава знаех че нещо липсва. Винаги се чудих защо **талантите** от столицата и най-големия град в Македония бяха „едно ниво“ по-напред от нас. Винаги бяха повече изложени и повече харесвани. Мислих си, анализирах колкото можех, но на края винаги излизах с резултата че нашите сили не бяха по-слаби от техните и на повече пъти ние сме излизали победителите. Но, както и да е когато израснах малко-повече този въпрос повече и не ми беше толкова странен. По-голям град – повече възможности.

От този опит, с течение на годините, задавах все повече нови и нови въпроси относно моя малък край – Защо да няма и при нас повече възможности? Защо да не си помагаме едни с други? Защо да не вървим като едно цяло, като отбор? Защо да не се социализираме повече? - Тези, и много други подобни въпроси завършваха със само един - Мога ли нещо да променя?

Много пъти съм фантазирал и веднага съм се отказвал. Много пъти летях все по-високо на небето и изведнъж спирах да махам с крилата, съответно падах още и още по- силно. Но, всеки път отново и отново се връщах на този въпрос, стана моя мания. Осъзнах че не съм толкова мощен и не мога да променям нещата както мен ми харесват. Но, все пък, някак си, имах уверенieto че мога да дам един малък пример за един мой измислен по-хубав свят и да оставам идеята на хората и съответно по-мощните да приложат за по-големи цели.

Беше 7-ми февруари 2015 година. Първа събудната мечта, фантазия от детството. Организирах **събитие** с основна цел **дарение**. Използвах отново областта на спорта, поточно **евента** представляваше турнир по волейбол. Всеки един имаше възможността

без значение на пол и възраст да участва с своя отбор и да покаже **таланта** си. От другата страна публиката имаше възможността да се наслаждава на майсторите във волейбола – нашите млади лица от Неготино. Входа в спортната зала беше срещу заплащане от около 65 стотинки (20 дена) – пари които на края бяха **дарение** за социално най-застрашеното семейство в Неготино за тази година. И така, два дни моите съграждани имаха шанса да слушат добра музика, да се видят едни с други и да гледат мачове, съответно да минат един малко по-различен ден.

Тази година във февруари се случи 6-тият такъв пореден вече традиционен **евент** който носи името „Неготино се буди“. Много ми е мило, особено че на хората им допадна **събитието** и при всяко начало на нова година сами почват да питат и помагат относно организиране на хепънинга. В междувременно мога да кажа че успя да се събудне още една моя мечта наречена Negotino Offline Fest. **Събитие** в което през деня на отворените спортни игрища на спортно рекреативния център „Младост“ се случват турнири в баскетбол, футбол, хандбал и тенис, докато през вечерта отново заедно се събираме на фестивал в който се наслаждаваме на локалните музически **таланти**. Просто най-хубавото за мен е че моментите които сме преживяли заедно на тези **евенти** остават като една хубава мемория зад мен, надявам се и всички останали.

Сега, повече не живея там. Но, както казват мъдреците „всеки камък си тежи на мястото си“, така и аз, малко странно, но все-още ми остават мечти който желая да ги направя за моя роден край. В последно време уеб приложенията от втория тип за които си говорихме се ползват масово. Така и аз съм малко по-активен на фейсбук което в момента представлява най-ползвано и най-уникално уеб приложение за социализиране. Там идеята ми е, да споделям всичко което е в полза на града Неготино. Така чрез постове успяваме да събираме финансови ресурси за **дарения** там където е най-потребно, да организираме **евенти**, да поздравяваме **добрите дела** на гражданите на Неготино, да се събираме, да шерваме **таланти**, да споделяме новини и т.н.

Супер, но все-някак забелязвам че идеята с фейсбук е много глобална, постовете са разпределени низ всичките профили, някои от те профили шерват поста само за техните приятели и не са публични, изисква време да потърсиш къде точно, какво и за коя цял е обявено защото основната страница на фейсбук се обновява почти на всяка секунда и всичките постове са някъде вече далече.

Попитах се отново – Защо да не е на едно място? С хубав графичен интерфейс? С ясна цел – напредък на града? С ясни информации разпределени по секции?

Цялата моя история и философия писана по-нагоре се свежда на само едно – идеята на тази дипломна работа.

Благодарение на уменията и знанията които придобих на Техническия Университет в София накараха ме да мисля една идея по-нагоре. Университета който ми отвори

вратите и цялостно научи на мощността на програмирането като една област която също така ми дава възможността на един много хубав начин да направя най-големите стъпки към събъдане на най-голямата мечта и решаване на всички мои реторически въпроси от детството – **развитие на малкия град Неготино**.

1.1. Анализ на тематичната област.

Всъщност, въпреки че горе-изопнатата идея на почти всички портали за градовете е добра и супер полезна за хората, целта на тази дипломна работа не е точно в това някой да гледа външните особеностите на моя град. Тази идея е статична информация която не представлява много затруднение, не е лесна, но е планирана да бъде добавена в бъдещето на това уеб приложение.

За конкретната идея на този проект идва момента когато въведението на тази дипломна работа засяга ми да си задам предизвикателството за да създам един малко по-различен уеб сайт от такъв тип. Реализацията да бъде базирана на **гласа на народа**. Относно, уеб приложението да представлява платформа която ще се базира върху потребители и техните потребности. Идеята за това лежи в това че според мен в един град най-важно е да се знае какво правят хората, какво им трябва и как да има по-добро развитие. Да има солидарност, подкрепа и разбиране помежду хората. Всеки един човек да има правата да споделя и чете информацията която го засяга за своя град.

Както казват живота е най-големия учител. Опита който трупаши през годините кара те на всеки следващ път да направиш работите все по-добро и по-добро. Така, в съобразение отново с анализа и моята интересна история първата част от уеб приложението с която се хванах е точно поддържката на **талантите** в Неготино. Винаги се възхитавах на хората които силно се опитват да успеят през живота си и исках да са забелязани малко повече. Така, за целите на това уеб приложение имам идеята всеки един да може да сподели колкото познава хора които са добри в техните области. Така, явно ще се споделят талантите и всеки ще има възможността да се радва за това какви хора живеят в Неготино. Същевременно, ако някой треньор за определен спорт, човек който търси певци, танцьори, музиканти за своята музическа група, парти или каквото и да е, човек който търси фотографер, сликар, или какъв и човек да се търси може да се намери точно в секцията на **талантите**. С това, според мен, може да се намери точния човек за определена работа, а от другата страна хората които искат да успеят да живота да получават оферти и да напредват все-повече и повече.

Живота по-някога знае да бъде суръв и донесе проблеми които според мен никой не ги заслужава. За съжаление, тези трудни моменти идват неочеквано, изведнъж хвърлен си в средата на морето и трябва да изплуваш към брега. Но, ние като хора, създадени сме

на този свят за да си помагаме и поддържаме. За това, следващата секция която добавих са **даренията**. Тук всеки гражданин на Неготино ще има възможността да поиска помощ с подробни информации за него или някой друг. Това може да го направи в явен или анонимен начин. Когато сме заедно сме по-силни. С тази идея всеки които има възможност ще помогне на определената нужда. Докато, другата част от секцията **дарения** е всеки който няма повече нужда от определено нещо или пък иска да подари нещо, да има възможност явно да сподели информация и това нещо да бъде дарено там където е най-потребно. Примерно ако някой си купи нов телевизор и повече няма нужда от стария, може да го сподели явно на уеб сайта на Неготино и да се намери фамилия която няма възможност си купи телевизор, па съответно да им го подари и да ги направи много щастливи.

Един град се отличава от друг по това какви хора живеят в него. По-точно, може да се каже какво правят за него. Има хора които се събуждат с мисълта какво ще направя днес за да направя този свят по-хубаво място за живееене. Ходят на работа, имат си фамилии, но в свободното време правят работи които са полезни за мястото в което живеем или пък за други хора. Като един пример може да се вземе ако някой изчисти парк или помещение което е притежание на града, помогне на стар или болен човек, на бедна семейства, се грижи за определени домашни животни които са бездомни и т.н. Всичките тези и много други **добри дела** според мен трябва да са публикувани. Трябва да бъдат похвалени хората който се грижат за своя град. С тази идея искам да мотивирам и другите хора да вземат пример и последват стъпките на своите съграждани. Така ще имаме шанса да се развиваме и живеем в едно по-хубаво място. Отново всеки потребител ще може да похвали себе си или някой съгражданин за **доброто дело** който е направил.

Следващата секция която мисля че е подходяща са **събитията**. Тук идва идеята за по-голяма солидарност, социализиране и напредък на града. Чрез тази част от платформата на това уеб приложение всеки ще може да си пусне **евент** с подробни информации и явно да покани хората на определено място, в определено време с определена цял. Тук няма значение какъв е типа на **събитието**, не е задължително да е от областта на култура, спорт или друго. Примерно може да се добави събитие за правене на определена добра работа. Създаваш **събитие** в платформата на Неготино и пишеш че доброволно ще слагаш и поправяш пейки на парка. Извикваш явно кой иска да помогне и получаваш група от хора с което хем имаш помощ, хем спестяваш на време в което може да направите друга добра работа. Друга пример е примерно имаш желание да играете футбол, но нямаете достатъчно хора. Пускаш **евента**, в него си има място, време, контакт и събиращ желаещи за да се забавлявате и дружите.

Една община също така е „живя”, когато нещо се случва в нея. Дали това е някое ново заведение, нова оферта, нови документи, нови спортни мачове, нова информация в областта на културата или историята, нова информация относно здравето и много други

подобни неща. Всичките тези нови работи спадат в секцията **новини**. Там, отново идва гражданина на Неготино като човек кой има възможността да сподели с останалите **някоя новина**.

Същевременно, съобразено с текущата ситуация в света хубаво е гражданите да са информирани относно вируса наречен **корона** или COVID-19. Според мен, би било много полезно когато ще имат цялата информация на длан. Така, вместо винаги да чакат да дойде часа на новините на телевизията, ще могат във всеки нов момент да се информират за текущото състояние както в тяхната земя, така и на световно ниво.

Тези секции и други които може да се добавят в бъдеще представляват **вътрешно развитие** на града Неготино. За тази цял, точно уеб технологиите дават най- хубава възможност за публично да се споделя тази информация и създаде нещо ново, нещо по-различно. Точно тези публикации които са позволени от всеки един гражданин, дават свободата на думата на физическите лице. Това право дава възможността за еднаквост помежду жителите и им дава един хубав шанс да не тормозят себе си изпращайки към повече портали и надявайки се че поне един ще обяви тяхното желание.

1.2. Съществуващи решения

1.2.1. www.visitsofia.bg

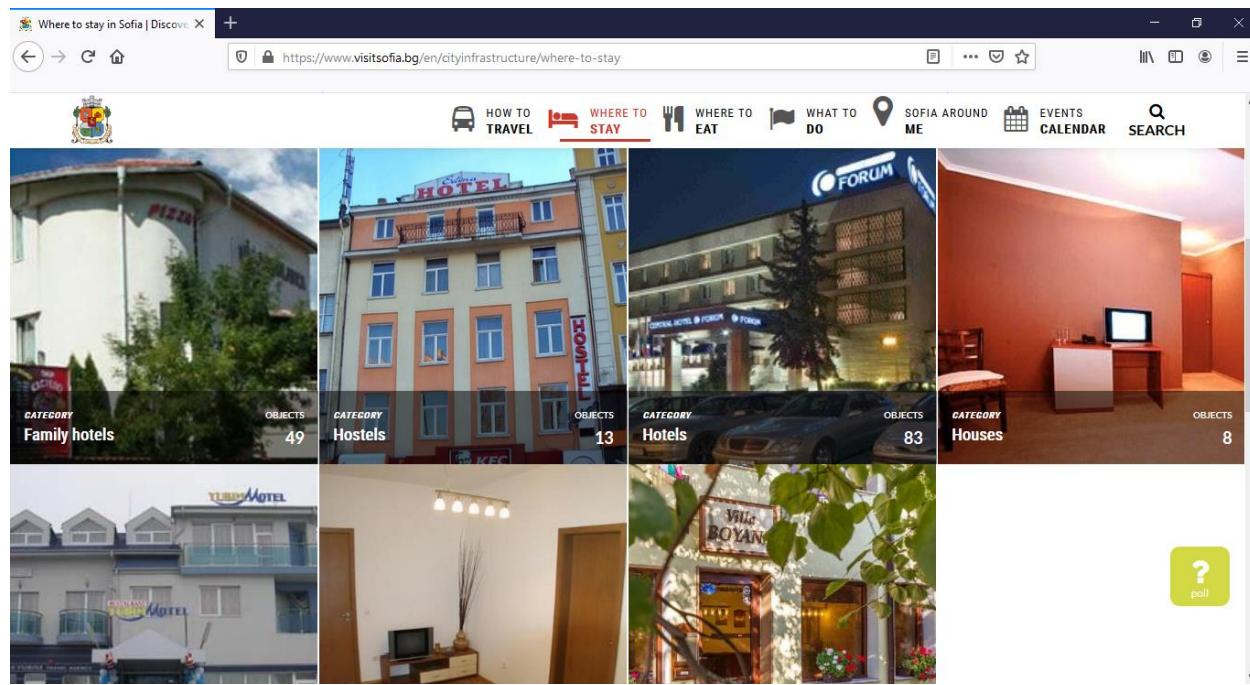
Visitsofia е сайт за столицата на България. Основната цел е да се докаже че в София има какво да се види и да се развие туризма в нея. Уеб приложението предоставя пълна информация за всичките особености на града София.

Част от основните функционалности на приложението са:

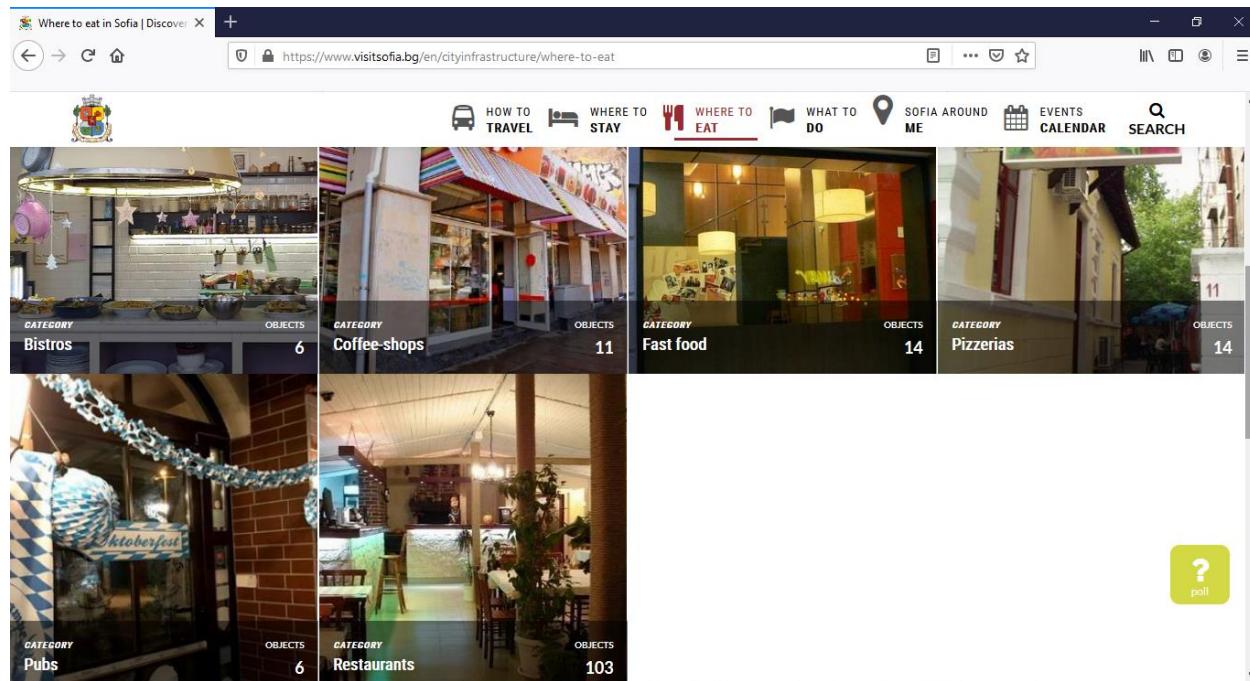
- Как може да се пътува (фиг. 1.1)
- Къде може да се нощува и остане (фиг. 1.2)
- Къде може да хапнем (фиг. 1.3)
- Какво може да се прави (фиг. 1.4)
- Събития (фиг. 1.5)
- Състояние относно Корона вируса
- Контакти

The screenshot shows a web browser window with the URL <https://www.visitsofia.bg/en/howtotravel>. The page title is "How to travel". The header includes the VisitSofia logo, a search bar, and navigation links for "HOW TO TRAVEL", "WHERE TO STAY", "WHERE TO EAT", "WHAT TO DO", "SOFIA AROUND ME", "EVENTS CALENDAR", and "SEARCH". Below the header is a banner image of a yellow bus. The main content area features the heading "How to travel" and four thumbnail images: a yellow bus, a yellow taxi, a parking garage, and a car interior with a "Rent-a-car" button. A green callout bubble with a question mark icon is pointing to the "Rent-a-car" button.

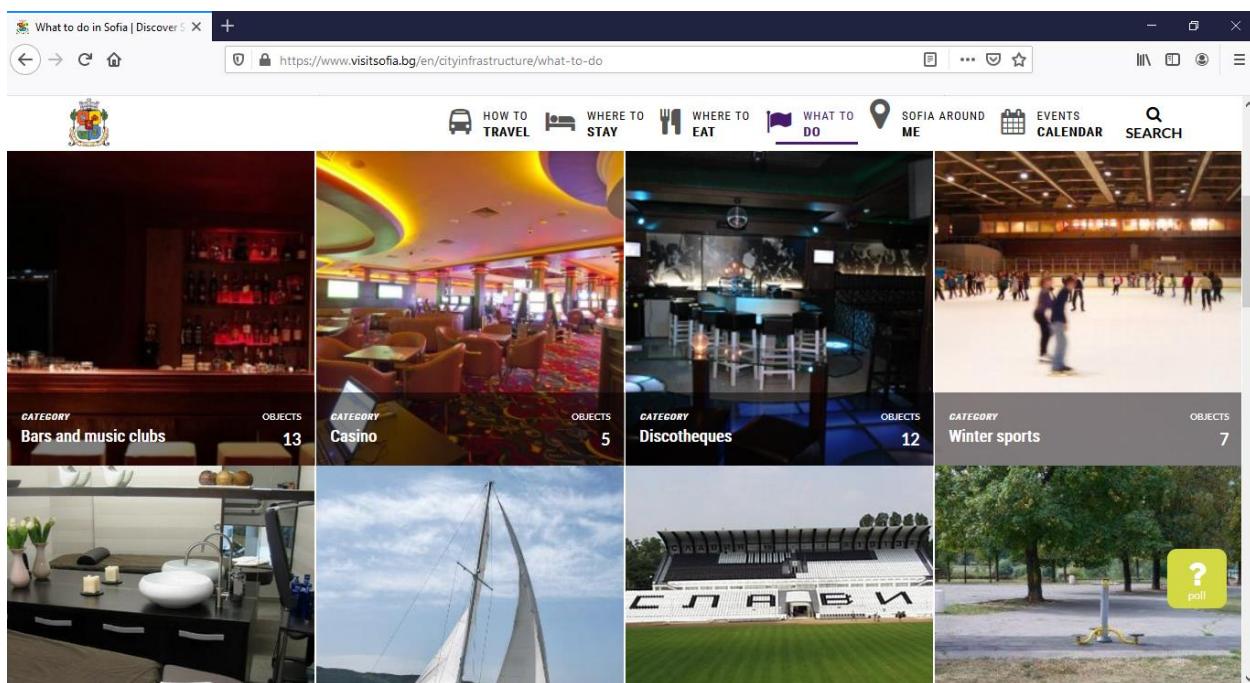
1.1 – Как може да се пътува



1.2 – Къде може да се нощува



1.3 – Къде може да се хапне



1.4 – Какво може да се прави

1.5 – Събития

Положителни страни на приложението:

- Добър графичен интерфейс
- Смяна на езика на уеб сайта

- Голям набор от информации
- Ясно разредени секции
- Лесен и удобен начин за достъпване на нещата
- Красиви снимки

Негативни страни на приложението:

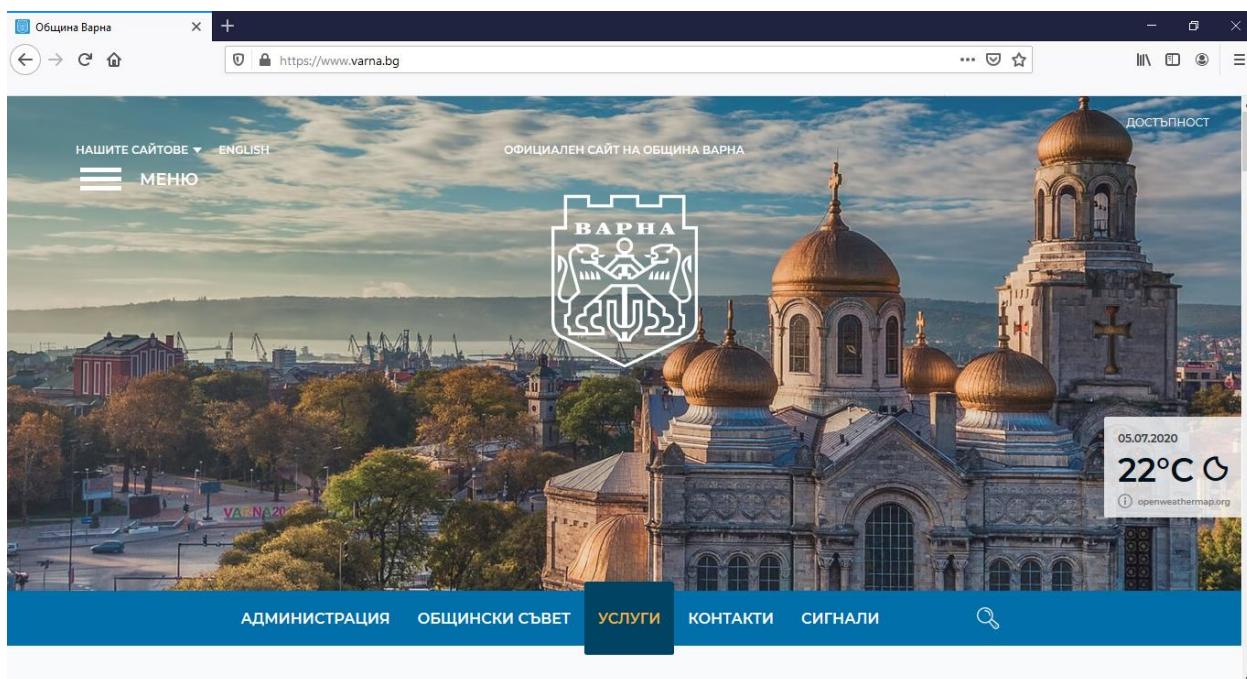
- Бавно зареждане между отделните страници и преходите между тях
- Селективност (само определен обекти са описани, не всичките)
- Няма възможност за потребители които да вкарват информация (В сравнение с моя проект. Не е негативна страна, това е решение на създателя)

1.2.2. www.varna.bg

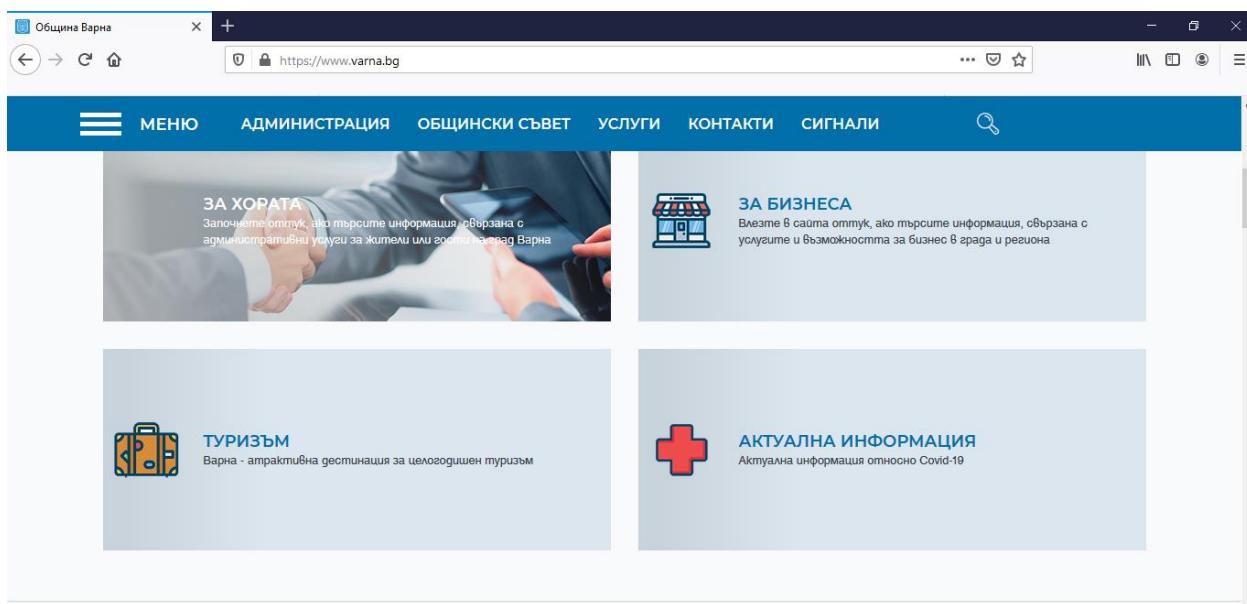
Този уеб сайт е официалния сайт на община Варна. Цели се както на туризъма, така и на полезните неща които нуди общината за своите граждани.

Част от основните функционалности на приложението са:

- Администрация
- Общински съвет
- Услуги
- Контакти
- Сигнали
- Туризъм (фиг. 1.7)
- Особености на Варна (фиг. 1.8)
- Новини – (фиг. 1.9)
- Документи



1.6 – Начална страница



АКЦЕНТИ

<https://www.varna.bg/bg/547>

1.7 – Основни функционалности

АКЦЕНТИ

БЪРЗИ ВРЪЗКИ

ПРОВЕРКА НА ЗАДЪЛЖЕНИЯ ЗА МЕСТНИ ДАНЪЦИ И ТАКСИ

ОБЩЕСТВЕНИ ПОРЪЧКИ – ПРОФИЛ НА КУПУВАЧА

ПРИЕМ В ДЕТСКИТЕ ЗАВЕДЕНИЯ И ПГ в УЧИЛИЩА

МОРСКА ГРАДИНА

НОВИНИ

НОВИНИ ОТ ВАРНА

1.8 – Особености на Варна

Съобщение във връзка с полагане на хоризонтална маркирова
Във връзка с Въвеждането на втория етап от „сина зона – широк център“ на
3-ти, 4-ти и 5-ти юли ще бъде положена хоризонтална маркировка по ул.
„Ангел Георгиев“, ул. „Христо Поповиц“, ул. „Христо Самаров“, ул. „Георги
Живков“, ул. „Парижка комуна“ и ул. „Никола Кънев“.

2 Юли 2020 Новини

Започна прием на документи за...

Математическата гимназия - с...

Отрицателни проби в детска...

75 фирми от Варна одобрени за...

1.9 – Новини**Положителни страни на приложението:**

- Бързо зареждане

- Доста хубав графичен интерфейс
- Ясно е определено кое къде се намира
- Лесен достъп до документи
- Лесен достъп до информация
- Смяна на езика

Негативни страни на приложението:

- Не-достатъчно развит туристически гайд за град на морето (предполагам съществува друг)
- Няма възможност за потребители които да вкарват информация (В сравнение с мой проект. Не е негативна страна, това е решение на създателя)

1.2.3. www.burgas.bg

Отново официален уеб сайт на една община, в случая Бургас. Освен, че има предназначение за документи и административни неща, включва и други полезни неща които се случват в момента в общината.

Част от основните функционалности на приложението са:

- Администрация
- Информация и услуги
- Новини (фиг. 1.11)
- Събития (фиг. 1.12)
- Транспорт
- Данъци и такси
- Е-услуги (фиг. 1.13)
- Образование (фиг. 1.14)
- Туризъм (фиг. 1.14)

- Спорт (фиг. 1.14)
- Култура (фиг. 1.14)
- Здравен каталог
- Бюджет

ИНФОРМАЦИЯ ЗА БОРБА С КОРОНАВИРУС

Пазете себе си и другите от заразяване!

ПРАВИЛА ЗА БЕЗОПАСНОСТ • ДАРИТЕЛСКИ СМЕТКИ • ГОРЕЦИ ТЕЛЕФОНИ • АКТУАЛНА ИНФОРМАЦИЯ

ДАНЪЦИ И ТАКСИ

БИЗНЕС

ЛИЧНА ПОМОЩ

1.10 – Начална страница

НОВИНИ

Общината ремонтира покрива на къщата на Духтев в Морската градина

Откриха новия Център за обществена подкрепа в Бургас

Невероятна атмосфера на Петро уикенда в Бургас

Близо 300 шестци донесоха седмокласници на Бургас

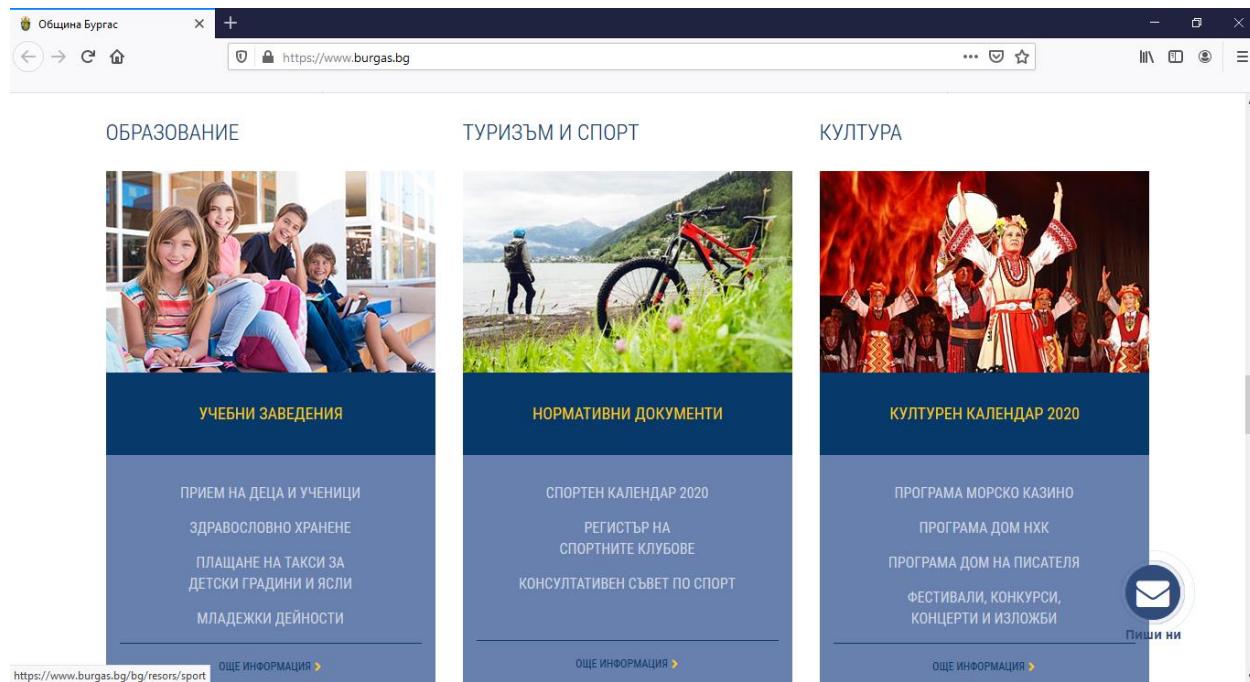
1.11 – Новини

The screenshot shows the "ПРЕДСТОЯЩИ СЪБИТИЯ" (Upcoming Events) section. It features two event cards: "Лято 2020 Уикенд в Бургас!" (Summer 2020 Weekend in Burgas!) from June 12 to 27, 2020, and "РЕТРО УИКЕНД В БУРГАС 3-5 ЮЛИ" (Retro Weekend in Burgas July 3-5). A sidebar on the right allows users to search for events by date or keyword. Below the events, there's a "ИНФОРМАЦИОНЕН БЮЛЕТИН" (Information Bulletin) section and a "Пиши ни" (Write to us) button.

1.12 – Събития

The screenshot shows the "ИНФОРМАЦИОНЕН БЮЛЕТИН" (Information Bulletin) section, which encourages users to subscribe for news, events, and announcements. Below it is the "Е-УСЛУГИ" (e-Services) section. This section lists various electronic services available through the portal, such as "АДМИНИСТРАТИВНИ УСЛУГИ" (Administrative Services), "ДЕЛОВОДНА СПРАВКА" (Business Certificate), and "ЕЛЕКТРОННА СИСТЕМА ЗА ЗАПИСВАНЕ В ПЪРВИ КЛАС" (Electronic System for Recording in First Class). To the right, there's a "Е-ПРАВИТЕЛСТВО" (e-Government) section with links to business and citizen services, and a "Е-УСЛУГИ" (e-Services) section for services provided by the Municipality of Burgas, NAP, NOI, etc.

1.13 – Е-услуги



1.14 – Образование, туризъм и спорт, култура

Положителни страни на приложението:

- Включени почти всички особености на един град
- Широк обем на информация
- Бързо зареждане
- Добре са организирани нещата
- Добър графичен интерфейс
- Смяна на езика
- Лесен достъп до информация

Негативни страни на приложението:

- Няма възможност за потребители които да вкарват информация (В сравнение с моя проект. Не е негативна страна, това е решение на създателя)

1.3. Цели и задачи

За съжаление, колкото повече се развива технологията в света, толкова повече хората стават странни и се затварят в себе си. Жivotът в 21-ви век стана малко по-различен и все повече сме зависни от „виртуалния свят“. По-принцип този „виртуален свят“ е предназначен за много полезни и хубави работи, обаче реалността е че все повече си губим времето на празно по социалните мрежи. Така, много често можем да мине деня и без да направим поне нещо малко полезно преди всичко за себе си, след това за хората около нас, нашия град или света.

За тази цел основното предизвикателство, което си задавам е да дам възможността на хората явно, чрез една пристапка да направят нещо, което ще е малко като обем, обаче от голямо значение за **развитието и бъдещето на града Неготино**. Разработване на софтуерно решение е най-подходяща за реализиране на моята цял. Решението се изразява в това да се създаде уеб приложение, което ще може да бъде достъпно както от гражданите на града, така и от всички други. Основната идея се базира върху вътрешно развитие на града, живеенето в един по-хубав свят. Сват в който си помагаме едни с други и улесняваме и правим света едно по-хубаво място за живеене.

Целта ми е, развитието да се базира от самите граждани на Неготино и техните желания. Те ще са основния извор на информация. С регистриране на свой профил или бързо логване чрез някоя от по-големите социални мрежи в сайта на Неготино да имат пълни права да споделят неща в отделните части на сайта. Така, обикновените потребители представляват авторите на всичките постове в уеб приложението. Основните секции на портала ще са: **таланти, дарения, добри дела, събития и новини**. В секцията таланти ще се представляват хора, които са от огромно значение за общината, хора които отличават града. В даренията ще има възможност за търсене на помощ или споделяне на нещо, което искаме да подарим. Секцията за добри дела ще е място за поздравление на хора които създават/правят нещо добро за града или техните съграждани. Събитията ще са частта където може да споделяме информация за определен евент от тип култура, спорт или нещо друго, също така и за възможност за бързо и лесно събиране на хора за някоя ежедневна работа от типа на помагане на някого, чистене на помещение, спортни игри, дружене, пътуване, алпинизъм и много много други подобни. Секцията новини ще предоставя възможност за ефективен преглед за това какво се случва в момента в общината. Същевременно малко по-различен вид секция която се в съобразение към моментното състояние в света в 2020 – корона вируса. Там имам за цел хубаво предоставяне на статистика относно общата бройка заболяли, заразените за деня, умрелите в текущия ден, общия брой смъртни случаи, лица, които са успели да победят вируса в текущия ден и общия брой на лица, които са победили корона вируса до сега.

Тази информация имам желание да бъде показана както глобално за света, така и за нашата и всички останали държави.

Сайта ще е от публичен вид, и всички ще могат да гледат и споделят направените постове. За споделяне на информация ще е необходимо да имат направена регистрация или да е направена автентикация чрез някоя от социалните мрежи. Така ще пренебрегнем нежелани анонимни постове. Също така, за добро филтриране на изпратените постове от потребителите, трябва да има някой над тях, който да каже дали тази информация става и се струва да бъде пусната явно. Така над обикновените потребители идват модераторите. Те са с едно ниво по-висока роля и имат дейността да приемат или отхвърлят request-натите постове. Също така ще имат възможност да гледат логовете където ясно се гледа кой потребител каква функция е извършил на сайта. Ще бъде реализирано още едно ниво над модератите, това са админите. Админите пък, ще имат същите права като модераторите и повече други. Те ще могат да добавят/трият модератор, добавят/трият админи, приемат или отхвърлят request-нати постове и трият вече съществуващи постове. Над тях съществува и още една роля която всъщност е притежателя на приложението. Конкретно тук, това може да се каже че съм аз. И аз имам ролята админ обаче с един допълнителен епитет - коренен. Това е направено така за да се пренебрегва възможността някой админ да изтриве моята роля и да стане притежател. Аз като профил – създател на приложението, ще имам пълните възможности, обаче никога няма да може да бъда мащнат от другите, докато аз ще мога да имам това право.

Така до момента, дефинирахме като цели следващите роли: **гост, обикновен потребител, модератор, админ, коренен админ** и следващите секции: **таланти, дарения, добри дела, събития, новини, корона вируса, логове, организация**. В следващите секции от тази дипломна работа ще се разясняват работите малко по-подробно.

За реализиране на тези цели си задавам следващите задачи:

- Възможност за регистриране на потребител
- Възможност за логване в сайта(автентикация)
- Възможност за помнене на логнатия акаунт
- Автентикация на потребителите с потребителско име и парола
- Авторизация на потребителите относно дадената роля
- Сигурност на профилите
- Достъпване само до определени ресурсите от страна на посетителите които не са логнати
- Да се предостави точната функция на потребителя относно неговата роля и нищо повече
- Възможност за споделяне на постове от потребителите
- Бързо зареждане помежду отделните страници

- Хубав потребителски интерфейс
- Удобно и лесно за ползване
- Системата да работи успешно с голям набор от информации
- Бърз и лесен достъп до информациите
- Красив feedback за нашите потребители
- Качествени функционалности
- Обективност на публикациите
- Възможност за споделяне на постовете на социалните мрежи
- Класация на топ 3 граждани които направили добри дела за текущия месец
- Корона вирус секция с ясни информация
- Логове с информация кой какво е направил, достъпни от модераторите панагоре
- Част за организация на потребителите и ролите, достъпна само от админите в която ще има възможност за смяна на определена роля или триене на потребител който е нарушил правилата на сайта

Всички тези цели и съответно задачи трябва да са добре компенсирани и реализирани в съображение с време, качество и ресурси и т.н.

2. Обзор на използваните технологии

2.1. Java

Java е език за програмиране на високо ниво, първоначално разработен от Sun Microsystems и издаден през 1995 г. Java работи на различни платформи, като Windows, MacOS и различните версии на UNIX.

Едни от най-важните части на Джава са:

- Обектно ориентиран – всичко в Java представлява даден обект. Това дава възможността за едно хубаво съотношение с реалния свет където имаме различни обекти и всеки от тях наследява или притежава друг обект.
- Простота – ако познаваме принципите на ООП (Обектно-ориентираното програмиране), много лесно може да пишем на Java
- Сигурност – С фийчърите на Java може да създаваме virus-free, tamper-free системи. Техниките за автентификация са базирани с енкрипция с публичен клуч.
- Архитектурно неутрален – компайлерот на Java генерира архитектурно неутрален обект в формата на файл, което позволява компилирания код да бъде пуснат на еди-коя машина която притежава Java runtime system.
- Портируем – с факта че е архитектурно неутрален и имайки независими аспекти на спецификациите прави Java портативен
- Робустен – Java прави удобността за елиминиране на грешките главно с помощта компайл и рънтайнм чекването.
- Ефективен
- Много-нишков
- Динамичен

Една от най-големите предимства на Java е вграденото автоматично управление на паметта. То освобождава програмистите от сложната задача сами да заделят памет за обектите и да търсят подходящия момент за нейното освобождаване. Това сериозно повишава производителността на програмистите и увеличава качеството на програмите, писани на Java. За автоматичното управление в Java се грижи Java Garbage Collector-от.

Той живее в виртуалната машина на Java. Всеки може да си направи свой колектор, обаче трябва да спазва спецификации на JVM. Всичките garbage collector-и

имплементират генерационна стратегия която категоризира обектите по „възрастта“. Идеята е че повечето обекти са създадени с кратко действие и трябва веднага да се изтрият. Така трите генерации които се споменават при garbage collector-а са: млада генерация, стара генерация, постоянна генерация.

Java програмния език е един хубав начин да развиваш своите софтуерни умения. Той е широко използван и доста мощен.

2.2. Spring Boot

Spring Boot е най-популярния open source framework базиран на програмния език Java. Написан от Род Джонсън, обявен официално в Йуни 2003, с лицензия от Apache 2.0 Spring framework-а предоставя много функции, които улесняват разработването на Java-базирани enterprise приложения. Тук веднага идва въпроса – какво е framework?

Най-общо казано в програмирането “framework” наричаме софтуер, който носи в себе си базови функционалности в зависимост от типа на приложение, което искаме да изграждаме. Това е така, тъй като приложенията от даден тип отговарят на определени нужди и така в един момент те започват да предлагат сходни решения и възможности. Разработчикът взема този общ софтуер и чрез добавяне на необходимия му код, той създава собственото приложение от което има нужда. Казано иначе, един фреймуърк (или наричан още „технологична рамка“ или „софтуерна рамка“) представлява универсална, преизползваема софтуерна среда, която е стандартизиран способ за изграждане и разполагане на приложения. По този начин технологичните рамки ускоряват разработката на софтуерни приложения, продукти и различни решения. За целта технологичните рамки могат да включват и допълнителни приложения и инструменти като компилатори, библиотеки, application programming interface-и (API-та), като всичко това ни позволява пълноценна разработка. Технологичните рамки имат няколко основни свойства, които ги различават от софтуерните библиотеки: инверсията на контрол (при което framework-ът „диктува“ на разработчика как да структурира програмата си и „следи“ поведението на потребителя), добавянето на допълнителен код (разработчикът може да модифицира технологичната рамка, чрез добавяне на външни модули код, които да му дадат нови възможности) и немодифицируем код (въпреки че разработчикът може да добавя външни модули, той не може да редактира основния код на framework-а по никакъв начин).

Милиони девелопери низ света ползват Spring framework-а за да създадат високо качество, лесно тестим софтуер с гъвкава възможност за преизползване на кода. Spring е лек framework като основната версия му е около 2MB. Основните характеристики на Spring Framework могат да бъдат използвани при разработването на всяко Java

приложение, но има разширения за изграждане на уеб приложения върху платформата Java EE. Spring framework-а има за цел да улесни разработването на J2EE и насърчава добрите програмни практики, като активира модел за програмиране, базиран на POJO.

Едни от най-големите бенефити на Spring са:

- Spring дава възможност на разработчиците да се разработват приложения от типа на ентерпрайз, използвайки POJO. Предимството да използваме само POJO е, че не се нуждаем от контейнерни продукти на EJB, като например сървър за приложения, тук имаме възможност да използваме само контейнер за сервлети, като Tomcat или някакъв друг подобен.
- Spring е организиран по модерен начин. Въпреки че номера на пакетите и класовете са големи, ние трябва да се грижим само за тези от които имаме нужда.
- Тестовете в апликация писана на Spring са прости и лесни
- Spring е много добре дизайниран web MVC framework. Спазва принципите и нуди лесна интеграция на кода по принципа на MVC (Model-View-Controller).
- Spring предоставя удобен API за превеждане на специфични за технологията изключения (хвърлени от JDBC, Hibernate или JDO, например)
- Тенденция към завхащане на по-малко ресурси за да може да се използва в компютри с по-малка мощност

Когато пишем комплексна Java апликация, класовете трябва да са колкото е по-възможно не-зависими един от други за да направим шанса за преизползване на тези класове по-лесен и да може по-лесно да ги тестваме при unit тестове. Така, технологията с която Спринг най-много може да се похвали е Dependency Injection (DI). Това позволява за спояване на тези класове заедно и по-също време чуването им на независими. Примерно ако имаме един клас А който зависи от клас Б. Така, по някакъв начин класа Б трябва да бъде включен в А. Тук, идва дейността на инжекция за да реши основните проблеми който ще изникнат след това и представлява сърцето на Spring framework-а.

Друг много важен компонент на Spring е Аспектното Ориентирано Програмиране (АОП). Аспектно-ориентираното програмиране (АОП) е относително нова парадигма в програмирането която цели да се увеличи модулността на софтуера и включва методи и инструменти които помагат за това. Този принцип помага да обедини множеството абстракции които се пресичат в програмния код. Като пример, може да се посочи „логването“ в дадена система, тъй като в зависимост от избрания вход в системата, поведението на всички методи и класове може да е различно. Така, идва възможността с един код който ще е написан на страна винаги да имаме възможността да достъпиме примерно от потребителското име или да се подсигурим че профила е логнат.

Типични примери за аспекти са: сигурност, качество на услугата (QoS), поддръжка на транзакции, синхронизация, трасиране и т.н.

2.3. MySQL

Релационните бази данни са такива бази данни, при които информацията се съхранява в таблици, които са съставени от записи и атрибути(свойства/полета). Терминът „релационна база данни“ за първи път е предложен през 1970 година от Едгар Код.

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 други. Предимствата на базите данни като средство за съхранение на информация са намаляване на обема на данните – отстраняване на повторение и осигуряване на интегритет. 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 трябва стара. Съхранението на данните в отделно хранилище гарантира, че дори и при повреда на софтуера, данните ще останат. 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 т.н. СУБД може да извършва регулярни операции върху данните – back up, restore и други.[7]

MySQL е много поточна, многопотребителска, SQL система за управление на бази данни (СУБД). MySQL AB разпространява MySQL като свободен софтуер под GNU General Public License (GPL), но също така под традиционните за комерсиален софтуер лицензи за случаи, когато използването е несъвместимо с GPL. Достъпни са програмни интерфейси, позволяващи програми, написани на различни програмни езици да имат достъп до MySQL бази данни. Такива са: C, C++, C#, Delphi (чрез dbExpress), Eiffel, Smalltalk, Java (с директна поддръжка), Lisp, Perl, PHP, Python, Ruby, REALbasic (Mac), FreeBasic, и Tcl, като всеки от тях има специфичен програмен интерфейс. Интерфейс тип ODBC наречен MyODBC позволява на други програмни езици, които поддържат ODBC интерфейс да комуникират с MySQL база данни, например: ASP или Coldfusion. MySQL е написан основно на ANSI C.

MySQL е популярна за интернет приложения като MediaWiki или Drupal и е база данни в LAMP, MAMP и WAMP платформи (Linux/Mac/Windows-Apache-MySQL-PHP/Perl/Python), и за софтуера с отворен код Bugzilla – приложение за проследяване на грешки. Нейната популярност като Интернет приложение е тясно свързана с популярността на PHP, като често, комбинирани с MySQL, са наречени *Динамичното дуо*. Лесно е да се намерят много източници, които ги комбинират в интернет статии или книги (*PHP and MySQL for Dummies*, *PHP and MySQL Bible*, *Beginning PHP and MySQL*, и други). В тези книги се твърди, че MySQL е по-лесен за изучаване от други бази данни. В пример от книгата за глупаци се посочва, че от MySQL може да се излезе с командите *exit* или *quit*, но това е вярно и за много други бази данни.

За администриране на MySQL може да се използва включеното приложение работещо от Командна линия (командите: mysql и mysqladmin). Също достъпни за сваляне от интернет сайта на MySQL са GUI приложения за администриране: MySQL Administrator и MySQL Query Browser.

MySQL работи на много различни платформи - включващи AIX, BSDi, FreeBSD, HP-UX, GNU/Linux, Mac OS X, NetBSD, Novell, NetWare, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, Windows 95, Windows 98, Windows ME, Windows NT, Windows 2000, Windows XP и други версии на Windows.

2.4. HTML5

HTML означава Hyper Text Markup Language. Това е стандартния маркъп език за изкарване на информация на уеб сайт. Първото публично достъпно описание на HTML е документ, наречен „HTML tags“, първо посочен в Интернет от Тим Бърнърс-Лий в края на 1991 г. Той описва 18 елемента, включващи начална, сравнително опростена конструкция на HTML.

HTML е език за маркиране, който уеб браузърите използват за да интерпретират и създават текст, изображения и други материали. Оригиналните характеристики за HTML са дефинирани в браузъра, като могат да бъдат променяни или подобрявани с допълнителна употреба на CSS (Cascading Style Sheets).

Описанието на документа става чрез специални елементи, наречени HTML елементи или маркери, които се състоят от етикети или тагове (HTML tags) и ъглови скоби (като например елемента <html>). 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 примерно <h1> и </h1>

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. Например <маркер> Никакъв текст. </край на маркера>. 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 оформлен и върху монитора на персонален компютър, и върху миниатюрния дисплей на пейджър или мобилен телефон.

HTML може да прикрепя скриптове писани на езици като JavaScript, което променя поведението на уеб страницата. Може да се използва Cascading Style Sheets (CSS), който определя изгледа и оформлението на текста и други материали. World Wide Web Consortium (W3C) поддържа и двете CSS и HTML и наследчава използването на CSS в HTML страниците от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне.

HTML5 е последният голям проект от HTML стандарта. Въпреки че стандартът е бил завършен (като версия, препоръчана за използване) едва през 2014 г. (докато предишната, четвърта, версия била публикувана през далечната 1997 г.), още от 2013 г. той оперативно се поддържал от браузърите, а разработчиците използваха работния стандарт. Както своите предшественици HTML 4.01 и XHTML 1.1, така и HTML5 е маркиращ език за създаване и предоставяне на съдържание в уеб пространството (World Wide Web).

HTML5 е смесица от много функции, въведени от различни спецификации, заедно с тези, въведени от софтуерни продукти, като уеб браузърите.

По-специално, HTML5 добавя много нови синтаксови функции. Те включват `<video>`, `<audio>` и `<canvas>` елементи, също така и интеграция на SVG съдържание, което е създадено да подобрява работата с мултимедийно и графично съдържание в уеб пространството без плъгини и техните APIs. Други нови елементи, като `<section>`, `<article>`, `<header>` и `<nav>` създадени да подобрят семантичното богатство на документите. Други елементи са премахнати. Също така са въведени и нови атрибути. Някои елементи, като `<a>`, `<cite>` и `<menu>` са променени.

2.4. CSS

Cascading Style Sheets (CSS) е език за стилизиране на уеб сайтовете. Предоставя хубава възможност за хубав графичен интерфейс на информацията която се сервира от HTML.

Той е създаден с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страница. В резултат на това HTML кода се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл, и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл.

CSS-а, всъщност трябва да определи дадена html част и да му направи определен стил. Този принцип се случва с дадените селектори. Селекторите работят по принципа на това да вземат нещо което характеризира този html tag. За тази цел може да се селектира по името на тага, класа или уникалното id.

1. Селект на таг селектор {атрибут: стойност}
2. Основния принцип за класовете е селектор.клас {атрибут: стойност}
3. За ID селектори #id {атрибут: стойност} / или също селектор#id {атрибут: стойност}/

CSS също предоставя възможност за писане на стила вътрешно в HTML-а чрез атрибута style.

2.5. JavaScript

JavaScript е скриптов език който дава възможността да се манипулира с елементите в HTML-а директно в от потребителя. Поддържа обектно-ориентиран и функционален стил на програмиране. Създаден е в Netscape през 1995 г. Най-често се прилага към HTML-а на Интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. Javascript е стандартизиран под името EcmaScript.

JavaScript може да влияе на почти всяка част от браузъра. Браузъра изпълнява JavaScript кода в цикъла на събития т.е. като резултат от действия на потребителя или събития в браузъра (например document.onLoad).

Основни задачи в повечето JavaScript приложения са:

- Зареждане на данни чрез AJAX.
- Ефекти с изображения и HTML елементи: скриване/показване, пренареждане, влючене, слайд шоу, анимация и много други.
- Управление на прозорци и рамки.
- Разпознаване на възможностите на браузъра.
- Използване на камерата и микрофона.
- Създаване на 3D графики WebGL.
- По-добър и гъвкав потребителски интерфейс

Какво не може да се прави с помощта на JavaScript:

- Не може да се записва информация на потребителския компютър или отдалечения сървър.
- Не може да се запазва информация директно в отдалечена база данни.

- Не може да се стартират локални приложения.

В нашия проект също така се ползва jQuery което е библиотека на JavaScript и позволява достъпването до елементите от html-а много по-лесно и едноставно.

2.6. Thymeleaf

Thymeleaf е език базиран на Java XML/XHTML/HTML5 който позволява да свързваме бек-енда и фронт-енда с това че може да интерпретираме неща сервириани от зад в нашия view. Той е модерен server-side за Java уеб и самостоятелна среда.

Thymeleaf е отворен за ползване и лицензиран от Apache 2.0. Създаден е да замести JavaServer Pages(JSP) и имплементира основните концепти на темплейтите. Работи перфектно и е най-вече приложен с Spring framework-а.

2.6. Apache Tomcat

Apache Tomcat е уеб сървър за Java уеб приложения. Той е безплатен и с отворен код и имплементира спецификациите на Sun Microsystems за JSP и Servlets. Той представлява контейнер за Java уеб приложения, който осигурява среда, в която да се изпълняват те. Той е изцяло написан на Java и е примерна имплементация на горепосочените спецификации.

Apache Tomcat има висока надеждност (High Availability) и може да продължава да обслужва заявки докато се извършва обновление на версията му. Това е важна способност, която се ползва в среди с високо натоварване.

Сървърът поддържа още и работа в клъстър. Това прави възможно разпределение на товара (load balancing) на различни инстанции на сървъра с цел да се повиши броят на обслужваните потребители.

Освен Java уеб приложения Apache Tomcat може да сервира и обикновени статични файлове.

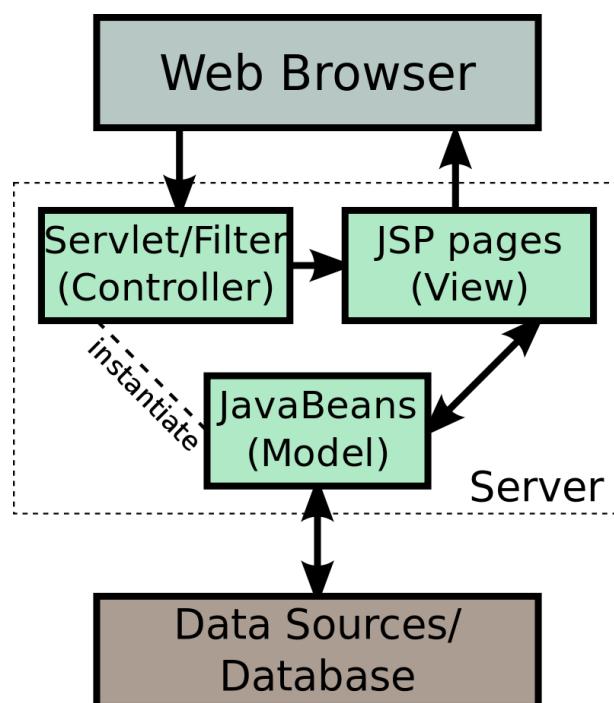
2.7. Извод

В основата на моето уеб приложение стои Spring framework-а, който е базиран на програмния език Java. Избрах точно тази технология защото смяtam, че е доста популярна и широко ползвана. Spring като един от най-добрите фреймурци на Java, дава ми възможността бързо и лесно да имплементирам MVC (Model-View-Controller)

шаблона. Позволява ми да работя с най-новите технологии и да се съобразя с желанията на клиентите.

Нуждата от съхраняване на различен тип данни за дълъг период от време доведе до избора на системата за управление на бази данни MySQL. Тази СУБД е широко разпространена, лесна за употреба и широко използвана за интернет и мобилни приложения. Необходимостта за връзката между уеб приложението и СУБД, както и независимите заявки, които трябва да се правят бързо и сигурно доведе до използването Spring технологията която чрез нейната вътрешна имплементация наречена Spring Data, позволява извършването на множество асинхронни заявки към сървър предоставящ определени услуги.

За изкарване на имплементацията на потребителския интерфейс ползвах основните технологии които за база на всяко едно уеб приложение. HTML5, CSS3 и JavaScript позволяват да сложа съвременен изглед на цялата информация, която се рендерира. Позволява ми да сервирам на хората един хубав потребителски интерфейс, с който лесно може да достъпят и реализират техните желания. Основната глобална архитектура гледана от горе е показана на фиг. 2.1. Единствената разлика там е че вместо JavaServer Pages (JSP) в моя проект ползвам Thymeleaf който ми върши същата функция за изкарване на необходимата информация получена от бек-енда.



2.1 – Глобална архитектура

[https://en.wikipedia.org/wiki/JavaServer_Pages#/media/File:JSP_Model_2.svg]

3. Проектиране на приложението

За проектиране и изграждане на съответната архитектура на уеб приложението, базата данни и връзката между тях е необходимо преди това да определим съответно функционалните и нефункционални изисквания, съобразени с предварителния анализ на други софтуерни решения от същия тип. Описаните по-долу функционални изисквания определят съответно потребителските възможности които трябва да бъдат реализирани за изпълняване на всички нужни функционалности. Следващата подточка, описва съответните функционални и нефункционални изисквания и описаните изисквания от към потребителската част.

3.1. Функционални и нефункционални изисквания

3.1.1. Функционални изисквания

1. Базата данни да е нормализирана
2. Достъп до вече обявените информации за не автентириани потребители (гост)
3. Алгоритъм за топ 3 граждани които са направили добри дела за текущия месец
4. Връзка към API което дава информация корона вируса
5. Превръщане на информацията към обекти с които може да манипулираме
6. Изкарване на постовете низходящо според датата на обява
7. Възможност за търсене по ключова дума в секцията новини
8. Възможност за шерване на отделните постове на Facebook и Twitter
9. Изискване на автентикация от страна на потребителя при използване на основните функционалности на приложението.
10. Възможност за регистрация на нов потребител в самото приложение.
11. Валидация и верификация на полетата за регистрация(предпазване от празни полета и въвеждане на минимум на символи за потребителско име, парола).
12. Проверка за съществуване на потребителското име което иска да се регистрира. Ако го има да се върне фийдбек с информация за избиране на ново име
13. Криптиране на паролите
14. Възможност за помнене на логнатия акаунт за определено време

15. Възможност за логване с акаунти от Facebook и Google
16. Разделяне на акаунтите на 4 типа: обикновени потребители (user), модератори (moderator), админи (admin), създател на приложението (root-admin или owner)
17. Да е реализиран шаблона на йерархия на ролите - почвайки от guest, user, moderator, admin па се до root-admin всички да притежават правата на тези под тях плюс допълнителни
18. Успешна авторизация с цял достъп на потребителите с разни роли само и само до това за което имат права.
19. При регистриране на нов акаунт да бъде възложена ролята – обикновен потребител
20. При логване с помощ от други апликации да бъде възложена ролята – обикновен потребител
21. Потребителите да имат възможността да постират в разните секции
22. Възможност за модераторите и ролите нагоре да приемат или отхвърлят request-натите постове
23. Да се изкарват логовете от действията на всички потребители с информация какво са направили, къде и кога
24. Модераторите и ролите нагоре да може да достъпват логовете
25. Модераторите и ролите нагоре да може да чистят логовете
26. Админите и owner-а допълнително да могат да трият вече обявени постове
27. Админите и owner-а да могат да достъпват организацията на ролите
28. Админите и owner-а да могат да трият потребителите
29. Админите и owner-а да могат да променят роли на останалите потребители
30. Админите и owner-а да нямат права да променят собствената роля за да се предотвратят нежелани действия
31. Възможност за изход от приложението.

3.1.2. Нефункционални изисквания

Уеб приложение

1. Приложението да бъде разработено за всички видове уеб браузъри

1. Да има хубав потребителски интерфейс
2. Интерфейса да е изчистен и лесно достъпван
3. Удобна навигация между отделните страници
4. При всяко логване да се намираме на началната страница
5. Снимките да са качествени
6. Информацията да е обективна и качествена
7. Зареждането помежду страните да бъде бързо
8. Ясен feedback към потребителите които споделят постове за успешен/неуспешен опит
9. Ясно приказване за това кога си логнат и кога не
10. Отговор при грешна парола/потребителско име
11. Ясен приказ къде точно грешим при логване/регистрация
12. Помощ как да си поправим нещата при регистрация
13. Ясна информация относно корона вируса
14. Истинските пароли да не са видими от никого, даже и от базата данни
15. Ясен приказ за постовете

База данни

1. Приложението да използва релационна база от данни за съхраняване на информацията
2. Приложението да ползва JPA Repository за връзка помежду данните и моделите в самото уеб приложение

3.1.3 Изисквания от към потребителската част

1. Ако съм потребител на приложението искам да мога да се регистрирам използвайки потребителско име, еmail и парола.
2. Ако съм потребител трябва да ми се даде ясен отговор(фиидбек) при въвеждане на не-достатъчни данни в полета за регистрация

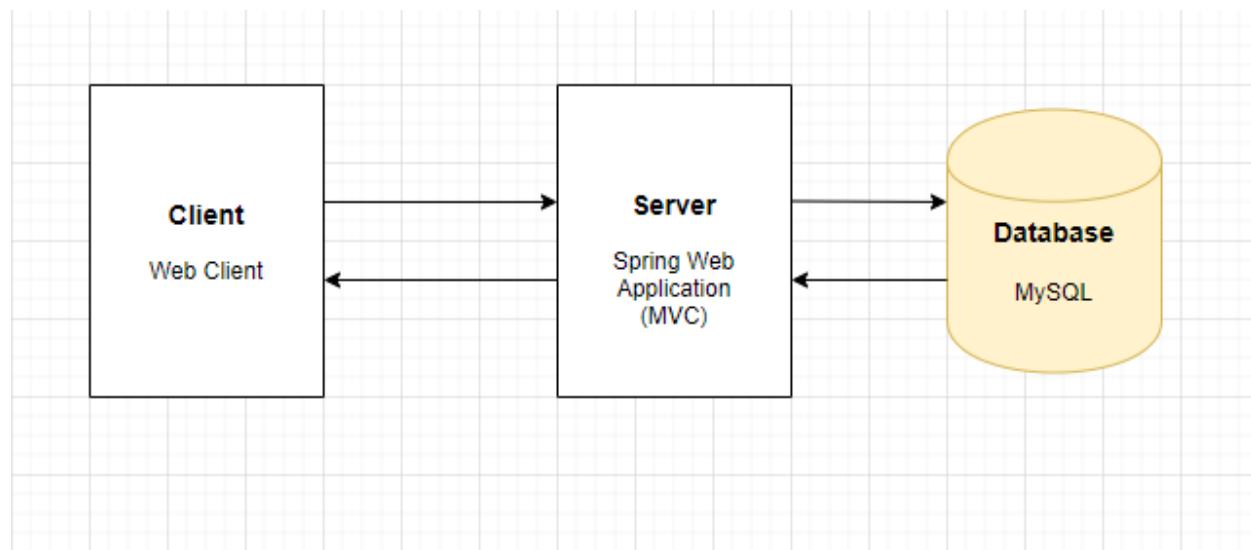
3. При регистриране да се проверява дали моето име е уникално (не съществувало до сега)
4. При съществуване на потребителското име да ми се изпише на екрана че съществува и да не ми се губи информацията от другите полета. Само да си въведа ново потребителско име с което няма да губа време за да въвеждам отново и отново потребителско име, парола и т.н.
5. При успешно регистриране да ми се даде знак че вече съществувам като потребител
6. При успешно регистриране да ме прехвърли към страницата за логване за да мога веднага да влезна в сайта
7. Да имам шанса да гледам новините, евентите, събитията, добрите дела, даренията, талантите и всичко което е особено за сайта на Неготино, без да се логирам
8. Да имам шанса да се логна бързо и едноставно без регистриране с помощ от Фейсбук и Гугъл
9. Ако имам мой акаунт в приложението да мога свободно да се логвам с потребителско име и парола
9. При логване с моя акаунт в приложението да имам пълна информация ако въведа неверни данни (потребителско име и парола)
10. Да имам възможност приложението да помни че съм логнат, за да не писвам отново потребителското име и парола за вход всеки път
11. При успешно логване да ме хвърли към началната страница на приложението
12. Като потребител с определена роля да имам правата да виждам всички части от приложението които са заделени за моята роля
13. Като обикновен потребител да имам предоставен удобен интерфейс и лесна навигация низ страниците на приложението
14. Като обикновен потребител да имам правата за създаване на публикация във всички възможни секции на приложението
15. Като потребител да имам възможността да добавям снимки на моите публикации
16. Да имам възможността при описание на моята публикация да виждам как приблизително тя ще изглежда
17. При въвеждане на не-достатъчни данни приложението да ми изписва къде точно съм згрешил и какво трябва да поправя
18. При успешна публикация да ми е потвърдено на екрана

19. Като потребител с заделена роля модератор да имам всички възможности като обикновения потребител
20. Като допълнителна функционалност с роля модератор да мога да приемам или отказвам постови
21. Като допълнителна функционалност с роля модератор да мога да гледам историите на всички потребители в приложението и съответно да ги чистя при претрупване
22. Като потребител с заделена роля админ да имам всичките права като модератора
23. Като допълнителна функционалност с роля админ да мога да тряя вече обявени публикации от всички потребители
24. Като допълнителна функционалност с роля админ да мога да гледам организацията на сайта по потребители
25. Като допълнителна функционалност с роля админ да имам възможност да тряя напълно потребители от приложението
26. Като допълнителна функционалност с роля админ да мога да смяна роля на определен потребител
27. Като потребител с заделена роля главен админ (ROOT-ADMIN или owner) да имам всичките права като админа
28. Като потребител с заделена роля главен админ да съм уникален с това че моя профил или роля да не може да бъде изтриена/променена
29. При излизане от profila mi приложението да ме слага на началната страница където мога да видя ресурсите предназначени за гостите.

3.2 Архитектура

3.2.1 Цялостна архитектура

Цялостната архитектура представлява общ преглед към нещата отгоре. За това как работи системата изцяло, от достъпването на клиента през уеб браузера, неговото взаимодействие с помощ на основните заявки GET, POST, PUT, PATCH, DELETE от които за целите на това уеб приложение ще се ползват основно GET и POST. След това, минаването на информацията през сървъра където се случват базовите неща, с операции за манипулиране на данните, което при нас ще е в имплементацията която ни нуди Spring framework-а и MVC (Model-View-Controller), па се до записване на тези данни в корена който, лежи най-отдолу базата от данни – MySQL. Глобалната архитектура е показана на фиг. 3.1.



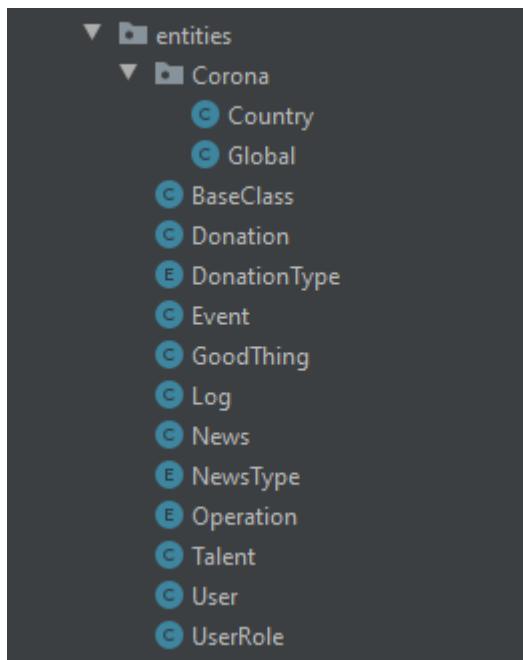
3.1 – Глобална архитектура

3.2.2 Архитектура според MVC шаблона

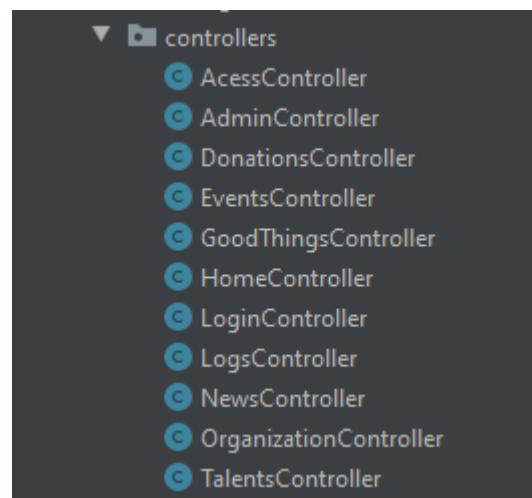
Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение. За пръв път този шаблон за дизайн е използван в програмния език Smalltalk. Подобни шаблони на MVC са MVVM(Model-View-Viewmodel), MVP(Model-View-Presenter), MVT(Model-View-Template).

1. **Model** – това е частта която представя ядрото с данни. Това са обектите които представляват нещо от „реалния свят“. Тези обекти за основните информации кои са директно свързани с таблиците от базата данни чрез използване на класовете от обектно-ориентираното програмиране. Всеки клас си има свои атрибути, методи и конструктори. Модела като една основа на шаблона стои най-отдолу на шаблона MVC и

е междинен слой помежду външната база данни и контролера. Неговите данни се взимат от по-горните слоеве, примерно контролера, които ги обработва и праща на view частта. Също така view частта чрез потребителски интерфейс може да промени тези модели които обратно ще се препратят към контролера който с помощ от своите пот-слоеве ще промени информацията в определените модели и съответно ще се запишат в базата данни. В нашия проект моделите ще се намират в пакета entities. На фигура 3.2 се виждат основните класове които представляват M(Model) от шаблона MVC(Model-View-Controller) и същевременно представляват таблиците в нашата база от данни.



Фиг. 3.2 – Модели



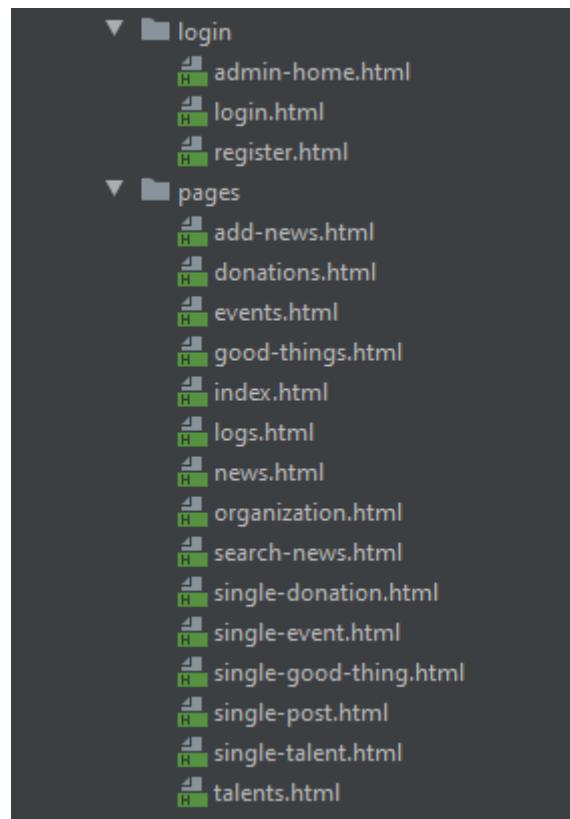
Фиг. 3.3 – Контролери

2. **Controller** – секцията която хендълва поискането на клиентите към определена функционалност на приложението с помощ на GET и POST заявки. При тази интеракция контролера актуализира моделите и не води към някой друг изглед(view). Разбира се в неговата част той си има пот-слоеве(примерно services, repository и т.н) които ще разгледаме

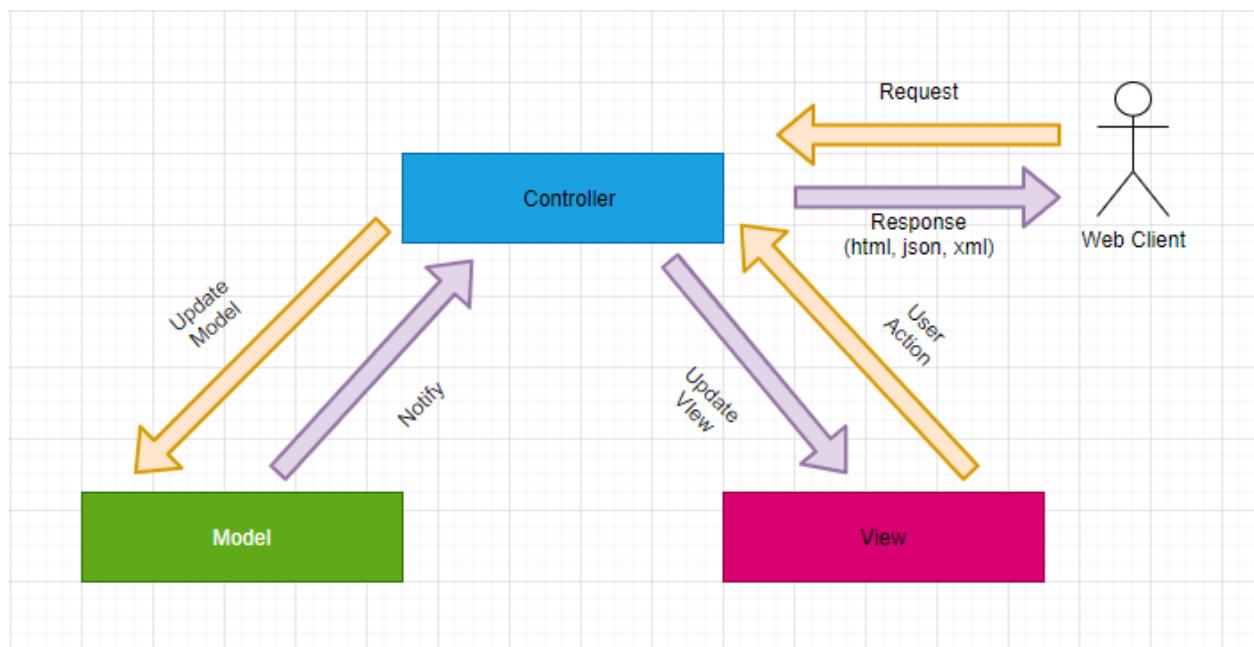
по-надолу.

В нашия проект контролерите ще се намират в пакета controllers (фиг. 3.3). Ще имаме повече контролери, тъй като искаме да реализираме повече страници и функционалности които да са достъпни за потребителя.

3. **View** – това е частта в която трябва да визуализираме данните от модела. Това е последната част от MVC шаблона която сервира на потребителя резултата от информациите които ги поискал. Тези данни пред да стигнат до изгледа са минали от модела който ги е взел от базата данни, през контролера за на край да се визуализират в някой от view-та. Изгледите в нашия проект сарендериирани с помощта на Thymeleaf engine. Тази технология ми дава възможността добре и да използвам един вид Java код в клиентската част(html).



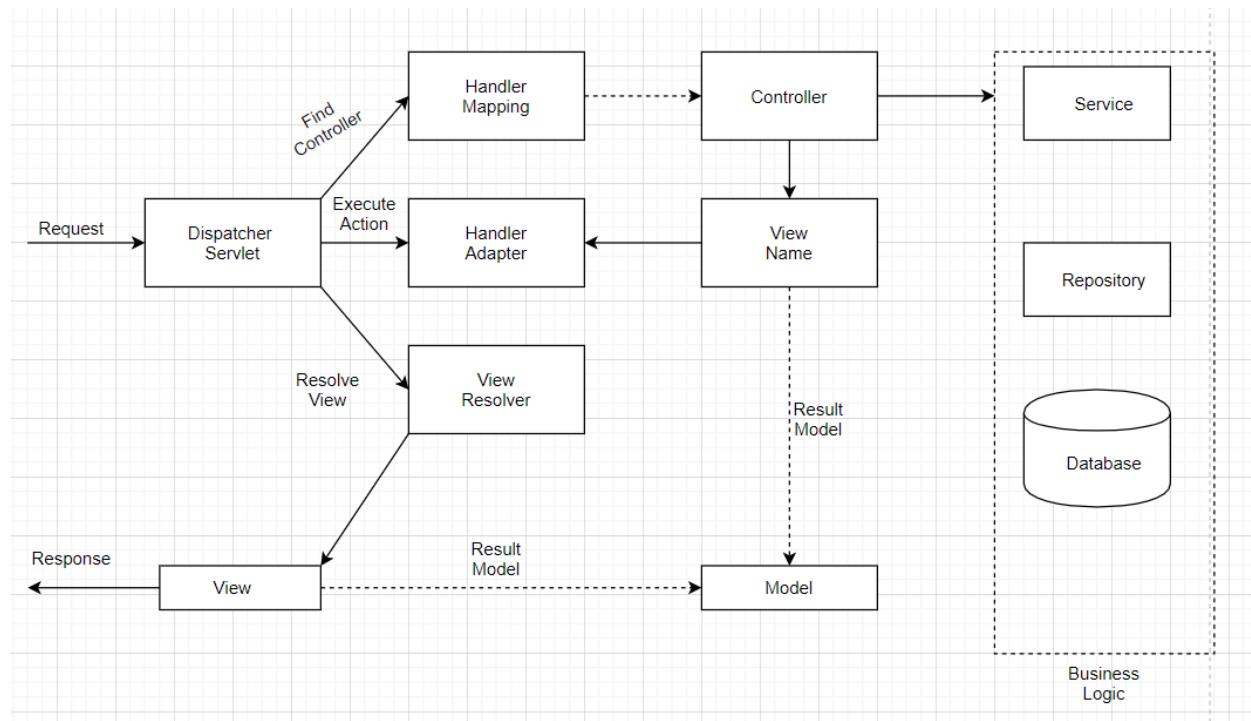
Фиг. 3.4 - Изгледи



Фиг. 3.5 – MVC - Control Flow

3.2.3 Как Spring имплементира MVC

MVC framework-а в Spring работи върху т.н Dispatcher сервлета който се деплои в Каталина или Томкат. Когато дойде един request той първо отива до диспачер сервлета. Диспачер сервлета е конфигуриран така, че да хваща абсолютно всички request-и. В момента в който намери request той търси контролер. Когато намери контролер търсим дали този контролер има мапинга който като цяло ни е метода който ще се изпълни или т.н контролер action. В момента в който го намерим взимаме контролера, той съответно стартира услуги, репозиторита и всичко, което му трябва (всичките му депенденси) и вътре в него изпълнява бизнес логиката. На края след извършване на логиката имаме някакво view, което ни връща хендлер. Този хендлер съответно може в себе си да извърши някакви действия (примерно може да е логин форма, която праща POST request). След извършване на действията трябва всичко да се събере на едно и да се върне като хендлер на диспачер сервлета. В момента, в който тези неща се връщат върху диспачер сервлета изпълнява action-а, който е пратил потребителю. Изпълнява го върху хендлера, прави някакво view или някакъв resolver, който ни дава view-to. Това нещо отива до view-to, нещата се комбинират в един модел (който контролера го пълни) и дава възможността за тези неща да могат да се рендерират в view-to. На края всичките тези неща, комбинирани в едно цяло, се изпращат като response.

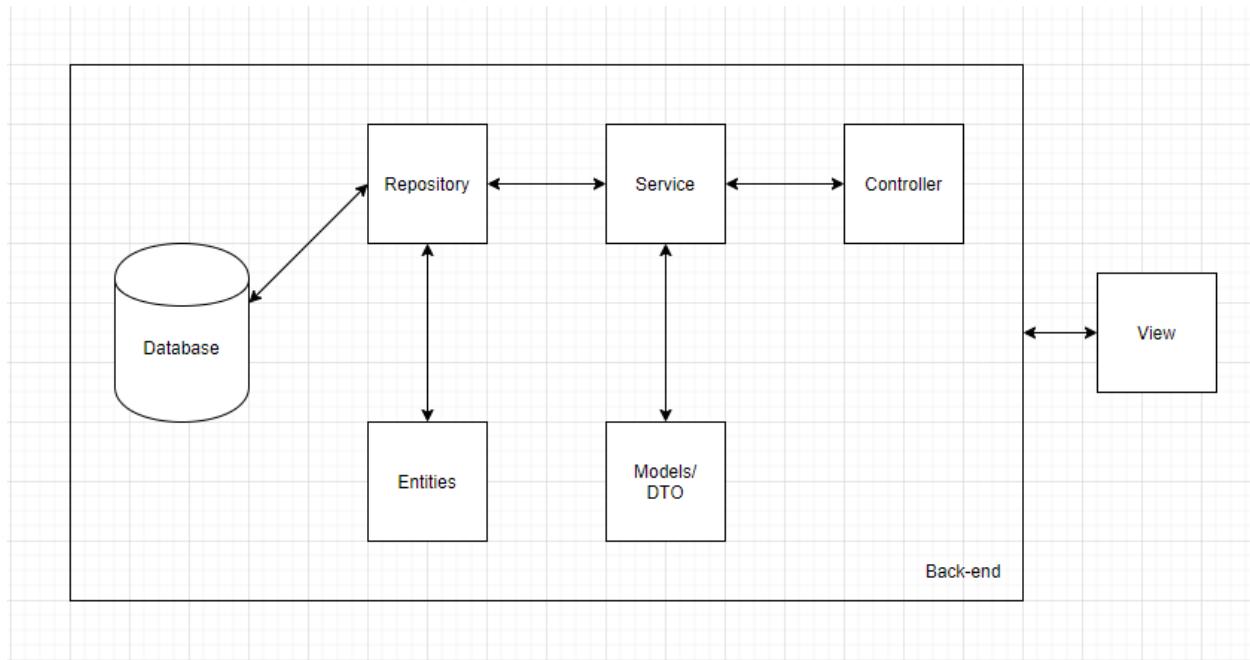


Фиг. 3.6 – Имплементация на MVC в Spring

3.2.4 Архитектура на уеб приложението

Архитектурата на уеб приложението, гледана от долу нагоре се състои от следващите компоненти:

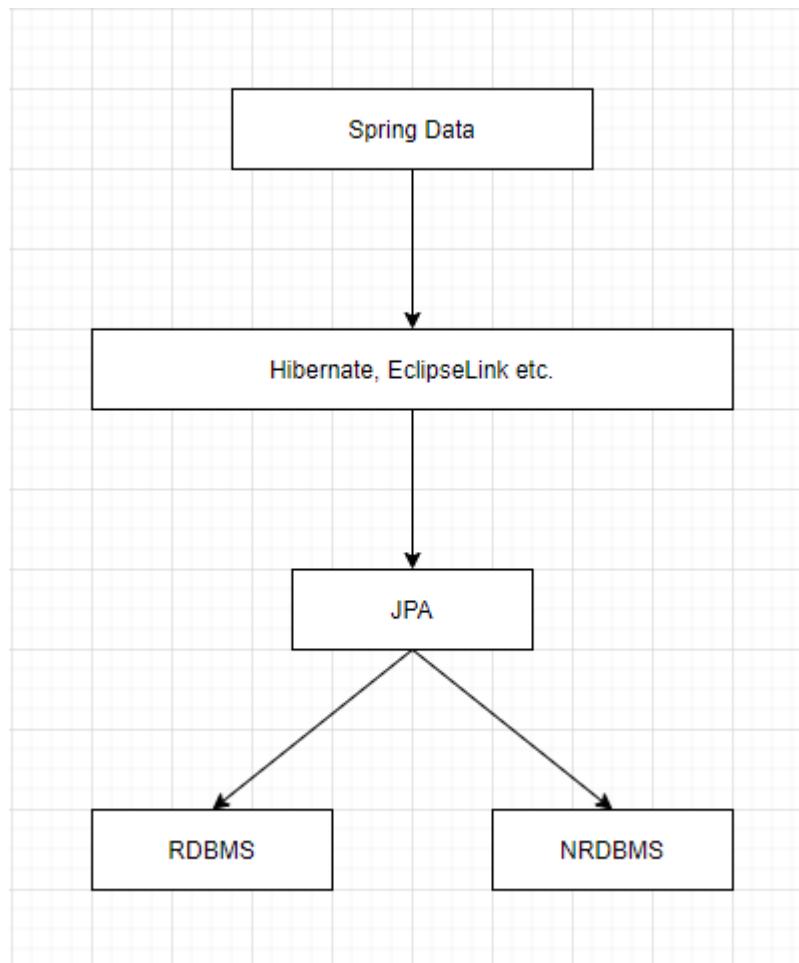
- База от данни – място където ще се съхраняват всичките необходими данни.
- Repository – компонент който ни служи за директна връзка с данните в базата от данни. Тука се очаква да се случва CRUD логиката – добавяне, ъпдейтване, триене, селектиране и всичките спецификации на query-тата които базата данни ни овъзможава. За по-лесно имплементиране на тези задачки, Spring framework-а предоставя свой компонент наречен Spring Data с който може много по-едноставно да се реализират query-тата.
- Entities – класове които представляват реалните обекти с които ще работим и които ще са запазени директно в базата данни във вида на таблици
- Service – слой който не е задължителен, обаче е добре да съществува за по-ясна и по-качествена архитектура. Той е инжектва Repository-то и в него трябва да се имплементира цялата бизнес логика на приложението. Описване на почти всичките методи на приложение: намиране на топ 3 автори на добри дела за текущия месец, регистрация на потребител, поставане на публична информация и т.н. Сервизите трябва да са инжектнати в контролери през които да им се извикват разните методи. Всеки метод има обръщане към базата от данни и съответните манипулации с данните в нея чрез предишния компонент Repository.
- Models – Компонент който представлява модела (M) от MVC шаблона. Те са същите класове от пакета entities, обаче са създадени за да се ползват в сервисите с цял да не се работи директно върху ентитата които реално са таблиците в базата данни
- Controller – Контролерите са класовете които се занимават с URL request-ите. Техните методи са наконфигурирани с определен мапинг. Съответно, те чакат някакво поискане на тоз мапинг от страна на потребителя, за да извикат съответния метод. Във тях се очаква да бъде включен отреден обект от сервисите чрез който да се извикват методи които са нужни за съответния request. Във тях не трябва да бъде реализирана никаква бизнес логика. Там трябва да има само валидация на данните. Когато свършат с обработката на данни контролерите трябва да връщат определено view което може да бъде html страница или определен отговор(response).



Фиг. 3.7 – Архитектура на уеб приложението

3.2.5 Архитектура на Spring Data

Spring Data е компонент на Spring framework-а който добавя един допълнителен слой над JPA провайдера. Той надгражда JPA, Mongo и друг. Spring Data намира приложение в компонента Repository. С негова помощ с много малко писане на код получаваме генериран код. Това помага, с това че много лесно може да манипулираме с данните, които са налични в базата от данни.

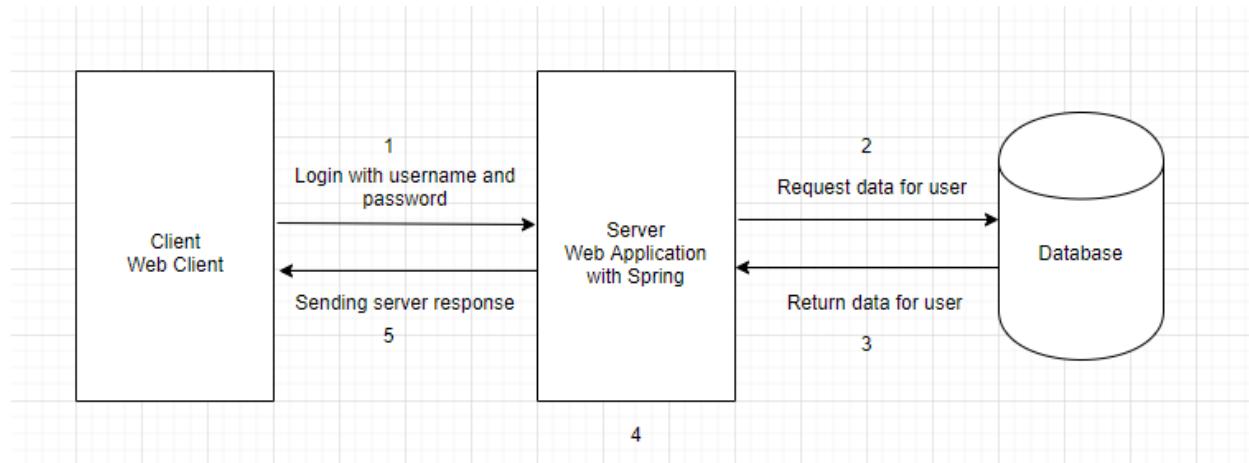


Фиг. 3.7 – Spring Data – абстракция

3.2.6 Архитектура и механизъм за автентикация и авторизация

Процесът на проверката дали потребителското име съвпада с паролата (автентикация) се състои в изпращане на потребителско име и парола от уеб сайта към сървъра, валидиране на данните на сървъра с проверката на съществуващите потребители в базата данни и при успешна валидация се преминава към началната страница на приложението. В противен случай се изписва уведомление за грешно потребителско име или парола. (Фиг. 3.8).

Тези неща са направени с помощта на Spring Security компонента на Spring framework-a който представя една добре структурирана имплементация която изцяло се занимава с потребителите и тяхната сигурност.

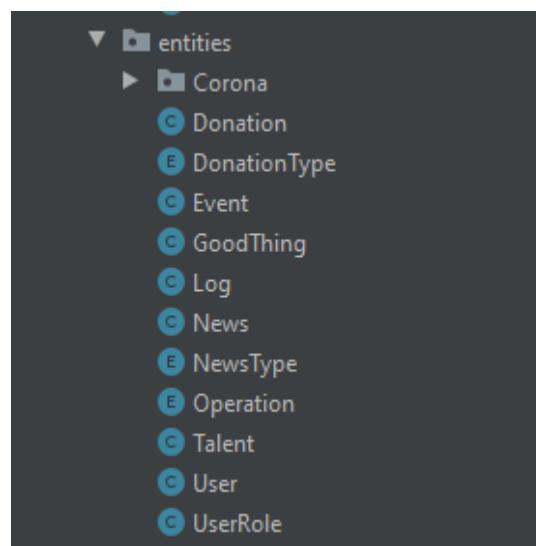


Фиг. 3.8 – Автентикация и Авторизация на потребител

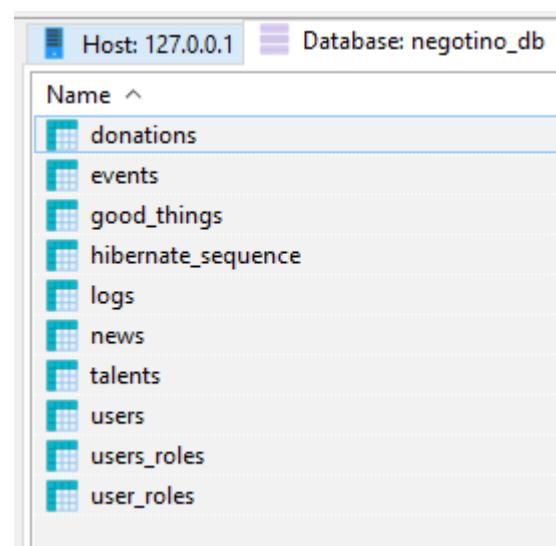
Авторизацията на потребителите ще става по подобен принцип където за логнатия акаунт ще се провери каква роля му е зададена в сървъра с базата от данни.

3.3 Проектиране на базата данни

Името на нашата база от данни е negotino_db. В нея ще се съдържат повече таблици, които са ни необходими. Всичките таблици са ни пряко свързани с класовете който имат конфигурация @Entity(M от MVC) в пакета entities. Същите тези класове представляват нашите таблици в базата данни negotino_db.



Фиг. 3.9 – Entities



Фиг. 3.10 – Таблици в базата данни

3.3.1 Таблица users

Таблица която съдържа всичките потребители и информацията необходима за един потребител във уеб или друго подобно приложение.

Име на колоната	Тип	Описание
id(PRIMARY_KEY)	varchar	UUID на потребителите
username	varchar	Потребителско име
password	varchar	Парола на потребителя
email	varchar	Еmail на потребителя
enabled	bit	Дали акаунта е активиран
account_non_expired	bit	Дали акаунта не е стар
account_non_locked	bit	Дали акаунта не е локнат
credentials_non_expired	bit	Дали креденшлите не са стари

Колоните от типа на дали е акаунта е активиран, локнат или изминат срок на потребителя са допълнителни проверки върху нашите клиенти. За момента, в нашия проект всичките тези ще са true, което ще ни означава те ще винаги ще са активни, никога няма да бъдат локнати или с изминал срок.

3.3.2 Таблица user_roles

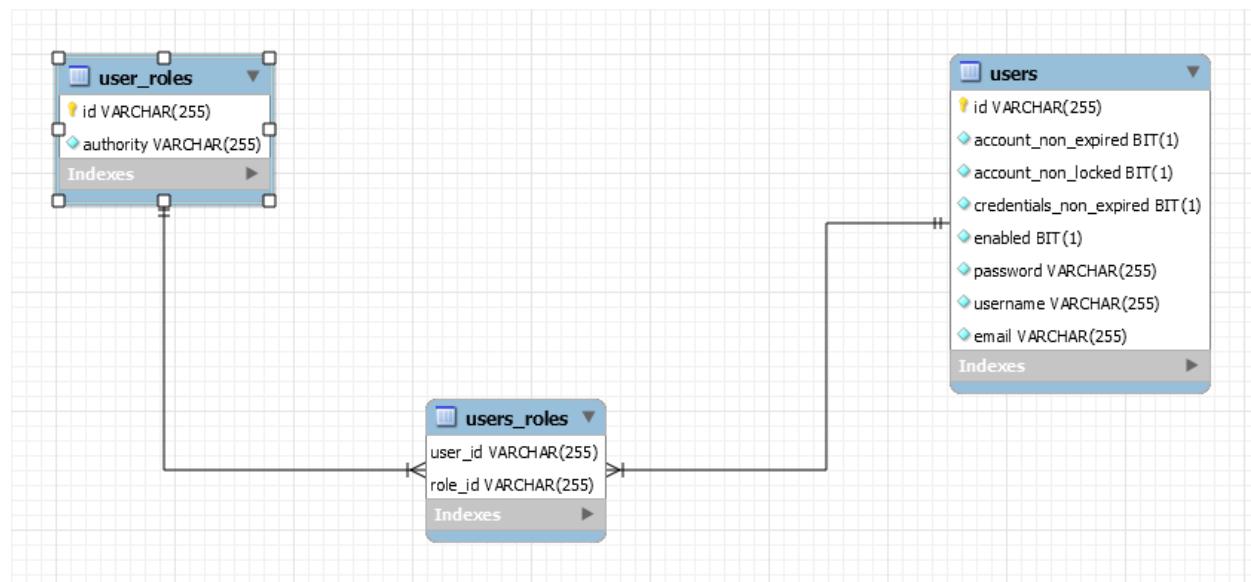
Допълнителна таблица, която ще дефинира различните роли на потребителите. По принцип, тези роли могат да са включват като String(varchar) в таблицата на user-ите. Обаче, тази логика е създадена за по-добро проектиране на базата данни. С това ние се предпазваме и намаляваме си проблемите в бъдещето. Един пример за това е ако след известно време искаме да променим всички роли от тип модератор към роля админ или някоя друга, да не ходим във всички редове от данни за потребител и навсякъде променяме колоната за ролята. При голям брой на потребител, това ще ни донесе големи проблеми и губене на време.

За това, прилагаме нова таблица която да описва отделните роли и с това имплементираме част от функционалното изискване което гласи – **нормализация на базата данни**.

Име на колона	Тип	Описание
id(PRIMARY_KEY)	varchar	UUID на ролята
authority	varchar	Име на ролята

3.3.3 Таблица users_roles

На края трябва по някакъв начин да свържем таблицата с потребителите(users) и таблицата с ролите(user_roles). Тази връзка ще бъде ManyToMany защото един потребител може да има повече роли, също така и една роля може да бъде присвоена на повече потребители.



Фиг. 3.11 – Many-To-Many Relationship

3.3.4 Таблица news

Тази таблица ще съдържа цялата информация която се отнася на секцията новини

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на новината
title	varchar	Заглавие на новината
text	varchar	Текст на новината
type	int	Категория на новината
date	datetime	Дата на обява
author	varchar	Автор на новината
approved	bit	Дали новината е приета или не. Решения на по-големи роли от обикновения потребител
img_name	varchar	Име на снимката
img_type	varchar	Тип на снимката
img_data	longblob	Запазване на снимката като стринг

Тук id-то, което представлява уникалния номер на таблицата е с тип integer. Идеята тук е по-лесно сортиране и тъпдейтване на данните в таблицата защото всяко ново id ще се инкрементира за 1. Така мога лесно да знам коя е най-ново обявена новина, коя е последна, колко нови обяви се случили, да ги сортирам и подобни манипулатии. Във всичките секции е приложена тази логика с задаване на първичния ключ като интегер.

Заглавието, текста, категорията на новината ще са въведени от самия потребител. Като тип на новината съществува енумерация с възможни категории, които потребителя ще може да избере.

Автора на новината се взима от потребителското име на логнатия акаунт, който въсъщност request-ва новината.

Датата на обява се взима от точния ден, месец, година и час в момента на поставане на публикацията

Основната логика, която предлага тази платформа за тази и всички други секции е да се манипулира с колоната approved, която въсъщност е boolean(bit) променлива. При request-ване на обява от страна на потребителите default стойността на approved е false(0). Тази стойност ще се променя от модераторите или админите. Те ще имат правата на допускане на новината в списъка на секцията чрез промяна на approved променливата на true(1). Логиката ще бъде по-подробно обяснена в следващата част на тази документация за дипломна работа – Реализация.

Img_data е мястото където ще съхраняваме всичките снимки които потребителите ще качват. Този тип е от тип longblob който ще съхранява големи стрингове

Колоните img_type и img_name са излишни и служат ни за по-подробна информация на снимката която е качена от потребителя.

3.3.5 Таблица events

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на събитието
name	varchar	Име на събитието
text	varchar	Текст на събитието
location	varchar	Място на събитието
date	datetime	Време на събитието
organizer	varchar	Организатор на събитието
contact	varchar	Контакт относно събитието
approved	bit	Дали събитието е публикувано(прието) или не.
img_name	varchar	Име на снимката
img_type	varchar	Тип на снимката
img_data	longblob	Запазване на снимката като стринг

Основно таблицата за събития и всичките следващи ще имплементират логиката на таблицата news която току-що обяснихме.

Тук, основната разлика е че има допълнително поле за локация на събитието, организатор и контакта на организаторът или определено лице което отговаря за събитието. Типа на contact е String(varchar) с идеята автора на публикацията да може да въвежда както номера, така и имена на това лице. За разлика от новините датата на събитието няма да бъде въвеждана автоматична в зависимост от момента на обява, ами ще бъде ръчно въвеждана от страна на потребителя който ще има възможността да сподели кога ще се случва самото събитие.

3.3.6 Таблица good-things

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на доброто дело
title	varchar	Заглавие на доброто дело
text	varchar	Текст на доброто дело
author	varchar	Автор на доброто дело
approved	bit	Дали доброто дело е публикувано(прието) или не.
img_name	varchar	Име на снимката
img_type	varchar	Тип на снимката
img_data	longblob	Запазване на снимката като стринг

3.3.7 Таблица donations

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на дарението
title	varchar	Заглавие на дарението
text	varchar	Текст на дарението
type	int	Тип на дарението
approved	bit	Дали дарението е публикувано(прието) или не.
contact	varchar	Контакт към дарението
img_name	varchar	Име на снимката
img_type	varchar	Тип на снимката
img_data	longblob	Запазване на снимката като стринг

Колоната type на дарението може да бъде от два типа: Търсене на дарение и Информиране че нещо искаме да дарим. Това се изпълнява с помощ на енумерация.

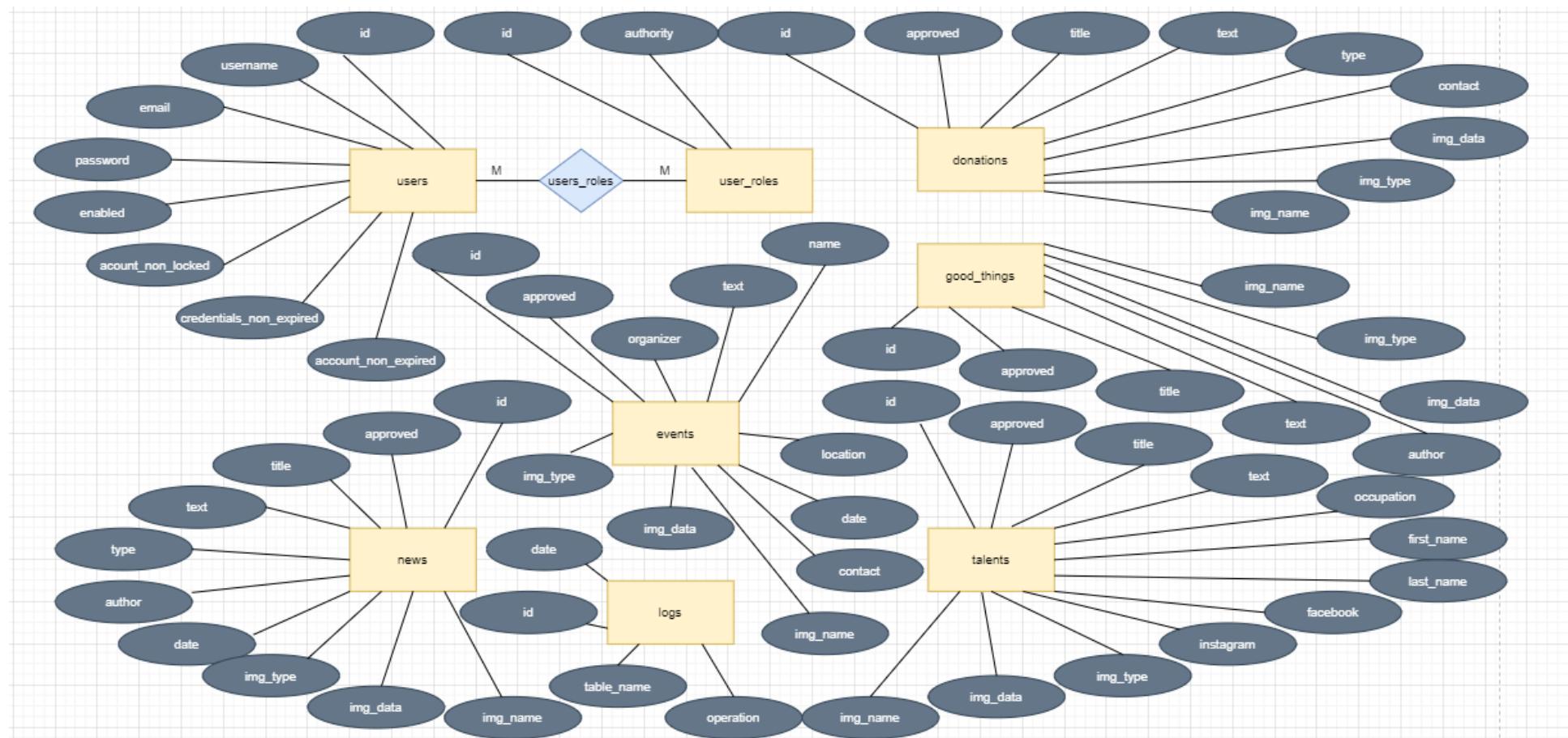
3.3.8 Таблица talents

Таблицата таланти ще сдържа основните информации около обявата за талант.

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на таланта
title	varchar	Заглавие на таланта
first_name	varchar	Име на таланта
last_name	varchar	Фамилия на таланта
text	varchar	Текст на таланта
occupation	varchar	Дейност на таланта
facebook	varchar	Facebook контакт
instagram	varchar	Instagram контакт
approved	bit	Дали таланта е публикуван(приет) или не.
img_name	varchar	Име на снимката
img_type	varchar	Тип на снимката
img_data	longblob	Запазване на снимката като стринг

Талантите ще имат име и фамилия, facebook и instagram контактикато допълнителни инпути от user-a. Също така има едно поле наречено окупация което съответства на това с какво точно се занимава съответния талант.

3.3.9 ЕР Диаграма на базата от данни (negotino_db)



Фиг. 3.12 – ЕР Диаграма на базата данни

3.4 Работен процес на приложението

Уеб сайта на Неготино представлява уеб приложение. Това е един основен сървър предоставящ основната функционалност необходима за приложението. Както посочихме по-горе в архитектурата, Spring приложението чрез основния Dispatcher Servlet разглежда какво е поискано от клиента и търси съответния контролер. След това контролера включва всички депенденси надолу, изпълнява съответната логика и връща view.

Разните функционалности на уеб приложението може да са достъпни от клиента чрез въвеждане на определените endpoint-и:

3.4.1 Начална страница - Без въвеждане на ендпойнт (/)

Основно сървъра на нашето приложение е конфигуриран да се деплоява на <http://localhost:8800> което е лесно променимо. Основната част тук е какво ще се въведе след него(ендпойнта). Така, началната страница ще ни се намира точно на <http://localhost:8800/>.

3.4.2 Новини - /news

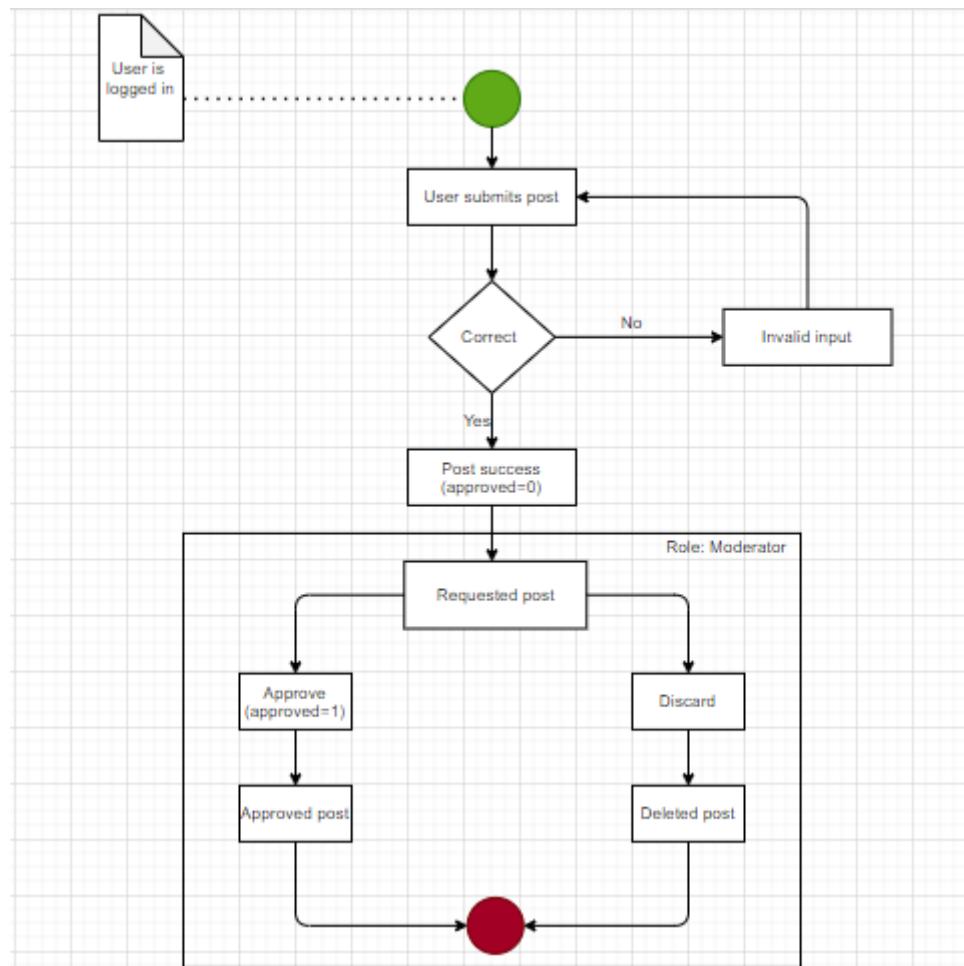
Новините се намират на /news. Това означава че при въвеждане на <http://localhost:8800/news> от страна на клиента ще ни се отвори секцията за новините. В нея ще се актуализират публикациите низходящо от най-нови към най-стари.

В тази страница допълнително съществува функционалността за търсене на новина по ключова дума. При вписване във входните полета на определена дума по коя искаме да търсим ще се реализира GET заявка. Примерно ако потърсим думата „неготино“, GET заявката ще ни бъде от типа <http://localhost:8800/news/search?searchTitle=неготино> което изкарва всички новини имащи в своето заглавие думата „неготино“.

Също така в секцията за новините има възможност за добавяне на новина. Това се случва при кликване на снимката „Додади Новост“ което е линк към нов енд пойнт който ще опишем в подсекцията 3.4.7.

Всички новини, които за поискани от обикновените потребители са с approved = false. Те са рендерираны най-отдолу на страницата /news и само потребителите с роля модератор или нагоре имат достъп до тях. Всяка новина във себе си на изгледа съдържа два бутона – одобряване(approve) и отхвърляне(discard). Също така, за по-лесно извършване на еднотипна функция към всички поискани публикации, под тях съществуват отново два бутона, които са одобряване на всичките(approve all) и отхвърляне на всичките(discard all). В следващите подсекции ще разгледаме какво точно правят те и как са имплементирани. На фигура 3.12 е описан живота на една публикация

в уеб приложението. Тази логика е приложена във всички следващи секции(events,good-things,donations,talents).



Фиг. 3.12 – Post workflow

3.4.3 Одобряване на изпратена новина - /news/approve

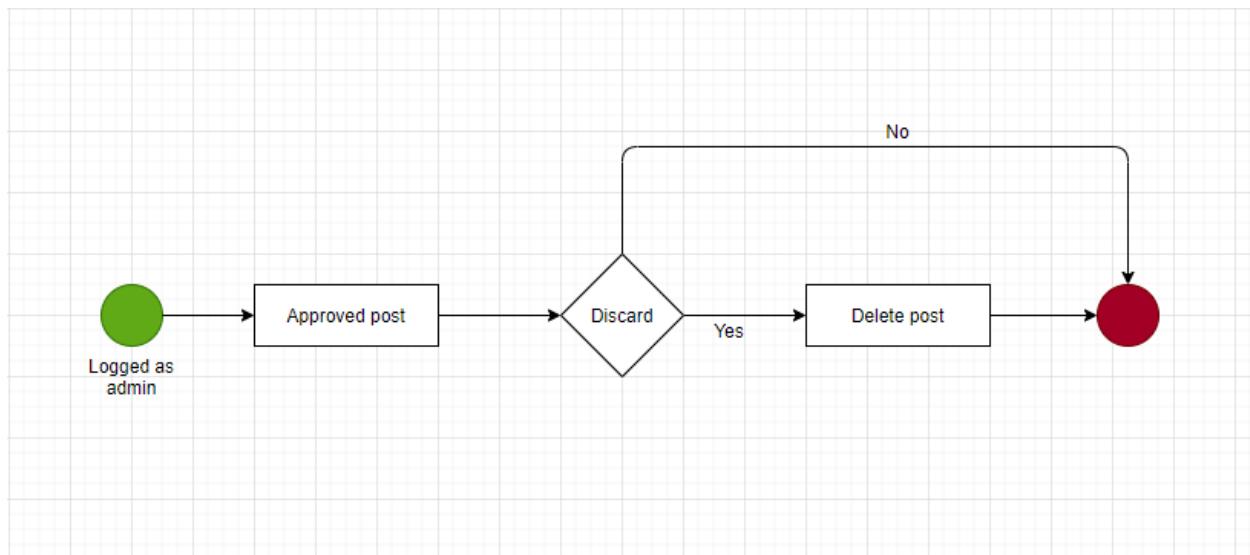
За достъп до тази част имат права само потребителите с роля модератор или нагоре. Този ендпоинт имплементира POST заявка в коя се изпраща уникалност число на новината. Това id се търси в базата данни и когато се намери се извършва съответната функционалност – в случая одобряване на поста. За тази цел, се върши само една единствена промяна – сътвърдане на полето approved на true. Така, при следващото рендериране на данните те ще излезнат като официални новини (одобрени).

3.4.4 Изхвърляне на новина - /news/discard

Отново може да се достъпи само от потребители с минимум роля модератор. POST заявка която изпраща id-то на събитната новина. При намиране на уникалното число,

поста е изтрит изцяло от базата данни защото според workflow-а една публикация в нашия проект при discard-ване на пост той трябва да се махне.

Тук идва време да включим и ролята админ която има допълнителна функционалност – да трябва да една публикувани новини. Така, фигуранта 3.12 (жизната на публикацията) ще се разшири с допълнителната възможност която всичките админи притежават. За тази цел трябва да се поясни че и одобрените постове могат да бъдат изтрити, но само от логнат админ.



Фиг. 3.13 – Допълнителна функционалност при роля админ

3.4.5 Одобряване на всички изпратени новини - /news/approve-all

POST заявка която изпраща всичките уникални числа на request-натите новини. Вътрешно се намират всичките новини от такъв тип и сътва им се полето approved на true.

3.4.6 Отхвърляне на всички изпратени новини - /news/discard-all

Отново POST заявка която изпраща всички уникални числа на поисканите новини. След това чрез изпълняване на бизнес логиката в сервисите, изцяло се троят всичките request-нати новини.

3.4.7 Добавяне на новина - /news/add

Преди да обясняваме функцията на този ендпоинт, трябва да посочим че той е достъпен само за клиентите които притежават акаунт и са логнати в него. В противен случай при достъпване от страна на гост, приложението автоматически ще не прехвърли към логин формата.

/news/add има имплементация на GET и POST заявка:

- GET – при гетване на /news/add с логнат акаунт системата ни дава възможност за да достъпите изгледа на формата за добавяне на новина и всички нейни изисквания.
- POST – тази заявка се извръща само при въведени валидни данни във формата за добавяне на новина с кликане връху бутона „Сподели“. Когато успешно ще въведем всички необходими данни, пост заявката ни изкарва информация за успешна публикация с помощ на AJAX (Asynchronous JavaScript And XML) който ни възмъжава асинхронно да запищем поста в базата данни чрез изпращане на съответните информации за него, при това без да се презарежда страницата. Така след успешна публикация ще си останем на същата страница която е /news/add където ще имаме шанса за да въведем нова публикация или да навигираме лесно към последните новини или някоя друга секция.

3.4.8 Събития - /events

При достъпване на този ендпоинт на потребителския интерфейс излизат рендерирани всички събития сортирани низходящо. Тук добавянето на нов евент отново е възможен само при логнат акаунт със всяка възможна роля, обаче той е имплементиран на същата страница където се намират вече публикуваните събития (/events).

Така, /events имплементира GET и POST заявка отделно:

- GET - /events изкарва на екрана всичките събития
- POST - /events отново с AJAX заявка вкарва request-ния евент в базата данни и изписва съобщение за успех на потребителя

3.4.9 Одобряване на изпратено събитие - /events/approve

Изпълняване на общата логика за новините. Approve на определеното събитие с POST заявка.

3.4.10 Изхвърляне на събитие - /events/discard

Discard на определения event и триенето му от всякаде с помощ на POST заявка.

3.4.11 Одобряване на всички изпратени новини - /events/approve-all

Прихващане на всичките изпратени събития с POST заявка.

3.4.12 Отхвърляне всички изпратени новини - /events/discard-all

Отхвърляне на всичките изпратени събития с POST заявка.

3.4.13 Добри дела - /good-things

Добрите дела имплементират и GET и POST заявка на ендпоинта /good-things, съответно за показване на всичките публикации и добавяне на добро дело.

3.4.14 Одобряване на изпратено добро дело - /good-things/approve

Прихващане на изпратено добро дело с POST заявка.

3.4.15 Отхвърляне на добро дело - /good-things/discard

Отхвърляне на изпратено добро дело с POST заявка.

3.4.16 Одобряване на всички изпратени добри дело - /good-things/approve-all

Прихващане на всички изпратени добри дела с POST заявка.

3.4.17 Отхвърляне на всички изпратен добри дела - /good-things/discard-all

Отхвърляне на всички изпратени добри дела с POST заявка.

3.4.18 Дарения - /donations

Отново съща логика като добрите дела и събитията, имплементира се и GET и POST заявка върху дадения ендпоинт (/donations).

3.4.19 Одобряване на изпратено дарение - /donations/approve

Прихващане на изпратено дарение с POST заявка.

3.4.20 Отхвърляне на дарение - /donations/discard

Отхвърляне на изпратено дарение с POST заявка.

3.4.21 Одобряване на всички изпратени дарения - /donations/approve-all

Прихващане на всички изпратени дарения с POST заявка.

3.4.22 Отхвърляне на всички изпратен дарения - /donations/discard-all

Отхвърляне на всички изпратени дарения с POST заявка.

3.4.23 Таланти - /talents

Същата логика като в добрите дела и събитията, имплементират се и GET и POST заявки съответно за рендериране на всички таланти и добавяне на определен талант който ще бъде вкаран асинхронно с помощ на AJAX.

3.4.24 Одобряване на изпратен талант - /talents/approve

Прихващане на изпратен талант с POST заявка.

3.4.25 Отхвърляне на талант - /talents/discard

Отхвърляне на изпратен талант с POST заявка.

3.4.26 Одобряване на всички изпратени таланти - /talents/approve-all

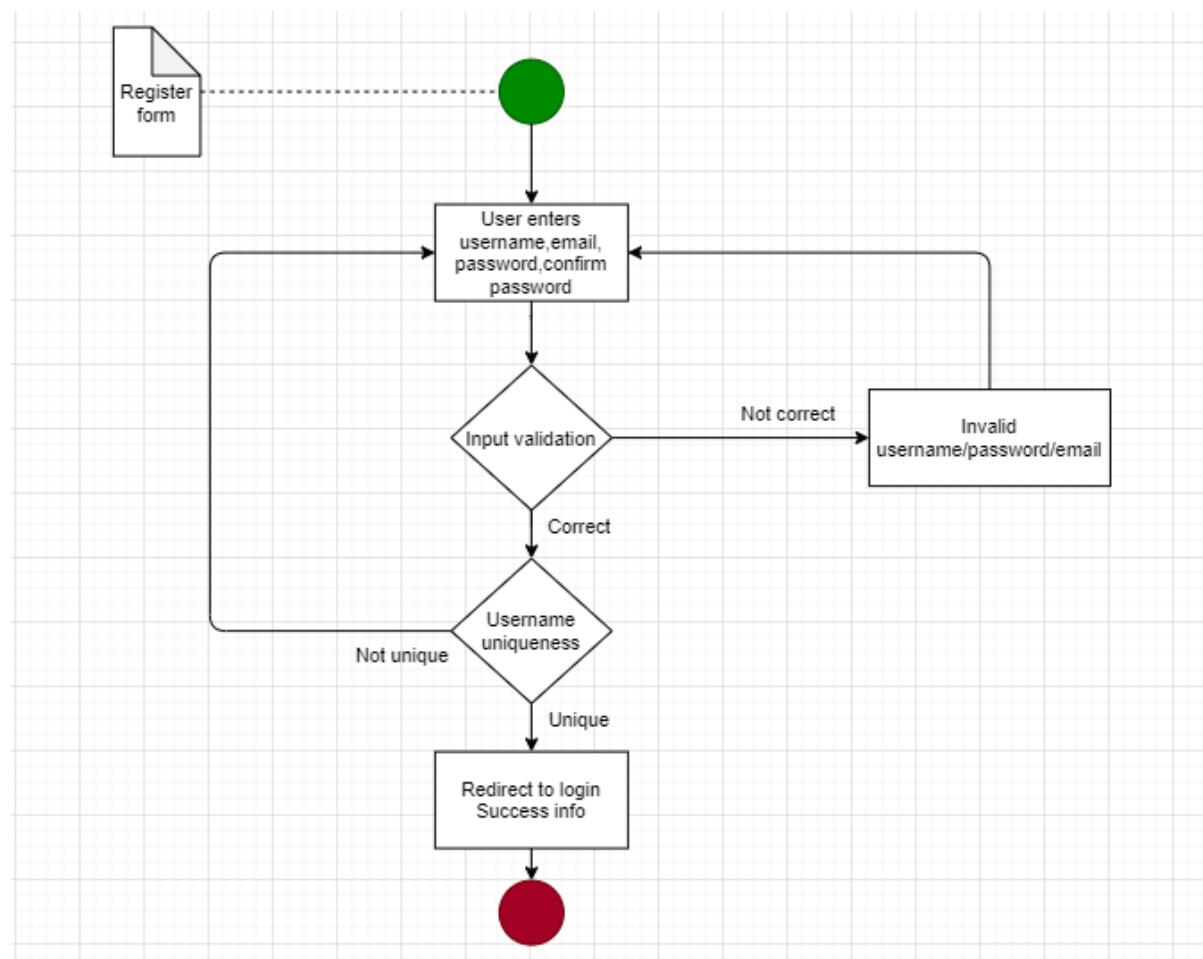
Прихващане на всички изпратени таланти с POST заявка.

3.4.27 Отхвърляне на всички изпратени таланти - /talents/discard-all

Отхвърляне на всички изпратени таланти с POST заявка.

3.4.28 Регистриране на потребител - /register

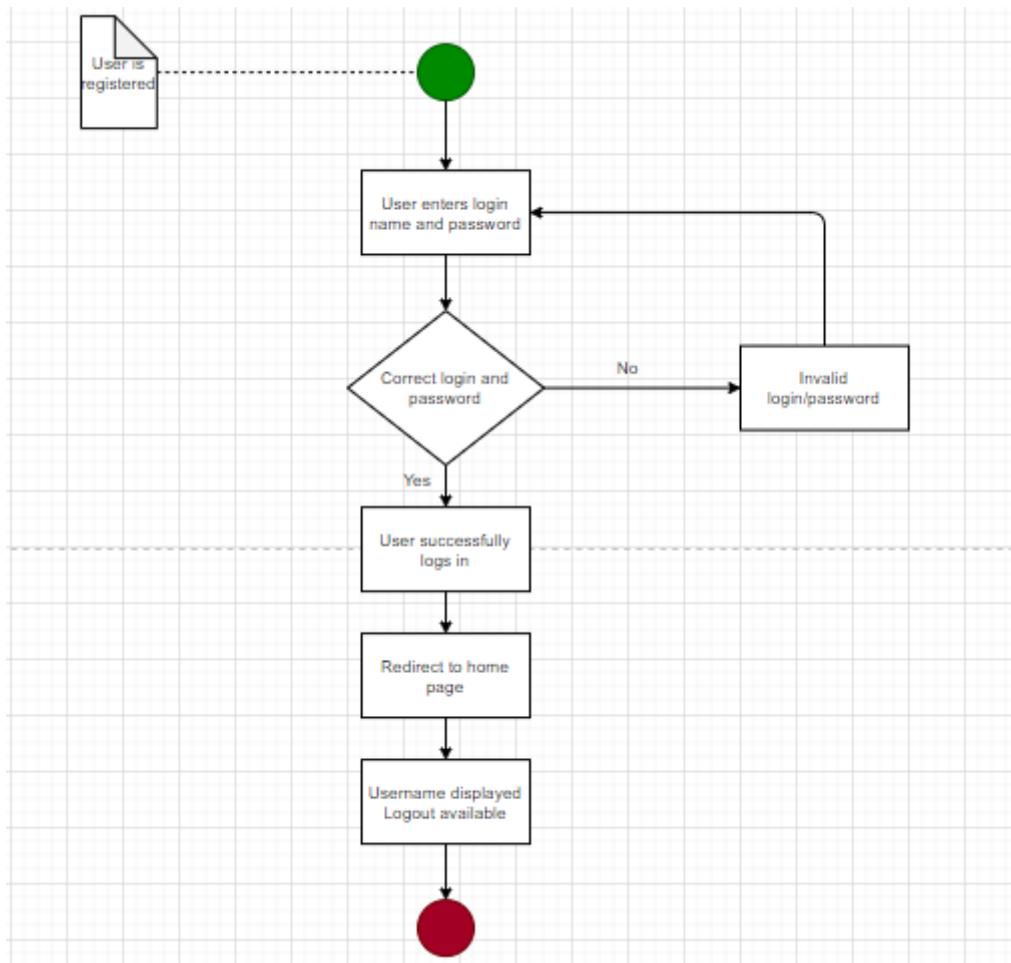
За създаване на нов потребител се връща формата за регистрация при GET заявка. При събmittelване на информацията за потребителя се използва POST заявка където при не успешен опит за регистриране не връща към същата страница и ни посочва къде сме сгрешили. При успешен опит контролера не пренасочва към логин формата, където ни изписва че сме успяли да се регистрираме успешно и нашият акаунт е вече активен. Това означава че вече може да влизаме като потребител в уеб приложението с посочване на съответното потребителско име и парола. Фиг 3.14 описва процеса на регистрирането



Фиг 3.14 – Register workflow

3.4.29 Вход за потребители - /login

За логиране на потребителите се достъпва GET заявка към /login който връща логин формата като view. Опит за вход се имплементира с POST заявка на /login mapping-a. При неуспешен опит, системата редиректва към същата логин форма с изписване на съответната грешка. При успешен вход с креденъшлите уеб приложението не прехвърля към началната страница и изписва потребителското име в горния десен ъгъл където е и възможността за logout. На фигура 3.15 е представен процеса при логването.



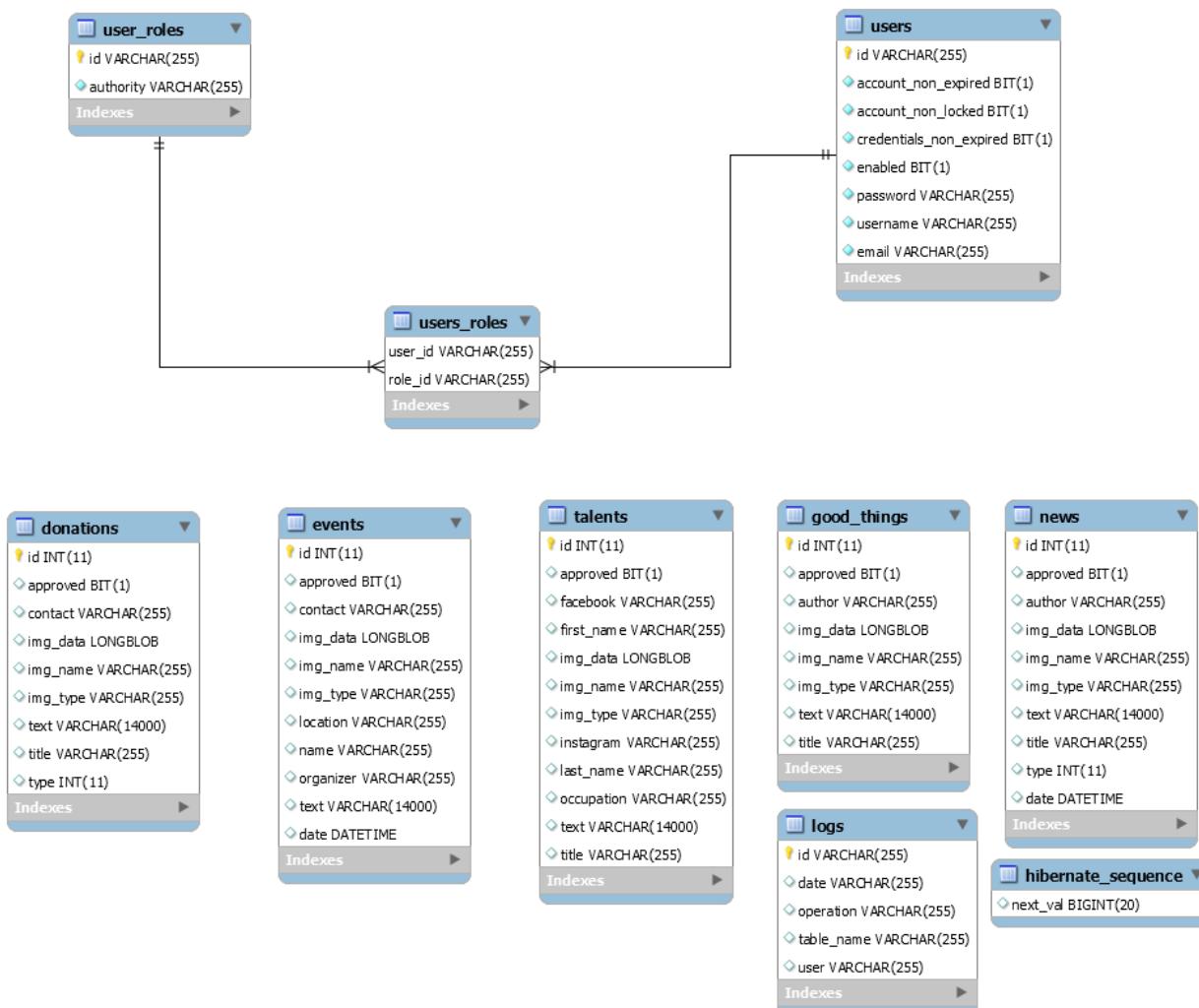
3.15 – Login workflow

4. Програмна реализация

За реализация на бек-енда на уеб приложението е използван Spring framework-а на програмния език Java. Относно клиентската част(фронт-енда) е ползван езика HTML, съобразен с CSS и JavaScript(jQuery), докато за server-side рендъринг на данните от бек-енда е използван Thymeleaf, който отлично работи в комбинация със Spring.

4.1 Реализация на базата данни

Почвайки най-долу, на фиг. 4.1 е представена схемата на базата данни която представлява мястото, където ще се съхраняват всичките данни необходими за работата на нашето приложение.



4.1 – Схема на базата данни (`negotino_db`)

Всички таблици описани на фигура 4.1 представляват Java класове от нашето приложение. Те съответстват на класовете от пакета entities и представляват частта M(Model) от шаблона MVC(Model-View-Controller).

4.1.1 Пример 1 – Таблица потребители (users)

Таблицата на потребителите(users) в кода е написана по следващия принцип:

```
@Entity
@Table(name = "users")
public class User implements UserDetails
{
    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(
        name = "UUID",
        strategy = "org.hibernate.id.UUIDGenerator"
    )
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private String id;
    @Column(name = "username", nullable = false, unique = true)
    private String username;
    @Column(name = "password", nullable = false)
    private String password;
    @Column(name = "email", nullable = false)
    private String email;

    private boolean isAccountNonExpired;

    private boolean isAccountNonLocked;

    private boolean isCredentialsNonExpired;

    private boolean isEnabled;

    private Set<UserRole> authorities;
}
```

Код 4.2 – Модела на потребителите

Лесно може да види, че класа всъщност представлява таблицата users, докато неговите атрибути са колоните в базата данни. Това е генерирано като едно цяло в базата данни при първото стартиране на проекта. Особеното тук е че уникалното число на user-ите е от тип стринг, с помощ на uuid генератор за една идея по-голяма сигурност на потребителите.

Както е посочено в проектирането на базата данни, ролите на потребителите са описани в отделен клас за по-качествена нормализация на базата от данни.

4.1.2 Пример 2 – Таблица роли (user_roles)

Класа който съответства на таблицата за ролите е следния:

```
@Entity
@Table(name = "user_roles")
public class UserRole implements GrantedAuthority
{
    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(
        name = "UUID",
        strategy = "org.hibernate.id.UUIDGenerator"
    )
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private String id;

    @Column(name = "authority", nullable = false)
    private String authority;
}
```

Код 4.3 – Модела на ролите

Така получаваме две таблици – един за потребителите, друг за всичките възможни роли в приложението. Свързването им е много към много(Many-To-Many) и е реализирано в класа User (Код 4.4).

```
@Override
@ManyToMany(cascade = CascadeType.ALL
            , targetEntity = UserRole.class
            , fetch = FetchType.EAGER)
@JoinTable(
    name = "users_roles",
    joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn(name = "role_id")
)
public Set<UserRole> getAuthorities()
{
    return this.authorities;
}
```

Код 4.4 – Връзка помежду потребители и роли

4.1.3 Пример 3 – Таблица таланти (talents)

Представени са основните атрибути на частта за талантите. Тук както беше посочено в проектирането id-то че е уникално число което се инкрементира за едно за целта на по-лесно сортиране и анализиране на публикациите. Колоната img_data е от типа на голям обект за успешно запазване на данните на снимката качена от потребителя, докато атрибута approved ще сторва булолеан тип за това дали поста е публикуван успешно или не.

```
@Entity
@Table(name = "talents")
public class Talent
```

```
{
    @Id
    @GeneratedValue
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private int id;
    @Column(name = "title")
    private String title;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @Column(name = "occupation")
    private String occupation;
    @Column(name = "text", length = 14000)
    private String text;
    @Column(name = "facebook")
    private String facebook;
    @Column(name = "instagram")
    private String instagram;
    @Column(name = "img_name")
    private String imgName;
    @Column(name = "img_type")
    private String imgType;
    @Lob
    @Column(name = "img_data")
    private byte[] imgData;
    @Column(name = "approved")
    private boolean approved;
}
}
```

Код 4.4 – Модела на талантите

Всички други таблици(news, events, good-things, donations) използват аналогичен начин за създаване на техните модели които са основата на шаблона Model-View-Controller.

4.2 Реализация на потребителската част

Приложението навлиза малко повече на вътре относно решаване на основните проблеми и въпроси които възникват реализирайки функционалността за главните двигатели в сайта - потребителите. Основно, за по-качествени услуги портала има за цел да не изисква притежаване на акаунт за достъп до неговите ресурсите. Той трябва да е общо достъпен за всичките, обаче за достъпа до основните функционалности които предлага платформата е необходим наличен профил. Тук веднага излизат маса на полемики относно достъпа и сигурността на потребителите. В следващите под точки се описва подробно какви решения са взети и съответно как са имплементирани.

4.2.1 Потребителски роли

Основния въпрос при една такава имплементация е как да се разделят посетителите на сайта. За тази цел, е взето решение да постои йерархия помежду

ролите. Ясно да се знае кой потребител каква роля притежава и съответно какви права има. С тази идея основните роли са разделени на:

- ROOT-ADMIN (Owner)
- ADMIN
- MODERATOR
- ROLE_USER (обикновен потребител)

Съответно, когато един посетител на сайта, без при това да се логне в него, не притежава определена роля т.е е анонимен или гост.

Притежаването на определена роля означава че имаш всичките права като тези под теб, плюс някои други които са специфични за твоята роля. Примерно, ако си admin чиято роля е на 2-ро място гледано от горе към долу, означава че ти си обикновен потребител, модератор, админ и наследяваш всички техни права, съответно имаш допълнителни такива които са особени за админите. Обаче, нямаш правата които има ролята над теб – в случая ROOT-ADMIN. Тази логика ни помага ясно да добавяме само новите права на потребителя с определена роля, без да ги забравяме основните.

Основната функционалност на уеб приложението е споделяне на публикация в съответните секции. Като такава, достъп до нея имат само и само посетителите, които притежават определен акаунт и са логнати в него. Това е така защото всичките налични акаунти в приложението имат минимум ролята обикновен потребител (ROLE_USER) за която са налични правата за споделяне. Така, споделянето на разните информации при логнат акаунт става сравнително лесно и бързо. Обаче, както беше описано в проектирането, за предпазване от нежелани постове, въвежда се основна проверка върху публикацията. Този филтър притежават потребителите с роля модератор или нагоре. При тях идва post request-а, преглеждат го и съответно имат възможност да го публикуват или отхвърлят. Допълнително към ролята модератор се добавя нова част която е преглеждане на историята на действията на всичките потребители т.е преглеждане на логовете. С възможността да изчистят определени или всички логове. Ролята която е над модератора е админа. Той наследява правата на модератора и обикновения потребител и притежава админските функционалности. Той вече може да изтрие и вече публикувани постове и да ги махне изцяло от сайта. Също така админите могат да достъпят допълнителна част която се казва организация. Там са листнати всичките потребители с потребителското име и ролята. Съответно, админите имат права да променят роли и трият определени акаунти който не са следвали правата на сайта. Тук особеното е не можеш да смениш своята роля или изтриеш себе си. Това е така за да се предпазим от нежелани последствия. Ако толкова искаме да сменим ролята или ни се изтрие акаунта, решението е да кажем на owner-а или някой друг админ да го направи това. Така, най-накрая идва най-главния т.е човека с най-големите права в сайта – притежателя на приложението. Въщност, името на ролята която притежава ROOT-ADMIN ни подсказва че самия той е един вид админ. Тук, веднага идва въпроса „Защо

тогава да го отделяме от другите админи?”. Но, тук идеята е че примерно ако ние като автор на приложението и съответно главен потребител притежаваме ролята админ и сме на една съща линия с останалите админи има голям шанс да изгубим статута на owner. Това става тогава когато ние ще дадем ролята админ на някой потребител и той съответно изтрие нашия акаунт или смени нашата роля на обикновен потребител. Тогава, ние губим всичко и той става най-главния в приложението, па съответно може да си прави каквото си поиска. За това, се въвежда ролята ROOT-ADMIN която е в притежание на само един единствен човек – owner-а. Така, ще е забранено изцяло да се трябва или смени ролята на ROOT-ADMIN-а. Админите които са под него може отново да си правят каквото поискат обаче знаят че никога няма да вземат мястото на owner-а. Даже вече и при правене на безразсъдни работи (триене на останали админи, сменяне на роли и т.н) някой от горе ги гледа и може да ги махне.

4.2.2 Сигурност на потребителите

За имплементацията на сигурността на потребителите въвеждаме един от най-силните компоненти на Spring, Spring Security. Библиотека която се грижи потребителските сесии, токени и всичко останало. Spring като цяло овозможава много лесно интегриране на този компонент. С няколко стъпки приложението може да придобие основните функционалности относно потребителската част.

Един малък фрагмент от нашата конфигурация на Spring Security ясно посочва кое view се отнася към логин страницата (/login) и по кои параметъри да става логването, в случая потребителско име и парола. defaultSuccessUrl означава къде да не метне приложението при успешно логване, за нашите цели това сме го наконфигурирали на нашата начална страница.

```
.formLogin()
.loginPage("/login")
.usernameParameter("username")
.passwordParameter("password")
.defaultSuccessUrl("/")
```

Код 4.5 – Spring security - Login settings

Останалите части които се включват в конфигурацията на потребителската част са помнене на логнатия потребител за определено време (в случая е сетнато на 1200 секунди), хендъринг на не-авторизирани страници, възможност на логване с помощ на OAuth2.0 стандарта, излизане от приложението и други възможности за нашите потребители.

4.2.3 Регистрация на нов потребител

По принцип е хубаво колкото е възможно по-рано да се валидират полетата за съответно по-бърз отговор и предотвратяване от ненужното навлизане на вътре. Така, при нашата форма за регистрация съществуват проверки относно дължината на полетата и техния инпут. Но, все пак има неща които няма как да се проверяват директно

на фронт-енда защото трябва да се провери в базата данни. Един такъв пример е проверката за съществуващо потребителско име което сме посочили че трябва да е уникално.

При успешна валидация, принципът за регистрация на нов потребител е следният – данните за регистрацията се изпращат чрез POST заявка до сървъра. Следва хендълване от контролера който се отнася за регистрирането на потребителите (Код 4.6). Там става проверката за дали паролата и конфирмацията на паролата са еднакви и при еднаквост вика метода createUser(Код 4.7), след което проверява дали потребителя е успешно записан или потребителското име е вече заето. createUser метода като параметър приема binding модела изпратен от потребителя. В метода на модела му се криптира паролата за да не е видима от никого и като такава се записва в базата данни. При всяко извикване на метода за регистрация се проверява дали таблицата за потребители е празна. При резултат че няма потребители метода автоматически създава първия потребител с роля ROOT-ADMIN или owner-а на приложението. Реализирането на това показва добре как е имплементирана юерархията на ролите с това че root-admin притежава и всичките роли които са под него. Докато, ако вече има налични потребители заделя се ролята USER_ROLE т.е обикновен потребител. Тук не се заделят ролите от типа на админ или модератор защото за тях е направена отделна част където промяната на роли ще става по лесен и бръз начин. На края от метода се хваща изключение относно уникалността на потребителското име.

```

@PostMapping("/register")
@PreAuthorize("isAnonymous()")
public String registerPost(@ModelAttribute("user") UserRegisterBindingModel
userRegisterBindingModel, HttpServletResponse response, Model model)
{

    if(!userRegisterBindingModel.getPassword().equals(userRegisterBindingModel.getConf
irmPassword()))
    {
        response.setHeader("PASSWORDS_UNMATCH", "1");
        String error = "Пасвордите не се совпадаат.";
        model.addAttribute("error", error);
        return "login/register.html :: #error";
    }

    boolean created = this.userService.createUser(userRegisterBindingModel);

    if(created == false)
    {
        response.setHeader("TAKEN_USERNAME", "1");
        String error = "Корисничко име вече постои.";
        model.addAttribute("error", error);
        return "login/register.html :: #error";
    }

    this.newRegister = true;
    return "login/login.html";
}

```

Код 4.6 – Контролера на регистрацията

```

public boolean createUser(UserRegisterBindingModel userRegisterBindingModel)
{
    User userEntity = this.modelMapper.map(userRegisterBindingModel,
User.class);

    userEntity.setPassword(this.bCryptPasswordEncoder
.encode(userEntity.getPassword()));
    Set<UserRole> authorities = new HashSet<>();

    if(this.userRepository.findAll().isEmpty())
    {
        authorities.add(userRoleFactory.createUserRole("ROOT-ADMIN"));
        authorities.add(userRoleFactory.createUserRole("ADMIN"));
        authorities.add(userRoleFactory.createUserRole("MODERATOR"));
        authorities.add(userRoleFactory.createUserRole("ROLE_USER"));
    }
    else
    {
        authorities.add(this.userRoleService.findRole("ROLE_USER"));
    }

    userEntity.setAuthorities(authorities);

    try
    {
        this.userRepository.save(userEntity);
        return true;
    }
    catch (DataIntegrityViolationException e)
    {
        return false;
    }
}

```

Код 4.7 – Регистрация на потребител

4.2.4 Автентикация на потребители

Входа на потребители става в страницата за логин чрез попълване на потребителско име и парола. Отново тук има валидация на въвеждащите данни, тъй като метода за намирането им в базите данни връща грешка ако даденото не съответства въведеното потребителско име или парола. Съответно при успешно логване, авторизира ни с дадената роля която притежаваме и пуска ни в началната страница на сайта

```

public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
{
    User foundUser = this.userRepository
                    .findByUsername(username)
                    .orElse(null);

    if(foundUser == null)
    {
        throw new UsernameNotFoundException("User not found.");
    }
    return foundUser;
}

```

Код 4.8 – Автентикация на потребител

4.2.5 Авторизация на потребители

За определяне към кои ресурси има права логнатия акаунт, съответно трябва да се извърши authorization и да се провери каква роля притежава. Има два начина за извършване на авторизацията: при контролерите или директно в html-а с помощта на thymeleaf.

a) Контролери

В нашето приложение най-често ползваме анотацията @PreAuthorize която означава главно предусловие за това дали дадения метод на мапинга на контролера е достъпен за потребител. Код 4.9 показва как един пост в секцията за таланти може да бъде приет само от потребителите с роля модератор. Тук идва идеята на йерархията на потребителите, така че тази функционалност е достъпна и за админите и роот админа защото те вътрешно притежават и ролята модератор.

```
@PostMapping("/approve")
@PreAuthorize("hasAuthority('MODERATOR')")
public String approveTalent(@RequestParam("talentApproveId") int id,
Authentication authentication)
{ . . . }
```

Код 4.9 – Авторизация на потребителите в контролерите

Авторизацията показана на Код 4.9 е аналогично ползвана за всички останали части където трябва да разграничим достъпа към отредените роли.

б) Изгледи

В изгледите авторизацията се реализира по подобен начин. Там thymeleaf технологията подпомага да определим дали логнатия акаунт притежава необходимата роля за да достъпи ресурса който предлага самия изглед (view).

Един хубав пример е приказан на фрагмента на 4.10 където за по лесно изясняване се избира същия проблем както в подточката за контролерите. Тук в този клас с id="req-talents" трябва да се рендерерират всичките събъмтнати форми от обикновените потребители който се очаква да бъдат приети или отхвърлени от модераторите или админите. С помощта на sec:authorize ние указваме на това тази част да е достъпна само за потребителите които притежават роля модератор.

```
<div id="req-talents" sec:authorize="hasAuthority('MODERATOR')" class="requested-talents m-2">
    ...
</div>
```

Код 4.10 – Авторизация на потребителите в изгледите

Аналогични решения се използват навсякъде в изгледите.

4.3 Реализация на постовете в разните секции

Основата функционалност на потребителите е да движат сайта с свои публикации в разните секции. Както беше описано в проектирането живота на поста минава низ няколко фази преди официално да е публикуван на сайта.

4.3.1 Постване от обикновени потребители (AJAX)

Обикновени потребители се смятат тези потребители които притежават акаунт и нямат назначени по-големи права или пък се логнат с помощ на Facebook или Google. Тези профили имат правата да споделят публикации в разните секции на сайта. Това става чрез форма в която има полета за определените информации. Повечето от инпутите имат валидация на фронт-енда която изисква да не останат празни, има полета в които трябва да се въведе дата и време където не е допустим текст и прикачване на снимка към съответния пост чийто изглед се вижда още в първия момент.

Особеното тук е че всичките постове се реализират асинхронно с помощ на AJAX. Тази техника предотвратява редиректването към същата страница и дава възможността информацията да си мине през сървъра във базата данни асинхронно докато през същото време ние да можем да събитнем нова публикация или да преглеждаме страницата.

Кода 4.11 показва как се реализира валидацията и AJAX заявката. Тази функция писана на JavaScript се отнася към формата с клас needs-validation която реално е частта за добавяне на пост в секцията за добри дела. С това успяваме да се предпазваме от всякакви събmitти които не изпълняват дадените requirement-и за отделните полета. При всяко неуспешно събmitване html-а изкарва съответната грешка. Когато всички условия ще са изпълнени се реализира POST заявка с помощ на AJAX. При успешен опит на екрана излиза модал за информиране на потребителя и частта където се приказват request-натите добри дела се ъпдейтва. След това изкараната снимка се изтрива, формата се ресетира и е вече е отново достъпна за ползване.

```
(function() {
  'use strict';
  window.addEventListener('load', function() {
    var forms = document.getElementsByClassName('needs-validation');
    var validation = Array.prototype.filter.call(forms, function(form) {
      form.addEventListener('submit', function(event) {
        if (form.checkValidity() === false) {
          form.classList.add('was-validated');
          event.preventDefault();
          event.stopPropagation();
        }
      else
      {
        event.preventDefault();
        event.stopPropagation();
        var formData = new FormData($('.needs-validation')[0]);
      }
    });
  });
})()
```

```
$ajax({
    url: "/good-things/add",
    method: "POST",
    data: formData,
    success: function (data, textStatus)
    {
        $("#success_tic").modal();
        $("#req-goodthings").replaceWith(data);
    },
    contentType: false,
    processData: false,
    cache: false,
});

$("#removeImage").click();
form.classList.remove('was-validated');
form.reset();

}
}, false);
});
},
false);
})();
```

Код 4.11 – Validation & AJAX

Подобна валидация и AJAX имплементация се използва при всичките изгледи на отделните секции.

Код 4.12 представя контролера който върши POST заявката която е достъпна само за обикновените потребители. Почва с това че сетва атрибута на доброто дело `approved` на 0(false) което означава че поста не е обявен публично т.е е request-нат и се очаква някой от по-горните роли да прихване и сетне на 1(true). След това идва проверката за дали потребителя е сложил снимка или не и съответното извикване на методите за добавяне на поста с потребителска или дефолтна снимка. В следващия фрагмент идва проверката дали логнатия акаунт е модератор и съответно вземане на request-натите постове. На края добавяне на извършената операция в логовете които ще изясним в следващите секции. Return-а ни представлява ъпдейт на div-а с id `req-goodthings` което въсъщност е обнова на поисканите публикации с вече най-новата (току що споделената).

```
@PostMapping("/add")
@PreAuthorize("hasAuthority('ROLE_USER')")
public String addGoodThing(@ModelAttribute GoodThing goodThing,
@RequestParam("files") MultipartFile file, Model model, Authentication
authentication) throws IOException
{
    goodThing.setApproved(false);

    if(!file.isEmpty())
    {
        this.goodThingsService.addGoodThing(goodThing,file);
    }
    else
    {
        this.goodThingsService.addGoodThing(goodThing);
```

```

    }

Authentication auth = SecurityContextHolder.getContext().getAuthentication();
if (auth != null && auth.getAuthorities().stream()
.anyMatch(a -> a.getAuthority().equals("MODERATOR")))
{
    List<GoodThing> requestedGoodThings =
    this.goodThingsService.getRequestedGoodThings();

    model.addAttribute("requestedGoodThings", requestedGoodThings);
    ImageEncoder imageEncoder = new ImageEncoder();
    model.addAttribute("defaultImageGoodThing", defaultImageGoodThing);
    model.addAttribute("imageEncoder", imageEncoder);
}

Log log = Logs.setupLog(authentication, "Good Things", Operation.Add);
this.logService.insertLog(log);

return "pages/good-things.html :: #req-goodthings";
}

```

Код 4.12 – Контролера който имплементира POST заявка за добавяне на добро дело

Кода 4.13 показва имплементацията за добавяне на доброто дело без снимка който се извиква когато потребителя не прикачил. Идеята тук е чрез друга отделна имплементация да има вече запазена default-на снимка която да се прикаже винаги когато такава не е прикачена от потребителя. С това целим към по-хубав потребителски интерфейс.

```

@Override
public void addGoodThing(GoodThing goodThing)
{
    this.goodThingRepository.save(goodThing);
}

```

Код 4.13 – Метода за добавяне на добро дело без снимка

Докато метода който имплементира добавяне на добро дело с въведената снимка от страна на потребителя записва във базата данни байтовете от снимката за след това да ги вземе и представи като снимка(Код 4.14).

```

@Override
public void addGoodThing(GoodThing goodThing, MultipartFile file) throws
IOException
{
    String imgName = file.getOriginalFilename();
    goodThing.setImgName(imgName);
    goodThing.setImgType(file.getContentType());
    goodThing.setImgData(file.getBytes());

    this.goodThingRepository.save(goodThing);
}

```

Код 4.14 – Метода за добавяне на добро дело със снимка

За гетване на всички постове поискани от потребителите се търси директно в базата данни с ключовата дума за approved=false която се assign-ва винаги когато се споделя

пост. Листа с всички такива постове се става обратен за да се вземат постовете от най-нови към най-стари защото първичният ключ (id-то) се инкрементира всеки път за едно повече.

```
@Override
public List<GoodThing> getRequestedGoodThings()
{
    List<GoodThing> requestedGoodThings =
        this.goodThingRepository.findAllByApprovedFalse();
    Collections.reverse(requestedGoodThings);

    return requestedGoodThings;
}
```

Код 4.15 – Метода за взимане на всички request-нати добри дела

Процеса на поставане на добро дело е описана цялостно тръгвайки от фронт-енда към бек-енда. Аналогични имплементации се прилагани и за другите секции(таланти, новини, събития, дарения).

Навсякъде, инputа от обикновените потребители отива като request-нат post при модераторите или ролите нагоре където се очаква приемането или отхвърлянето му чиято реализация ще изясним в следващата част 4.3.2.

4.3.2 Приемане/Отхвърляне на постовете от модераторите

Основно модераторите са потребителите които ще решават какво да се включи и какво не в нашия уеб сайт. Чрез POST заявки за approve и discard имат възможност за публикуване на вече request-нати постове от потребителите. Контролерите за приемане и отхвърляне съответно са приказани на код 4.16 и код 4.17

```
@PostMapping("/approve")
public String approveGoodThing(@RequestParam("goodThingApproveId") int id,
Authentication authentication)
{
    boolean approved = this.goodThingsService.approveGoodThing(id);
    Log log = Logs.setupLog(authentication, "Good Things", Operation.Approve);
    this.logService.insertLog(log);

    return "redirect:/good-things";
}
```

Код 4.16 – Контролер за прихващане на пост за добро дело

```
@PostMapping("/discard")
public String discardGoodThing(@RequestParam("goodThingDiscardId") int id,
Authentication authentication)
{
    boolean discarded = this.goodThingsService.discardGoodThing(id);
    Log log = Logs.setupLog(authentication, "Good Things", Operation.Discard);
    this.logService.insertLog(log);

    return "redirect:/good-things";
}
```

Код 4.17 – Контролер за отхвърляне на пост за добро дело

Методите approveGoodThing и discardGoodThing които вършат целта на прихващане и отхвърляне съответно са имплементирани в сервисите и приказани на код 4.18 и код 4.19.

```
@Override
public boolean approveGoodThing(int id)
{
    Optional<GoodThing> getRequestedGoodThing =
this.goodThingRepository.findById(id);

    if(getRequestedGoodThing.isPresent())
    {
        getRequestedGoodThing.get().setApproved(true);
        this.goodThingRepository.save(getRequestedGoodThing.get());
        return true;
    }

    return false;
}                                Код 4.18 – Метод за прихващане на изпратения пост
```

```
@Override
public boolean discardGoodThing(int id)
{
    Optional<GoodThing> getRequestedGoodThing =
this.goodThingRepository.findById(id);

    if(getRequestedGoodThing.isPresent())
    {
        this.goodThingRepository.delete(getRequestedGoodThing.get());
        return true;
    }

    return false;
}
```

Код 4.19 – Метод за отхвърляне на изпратения пост

Подобни функции се реализират относно приемане или отхвърляне на всичките request-нати наведнъж което подпомага за по-бърза реализация.

Аналогични методи за контролери и методи за реализация на функцията която притежават потребителите с роля модератор са имплементирани при всички останали секции.

4.3.3 Отхвърляне на съществуващи постове от админите

Админите имат допълнителна функционалност за триене на вече публикувани постове в разните секции. С това админа представя роля над модератора и ако реши че някоя публикация не е съответна за секцията може да я изтрие изцяло от сайта. Функцията е подобно имплементирана както фрагмента на код 4.19.

4.4 Реализация на намиране на топ 3 добри дела

Една от функционалностите която добре показва как нашето приложение работи директно върху базата от данни с помощ на абстрактния слой предоставен от компонента Spring Data е намирането на топ 3 създатели на добри дела. Чрез създаване на JPA репозиторита ние можем правим директно заявка към базата данни и да получим желания резултат. На код 4.20 се вижда имплементацията на цялото repository на добри дела. Там добре се вижда че в името на самото деклариране на методите се показва какво да се вземе от базата данни което е мощността на Spring Data. Обаче за посложните неща които се нуждаят на програмиста се пишат кратки Query-тата като в декларацията на четвртия метод в този интерфейс. Там имплементираме наша специфична цел която в момента е вземане на три най-добри създатели на добро дело. За това групираме таблицата за добри дела по авторите на вече публикуваните постове, броим колко добри дела има всеки автор и ги сортираме низходящо според тази бройка.

```
public interface GoodThingRepository extends JpaRepository<GoodThing, Integer>
{
    List<GoodThing> findAllByApprovedFalse();
    List<GoodThing> findAllByApprovedTrue();
    List<GoodThing> findTop10ByApprovedTrueOrderByIdDesc();

    @Query("SELECT u.author, count(*) as v FROM GoodThing u WHERE u.approved=1
group      by u.author order by v DESC ")
    List<Object[]> getTopThreeAuthors();

}
```

Код 4.20 – GoodThings Repository Interface

Следвайки проектирането на нашето уеб приложение, метода от този интерфейс е извикан в сервизите където се взимат първите три. Тези неща се слагат в LinkedHashMap с целта да се сложе името на автора и съответно сумата на добрите дела които направи. След това тази информация да се сервира на контролера и view-то.

```
@Override
public LinkedHashMap<String, Long> getTopThreeAuthors()
{
    List<Object[]> list = this.goodThingRepository.getTopThreeAuthors();

    if(list.size() >= 3)
    {
        LinkedHashMap<String, Long> topThreeAuthors = new LinkedHashMap<String,
        Long>();

        for (int i = 0; i < 3; i++)
        {
            String key = (String) list.get(i)[0];
            long value = (Long) list.get(i)[1];
            topThreeAuthors.put(key, value);
        }
        return topThreeAuthors;
    }
}
```

```

    return null;
}

```

Код 4.21 – Вземане на топ 3 автори на добри дела

4.5 Реализация на търсене на новина

Новините като част от всеки момент от нашето ежедневие се случват непрекъснато. Нашата платформа като такава предлага бързото и простото му публикуване. Това води до много постове в секцията за новини което създава конфузия при търсене на определена новина.

Реализацията на търсачката на новини се намира отново в репозиторитата т.е интерфейса на новините. Там е деклариран метода findAllByTitleContains на Код 4.22 където в самото име е дефинирано че искаме да вземем от базата данни само публикувани новини които в своето заглавие съдържат стринга, които му е зададен като параметър. Spring Data компонента предоставя

```

public interface NewsRepository extends JpaRepository<News, Integer>
{
    List<News> findAllByApprovedFalse();
    List<News> findAllByApprovedTrue();
    List<News> findAllByTitleContains(String title);
}

```

Код 4.22 – News Repository Interface

Викането на този метод става отново в сервисите(Код 4.23) където се връща листа от новини съдържащи въведената ключова дума.

```

public List<News> searchTitle(String title)
{
    return this.newsRepository.findAllByTitleContains(title);
}

```

Код 4.23 – Търсене на новина

4.6 Реализация на логовете (потребителски истории)

Всички потребители имат шанса да правят работи, за които притежават права в нашето приложение. Обаче, по някога тези права се злоупотребяват и идва момента когато идеята на сайта ходи към отрицателна посока. Така, за качествено следване на действията на всичките акаунти е имплементирана частта за потребителските истории. Целта е предпазване от нежелани активности. Няколко примери за това може да бъдат:

- Неверно съдържание добавено от обикновените потребители
- Не хубави публикации от страна на обикновените потребители
- Нелогически отхвърляне на request-нати постове от модераторите
- Нелогически приети request-нати постове от модераторите
- Отхвърляне на вече публикувани постове без причина от админите

- Не подходящо променена роля на определен потребител (функционалност описана на 4.7.1 Смяна на ролята на определен потребител)
- Изтриване на акаунт без причина (4.7.1 Изтриване на определен потребител)

Всички тези и много други може да засегнат сайта към нежелан output. За това частта на логовете ни предоставя хубаво предоставлен потребителски интерфейс в който ясно ще се вижда какви дейности са извършени върху сайта. Така може точно да се види кой потребител какво направил, в коя секция и кога. Цялата информация е добре структурирана с хубава възможност на сортиране по определена колона или търсене по ключова дума.

Освен идеята за намиране на грешки, потребителските истории може да бъдат полезни примерно и за:

- Анализ на дейностите на потребителите
- Да видим най-активните потребители
- Да видим кои секции са най-популярни
- Да анализираме по кое време сайта е най-вече използван
- Да видим кой модератор върши най-много работа
- Как нарастват потребителите

Тези и други подобни примери могат да служат за правене на анализ и статистики които могат да помогнат относно подобряване на сайта.

Достъп до потребителските истории имат всички потребители които притежават ролята модератор. Те могат да преглеждат всички логове и съответно да анализират нещата. Обаче, модератори нямат права да трят логовете. Тази възможност има само owner-а на приложението т.е потребител с роля ROOT-ADMIN. Тази идея е реализирана за целта никой от модераторите или админите да може да направи злоупотреба на сайта и съответно изtrie историята зад себе си.

Реализацията на методите на потребителските истории е извършена в сервиза за логовете. Докато, самите функционалности са извиквани от почти всички контролери с цел добавяне на лог за извършената операция. На кода 4.24 е показан статичен метод на общия клас Logs. Името на този метод е setupLog който има целта да състъпне лог като създаде обект от тип Log(модела в базата данни). На този обект както е показано във фрагмента 4.24 се заделя името на потребителя който е логнат, операцията която направил и таблицата в базата данни която променил, като всички тези се задават като параметри в метода. Също така на атрибута за времето се заделя текущото време в който е извикан този метод.

```
public static Log setupLog(Authentication authentication, String table, Operation operation)
{
    Log log = new Log();
    log.setTable(table);
    log.setOperation(operation);
    log.setAuthentication(authentication);
    log.setTimestamp(new Date());
    return log;
}
```

```

Date date = new Date();
log.setUser(authentication.getName());
log.setOperation(operation);
log.setTableName(table);
log.setDate(dateFormat.format(date));

...
return log;
}

```

Код 4.24 – Сетьпване на лог

Този метод се извиква във всеки контролер в който се върши CRUD операция върху базата данни. Фрагмента 4.25 представлява пример от контролера за даренията. Този принцип се ползва аналогично при всичките контролери.

```

Log log = Logs.setupLog(authentication, "Donations", Operation.Add);
this.logService.insertLog(log);

```

Код 4.25 – Добавяне на лог

С цел изчистване на потребителските истории, определен лог или всичките логове както посочихме могат да бъдат изтрити само и само от owner-а. Реализацията на тези методи е показана на 4.26 и 4.27.

```

@Override
public void removeLogsById(List<String> list)
{
    for(String uuid : list)
    {
        Log log = this.logsRepository.findLogById(uuid);
        this.logsRepository.delete(log);
    }
}

```

Код 4.26 – Триене на определени логове от owner-а

```

@Override
public void removeAllLogs()
{
    this.logsRepository.deleteAll();
}

```

Код 4.26 – Триене на всички логове

4.7 Реализация на организацията на потребителите

Уеб приложението предоставя един приятен потребителски интерфейс в който лесно и ефективно може да организираме нашите потребители. Тази част е достъпна само за потребителите които притежават ролята админ. В нейния изглед са листнати всичките потребители на сайта и тяхната роля. Като за всеки един потребител може да се извършат две функционалности:

- Промяна на ролята им
- Триене на потребителя

4.7.1 Промяна на ролята на определен потребител

Първата функционалност дава възможността лесно да дадем права на определен потребител. По принцип, до потребителското име е посочена ролята която притежава. Веднага до него с dropdown мени са листнати всички други възможни роли които може да му заделим. Като ролята ROOT-ADMIN не е достъпна и не може да бъде заделена на никого защото е уникална и само owner-а на приложението я има. Тук също е важно да се спомене че листата на потребители не съдържа текущо логнатия акаунт. Идеята тук е да нямаме възможност да си променим нашата роля за да пренебрегнем нежелани резултати относно текущите права които притежаваме. След избиране на поисканата роля и натискане на save button-а се вдигат заделените контролери които пък извикват бизнес логиката за смяна на роля. Метода changeRole посочен на код 4.27 приема две параметри: потребителя който е избран и новата роля която искаме да заделим. Търси се посочения потребител в базата данни и му се заделя новата роля както и всички роли които са под нея с което добре се описва логиката на юерархията на ролите. На края се ъпдейтват връзките във таблицата users_roles която междинна таблица между потребителите и ролите и е по принципа на many-to-many relation поради факта че един потребител може да притежава повече роли и една роля може да е заделена на повече потребители.

```

@Override
public boolean changeRole(String username, String newRole)
{
    User user = (User)loadUserByUsername(username);
    Set<UserRole> authorities = new HashSet<>();

    if(newRole.equals("ADMIN"))
    {
        authorities.add(this.userRoleService.findRole("ADMIN"));
        authorities.add(this.userRoleService.findRole("MODERATOR"));
        authorities.add(this.userRoleService.findRole("ROLE_USER"));
    }
    else if(newRole.equals("MODERATOR"))
    {
        authorities.add(this.userRoleService.findRole("MODERATOR"));
        authorities.add(this.userRoleService.findRole("ROLE_USER"));
    }
    else if(newRole.equals("ROLE_USER"))
    {
        authorities.add(this.userRoleService.findRole("ROLE_USER"));
    }
    else
        return false;

    user.setAuthorities(authorities);

    return this.userRepository.save(user) != null;
}

```

Код 4.26 – Промяна на ролята

4.7.2 Изтриване на определен потребител

Втората функционалност която може да се направи върху акаунтите в уеб сайта е изтриването на определен потребител. Тази активност не е много желана и е хубаво да се прави много рядко. Точно заради това при всеки опит за изтриване на селектнат потребител системата не пита допълнително, за дали сме сигурни за това. Обаче, от друга страна е много хубаво да се предпазваме от потребители които правят негативни работи върху сайта ни. Точно тази функционалност дава ни възможност много бързо и лесно да предпазим приложението от лоши потребители. Реализацията на тази функционалност е посочена в метода deleteUser който се намира в UserServiceImpl където според проектирането се намира бизнес логиката на приложението. Там отново става намиране на потребителя по зададено потребителско име като параметър на метода. След намирането му трябва да го изтрием. Обаче, тук е малко особено защото при директно изтриване на акаунта ще дойде до голяма грешка и объркване. Това е така защото дадения потребител е свързан с определена роля/роли в таблицата users_roles. За тази цел, първо изтриваме всичките му заделени роли чрез сътванието им на null за след това успешно да го изтрием от таблицата users.

```
public void deleteUser(String username)
{
    User user = (User)loadUserByUsername(username);
    user.setAuthorities(null);
    this.userRepository.delete(user);
}
```

Код 4.26 – Промяна на ролята

4.8 Реализация относно информацията за вируса COVID-19

Съвсем най-популарната тема за тази година е вируса който завляда света наречен корона вирус или COVID-19. За по-качествено здравеопазване е имплементирана информация относно общата бройка и новите за деня заразени, умрели, излеченит както глобално за света, така и за всички земи отделно.

За цел е реализиран GET HTTP метод към API което предоставя цялата информация относно текущото състояние на статистиките. Тези данни трябва по някакъв начин да се рендерират и вземе само това от което се нуждаем

Цялата реализация се случва в класа `CoronalInfo` който дефинира статични методи. Първия от тях е гетване на ресурсите като JSON обект (.

```
public static JSONObject getApiResponse() throws IOException
{
    OkHttpClient client = new OkHttpClient().newBuilder()
        .build();
    Request request = new Request.Builder()
        .url("https://api.covid19api.com/summary")
        .method("GET", null)
        .build();
    Response response = client.newCall(request).execute();
    String jsonString = response.body().string();
    JSONObject obj = new JSONObject(jsonString);

    return obj;
}
```

Код 4.27 – GET HTTP метод към целевото API

Следващия метод реализира вземането на глобалната информация т.е текущото състояние на ниво свят. Този метод като параметър поема JSON обект и ObjectMapper за мапване на обекта към определен клас. При извикване на този метод ще се подаде получения резултат от предишния метод – `getApiResponse()`. Докато, като върнат параметър този метод връща обект от класа `Global` показан на код 4.29.

```
public static Global getGlobalInfo(JSONObject obj, ObjectMapper mapper) throws
IOException
{
    JSONObject globalJSON = obj.getJSONObject("Global");
    Global global = mapper.readValue(globalJSON.toString(),
        new TypeReference<Global>(){});

    return global;
}
```

Код 4.28 – Вземане на глобалното текущо състояние

```
public class Global
{
    public int NewConfirmed;
    public int TotalConfirmed;
    public int NewDeaths;
    public int TotalDeaths;
    public int NewRecovered;
```

```
    public int TotalRecovered;
}
```

Код 4.29 – Класа който приема глобалната информация

Следващия метод взима текущото състояние в отделните земи. Тук се реализира подобна логика както в горния метод. Допълнително на края се сортира листата на земи по компаратора CoronaConfirmedComparator описан на код 4.31. На края се връща листа от обекти от тип Country – клас показан на код 4.32.

```
public static List<Country> getCountriesInfo(JSONObject obj, ObjectMapper mapper)
throws IOException
{
    JSONArray arr = obj.getJSONArray("Countries");
    String countriesJSON = arr.toString();
    List<Country> countries = mapper.readValue(countriesJSON, new
TypeReference<List<Country>>(){});
    Collections.sort(countries, new CoronaConfirmedComparator());

    return countries;
}
```

Код 4.30 – Вземане на текущото състояние по земи

```
public class CoronaConfirmedComparator implements Comparator<Country>
{
    @Override
    public int compare(Country a, Country b)
    {
        if(a.TotalConfirmed > b.TotalConfirmed)
        {
            return -1;
        }
        else
        {
            return 1;
        }
    }
}
```

Код 4.31 – Компаратор за сортиране по общата бройка заболяни

```
public class Country
{
    public int NewRecovered;
    public int NewDeaths;
    public int TotalRecovered;
    public int TotalConfirmed;
    public String Country;
    public String CountryCode;
    public String Slug;
    public int NewConfirmed;
    public int TotalDeaths;
    public String Date;
    public JsonNode Premium;
}
```

Код 4.32 – Класа за състоянието в земята

Последният метод който е имплементиран в CoronaInfo класа е getMacedoniaInfo. Целта на този метод е взимане на отделната информация за държавата Северна Македония, земя в която припада града Неготино. Тук държавата се взима по нейния индекс от листата с информации предоставена от API-то. Обаче, реализирана е допълнителна проверка за това дали наистина този индекс съответства на Македония защото реално е лесно променим от автора на API-то. При неточност се обхожда цялата листа от земи и се търси земята по countryCode("MK") и нейното име("macedonia"). Тази допълнителна проверка ни овъзможава да не сгрешим и да вземем само информацията която се отнася за Македония. Тук се връща обект от тип Country който всъщност представлява информацията заделена за земята която искаме.

```
public static Country getMacedoniaInfo(JSONObject obj, ObjectMapper mapper) throws
IOException
{
    Country country;
    JSONArray arr = obj.getJSONArray("Countries");
    JSONObject jsonObject = arr.getJSONObject(100);
    String countryCode = jsonObject.getString("CountryCode");
    String slug = jsonObject.getString("Slug");

    if(countryCode.equals("MK") && slug.equals("macedonia"))
    {
        country = mapper.readValue(jsonObject.toString(),
        new TypeReference<Country>(){});
        return country;
    }
    else
    {
        String countriesJSON = arr.toString();
        List<Country> countries = mapper.readValue(countriesJSON,
        new TypeReference<List<Country>>(){});
        for(Country currCountry : countries)
        {
            if(currCountry.CountryCode.equals("MK") &&
            currCountry.Slug.equals("macedonia"))
            {
                return currCountry;
            }
        }
    }
    return null;
}
```

Код 4.33 – Вземане на текущото състояние за Македония

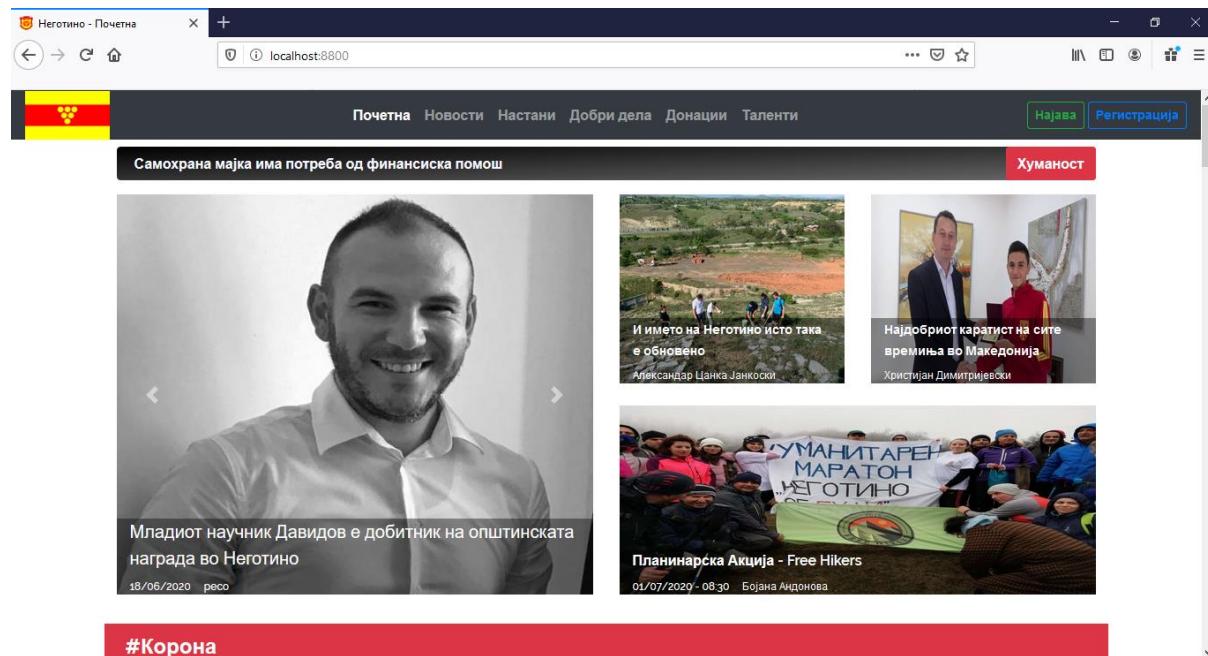
Извикването на всички тези методи става в контролера за началната страница. Там първо се създава обект от тип ObjectMapper, взема се респонса като JSON object с чиято помощ натам се взимат всички необходими информации за глобалната статистика и информация за държавата Македония.

```
ObjectMapper mapper = new ObjectMapper();
JSONObject obj = CoronaInfo.getApiResponse();
Global global = CoronaInfo.getGlobalInfo(obj, mapper);
List<Country> countries = CoronaInfo.getCountryInfo(obj, mapper);
Country mkd = CoronaInfo.getMacedoniaInfo(obj, mapper);
```

Код 4.34 – Извикване на методите от класа CoronaInfo

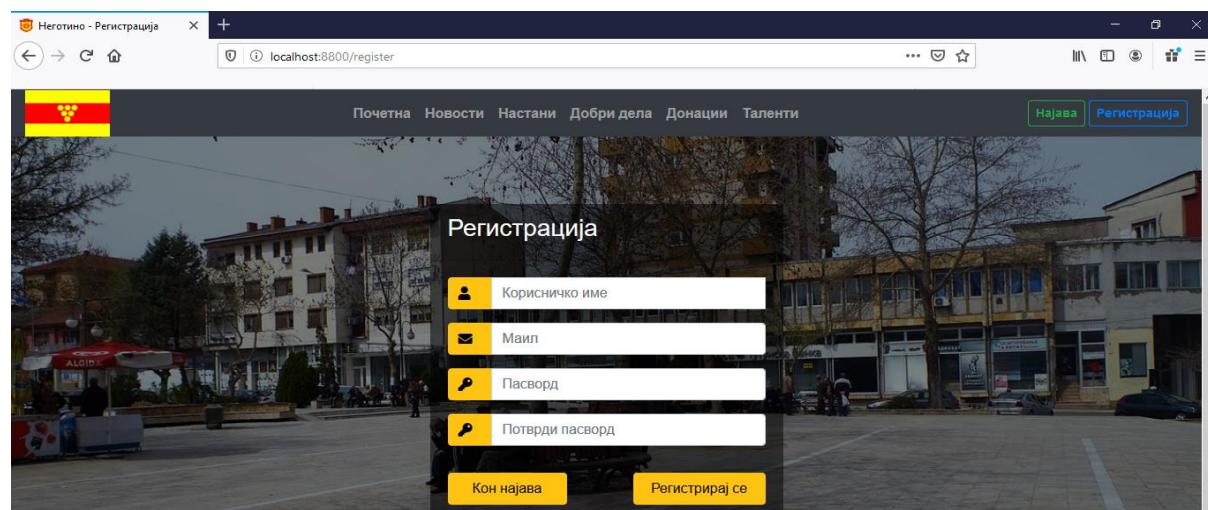
5. Ръководство на потребителя

Уеб приложението за града Неготино е достъпно за всеки един, без значение дали е логнат в сайта или не. Началната страница на сайта е показана на фигура 5.1. Цялата информация на сайта е описана на македонски език защото като цяло приложението е предназначено гражданините на града Неготино.

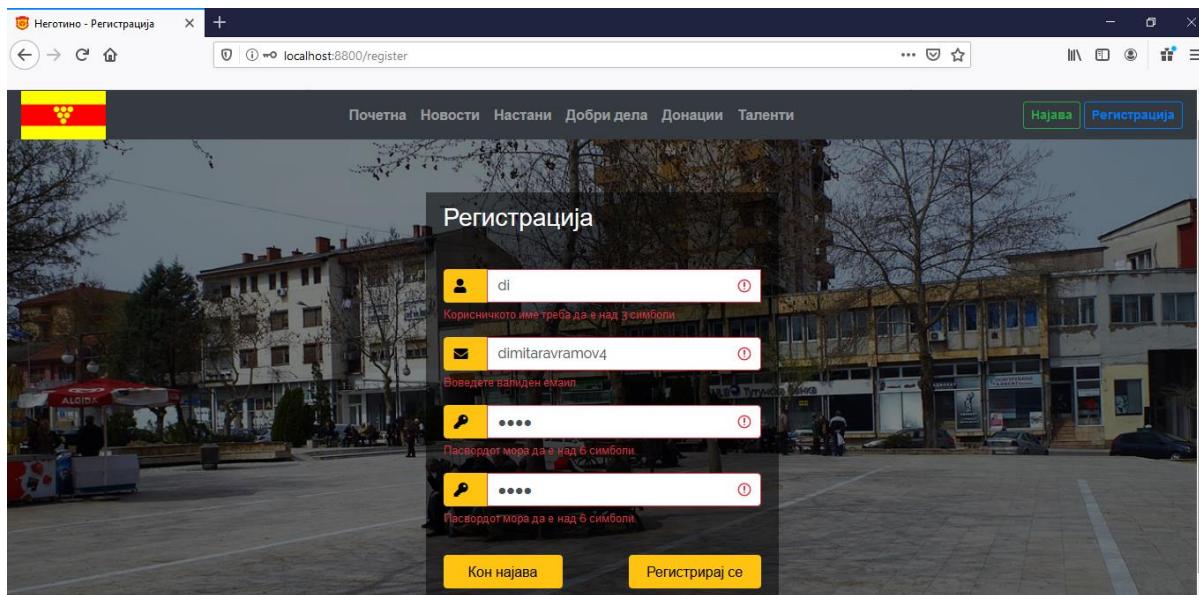


Фиг. 5.1 – Начална страница на уеб сайта

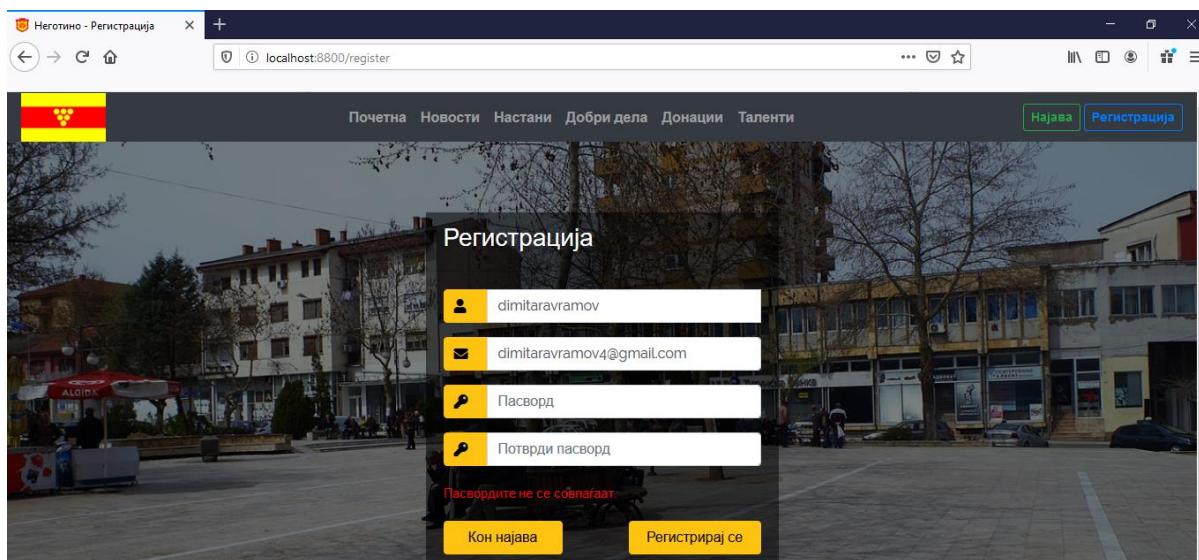
На фиг. 5.1 се вижда че в момента няма логнат потребител. При такова положение приложението е достъпно само за преглеждане и не позволява достъп към основната функционалност – споделяне на публикации от потребителите. За това ще описваме положенията при наличие на потребител за добре да изясним целта на тази дипломна работа. Съответно трябва да създадем свой акаунт, с помощ на бутона Регистрация.



Фиг. 5.2 – Регистрация на потребител



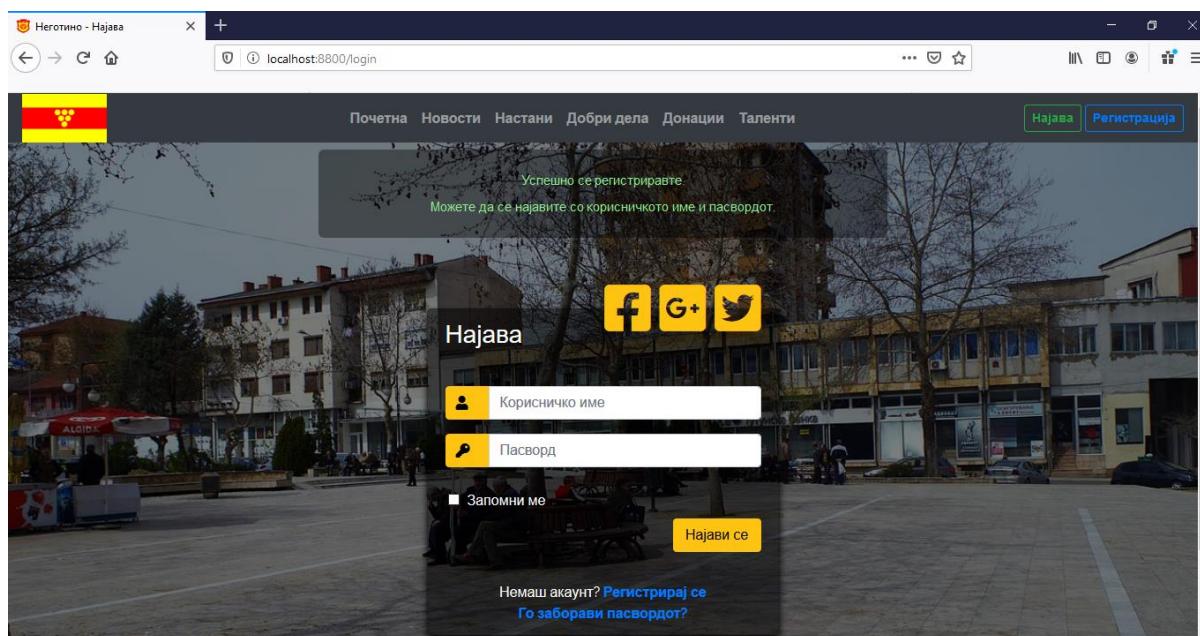
Фиг. 5.3 – Front-end валидация



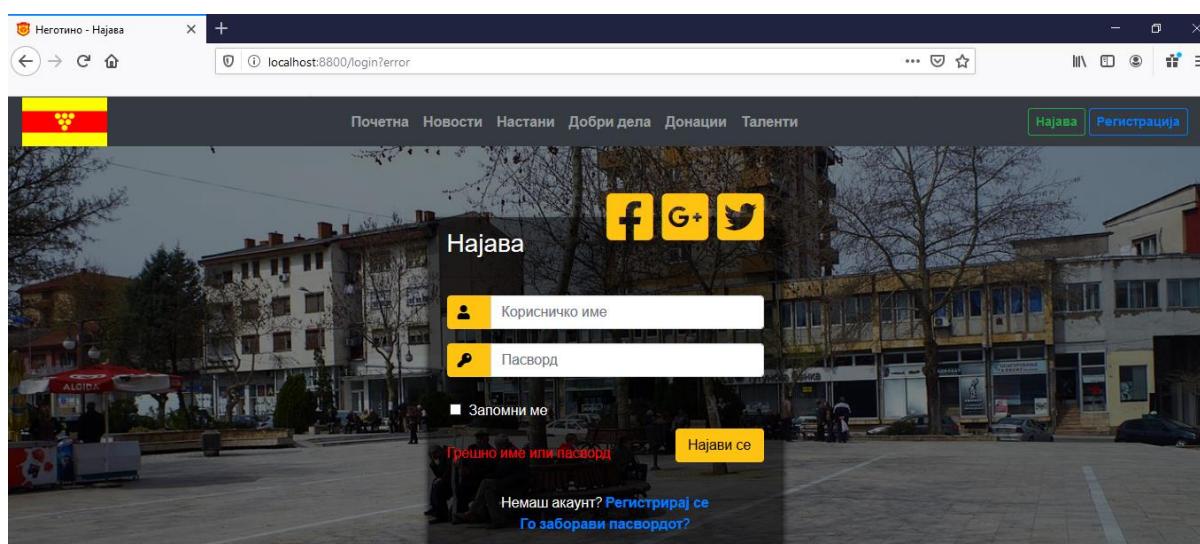
Фиг. 5.4 – Умишлен опит с несвъпадащи пароли, триене само на паролската част

Докато, вече на фиг. 5.5 вече успяхме да се регистрираме, което означава че имаме първия потребител в приложението. Тук идва реализацията на потребителската част където обясниме, че първия регистриран акаунт получава ролята ROOT-ADMIN или owner на приложението. Съответно, той притежава всички възможни права в сайта.

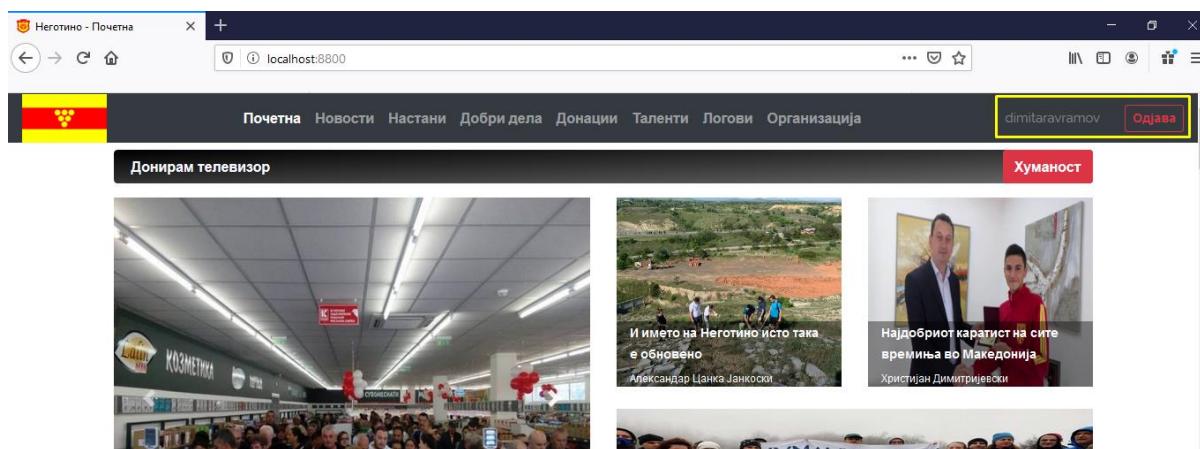
След успешното регистриране на фигура 5.5 е изписана информация че сме успешно регистрирани и вече може да влезнем в новия акаунт с помощ на потребителското име и паролата. Другия начин по който може да се логираме е с помощ на някоя от социалните мрежи. Там е реализирана OAuth 2.0 авторизация, като всичките автоматически приемат ролята на обикновен потребител.



Фиг. 5.5 – Успешна регистрация



Фиг. 5.6 – Валидација при логин



Фиг. 5.7 – Успешно логване (посочено с жълто потребителско име и възможност за изход)

Аналогично създаваме още 4 нови акаунти с потребителски имена: **Petar, moderator, tijana** и **Toshko**. Автоматично при всяка регистрация след първата както и при логването с помощ на трети страници(фейсбук, гугъл), акаунта притежава ролята на обикновен потребител. Тези роли са организирани от страна на админите. В случая единствения админ е dimitaravramov и с неговия акаунт може да достъпиме частта за организацията на потребителите показана на фигура 5.8.

Username	Role	Change Role
moderator	ROLE_USER	Change Role
Petar	ROLE_USER	Change Role
tijana	ROLE_USER	Change Role
Toshko	ROLE_USER	Change Role

Фиг. 5.8 – Организация на потребители

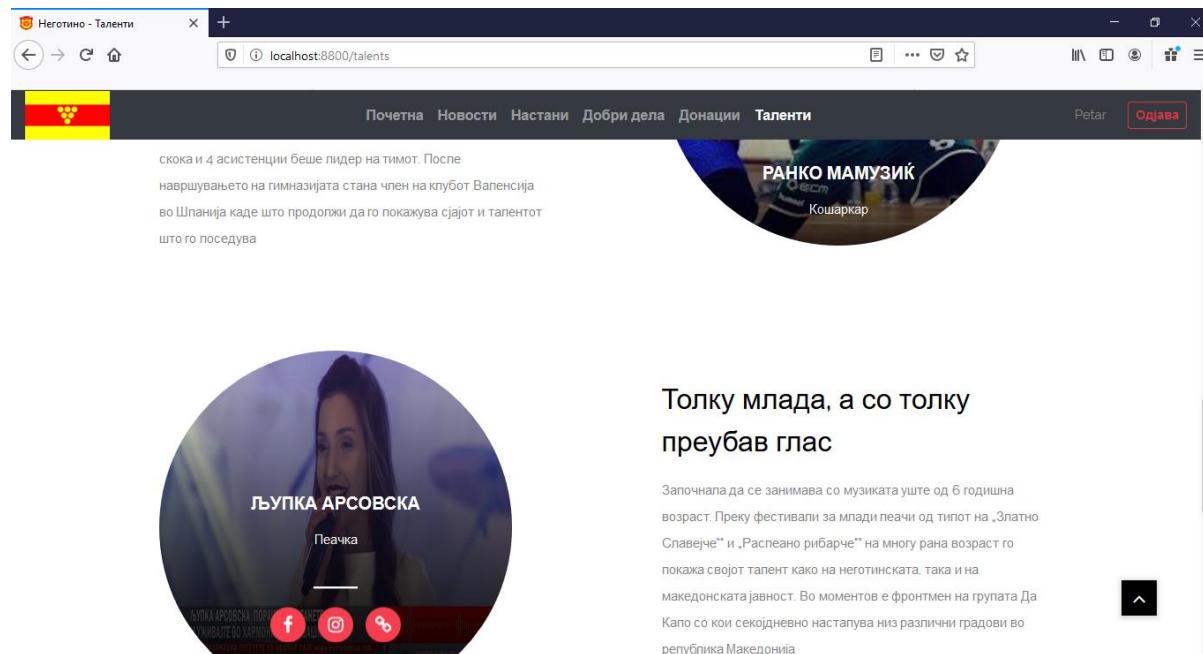
При назначаване на ролята модератор на акаунта с потребителското име moderator, ролята админ на потребителя tijana и триене на потребителя Toshko получаваме резултат показан на фигура 5.9.

Username	Role	Change Role
moderator	MODERATOR	Change Role
Petar	ROLE_USER	Change Role
tijana	ADMIN	Change Role

Фиг. 5.9– Промяна на роля и триене на потребител

Така, с помощ на тази функционалност за момента имаме точно 4 потребители от които съответно всеки притежава една от възможните роли (USER, MODERATOR, ADMIN, ROOT-ADMIN). За показване на възможностите на всичките хора логнати в сайта, които въобще представляват основните източници на информации ще се логнем с потребителя Petar – обикновен потребител.

При достъпване на секцията „Таленти“ ние можем да гледаме всички публикувани таланти, което е достъпно и за клиентите които не са логнати в сайта - фигура 5.10



Фиг. 5.10 – Преглеждане на секцията таланти

Обаче, всичките логнати потребители имат възможност за добавяне на талант с основните информации, които съдържа една публикация както и съответна снимка.

Наслов	
Име	Презиме
Дејност (Музичар, Спортсмен итн.)	
Сподели го твојот или талентот на некој познат. Објави ги способностите, достигнувачта, историјата, соновите и покажи дека Неготино има личности што претставуваат гордост на самиот град.	
Facebook	Instagram
Сподели	

Фиг. 5.11 – Добавяне на талант

Тази форма на споделяне притежава праста валидация за предпазване от празни полета показана на фигура 5.12. Обаче, е позволено да се сподели информацията без да се прикачи снимка за което се заделя default-на такава.

Европскиот и светски претставник на Македонија е тс ✓

Тони Презиме ⓘ
Воведи презиме

Дејност (Музичар, Спортст итн.) ⓘ
Воведи дејност

Сподели го твојот или талентот на некој познат. ⓘ
Објави ги способностите, достигнувањата, историјата, соновите и покажи дека Неготино има личности што претставуваат гордост на самот град.

Описи го талентот

Facebook ⓘ Instagram ⓘ

Сподели

ДОДАДИ ФОТОГРАФИЈА

Фиг. 5.12 – Валидация при добавянето на талант

Европскиот и светски претставник на Македонија е тс ✓

Тони Лазов ✓ Лазов ✓
Шахист ✓

Најдобриот шахист во република Македонија до 18 години е токму младиот Тони Лазов од Неготино. Пет пати ја претставувал нашата држава на европските првенства и два пати на светските. Уште како мал вејбайки секојдневно со својата стрина Тони се залубил во овој спорт. На само 8 години успеа да го освои државното првенство и стане еден од нај надежните таленти

<https://www.facebook> ✓ <https://www.instagram> ✓

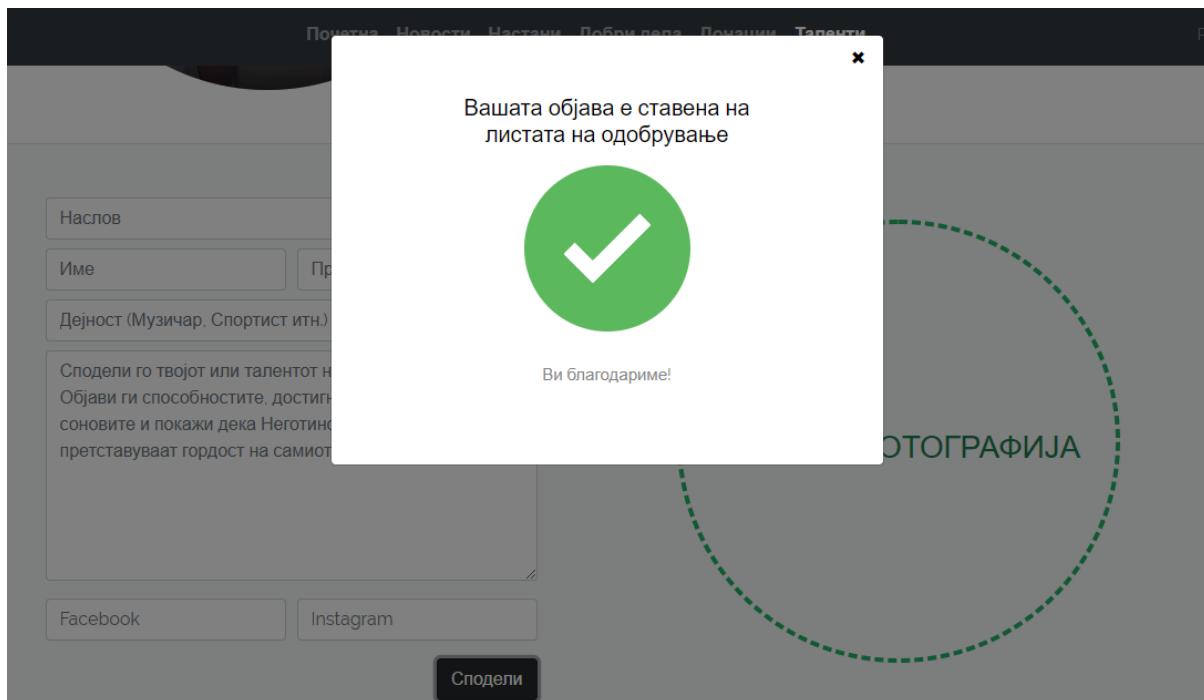
Сподели

REMOVE TONILAZOV.JPG

Фиг. 5.13 – Успешно попълване на формата

При въвеждане на формата и успешно споделяне на информацията на екрана се изписва съобщение за успех. Там писва, че нашия пост е из pratен за преглеждане и се очаква приемане/отхвърляне от модераторите. В същото време формата се ресетира и е вече достъпна за нова публикация.

Подобни действия се случват във всичките секции на сайта – донации(дарения), добри дела, настани(събития), новости(новини).



Фиг. 5.14 – Успешен пост

За да покажем как този пост ще дойде при модераторите ще се логнем с потребителя, на който му заделихме такава роля. На фигура 5.15 се вижда че модераторите имат същите права както и потребителите, но допълнително от долу му излизат request-натите таланти които съответно може да публикува или изтрие. На потребителите с роля модератор, админ, или главен админ допълнителните функционалности им са описани на английски език за лесно да може да се видят разликите относно обикновените потребители.

Европскиот и светски претставник на Македонија е токму од Неготино

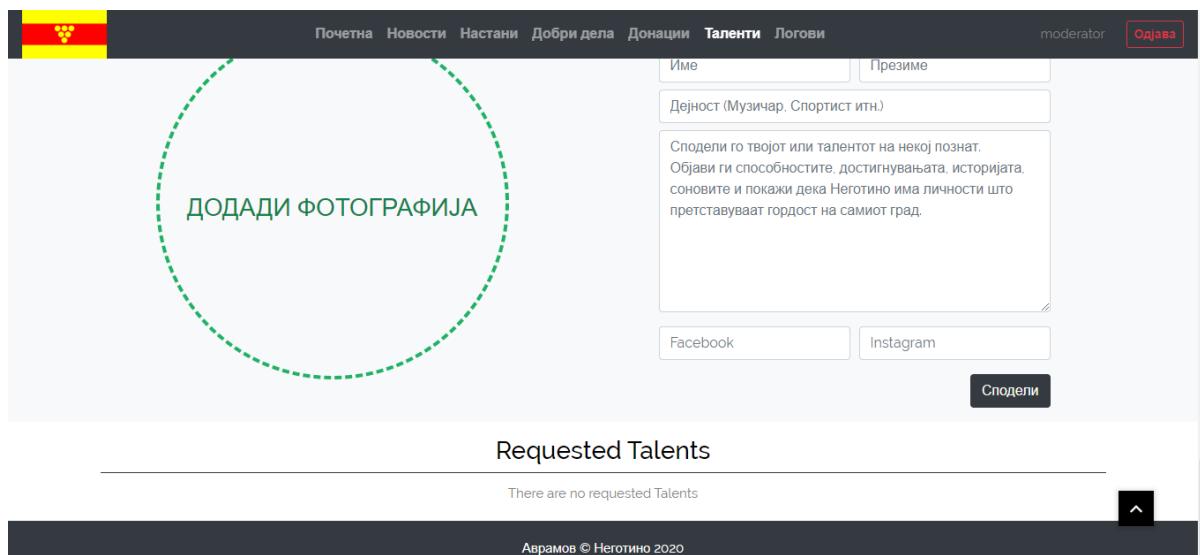
Најдобриот шахист во република Македонија до 18 години е токму младиот Тони Лазов од Неготино. Пет пати ја претставувал нашата држава на европските првенства и два пати на светските. Уште како мал вежбайќи секојдневно со својата стрина Тони се зарубил во овој спорт. На само 8 години успеа да го освои државното првенство и стане еден од најнадежни таленти на сите времиња во Македонија со освени вкупно 10 титули.

Approve Discard

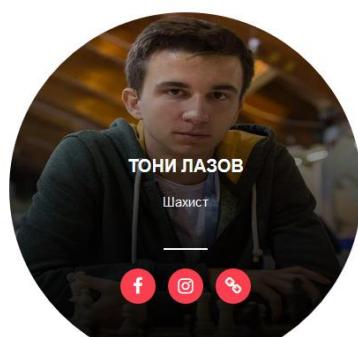
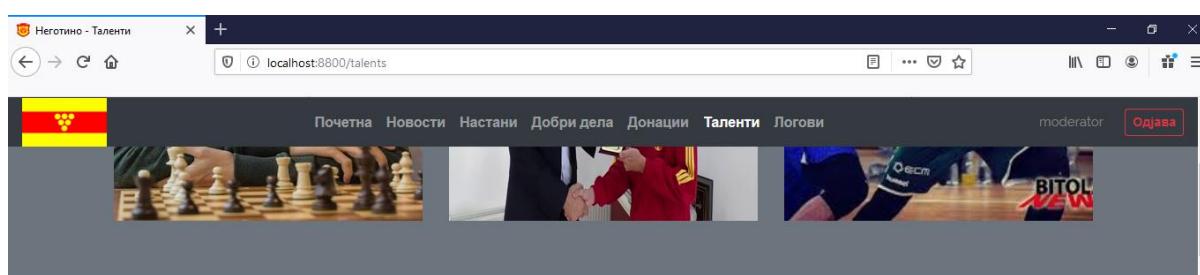
Approve All Discard All

Фиг. 5.15 – Хендълване на request-нати публикации

При натискане на Discard ще се отхвърли този пост изцяло и ще изчезне от това място (Фигура 5.16). Докато при натискане на Approve поста се публикува явно и излиза най-отгоре (Фигура 5.17)



Фиг. 5.16 – Discard на поста

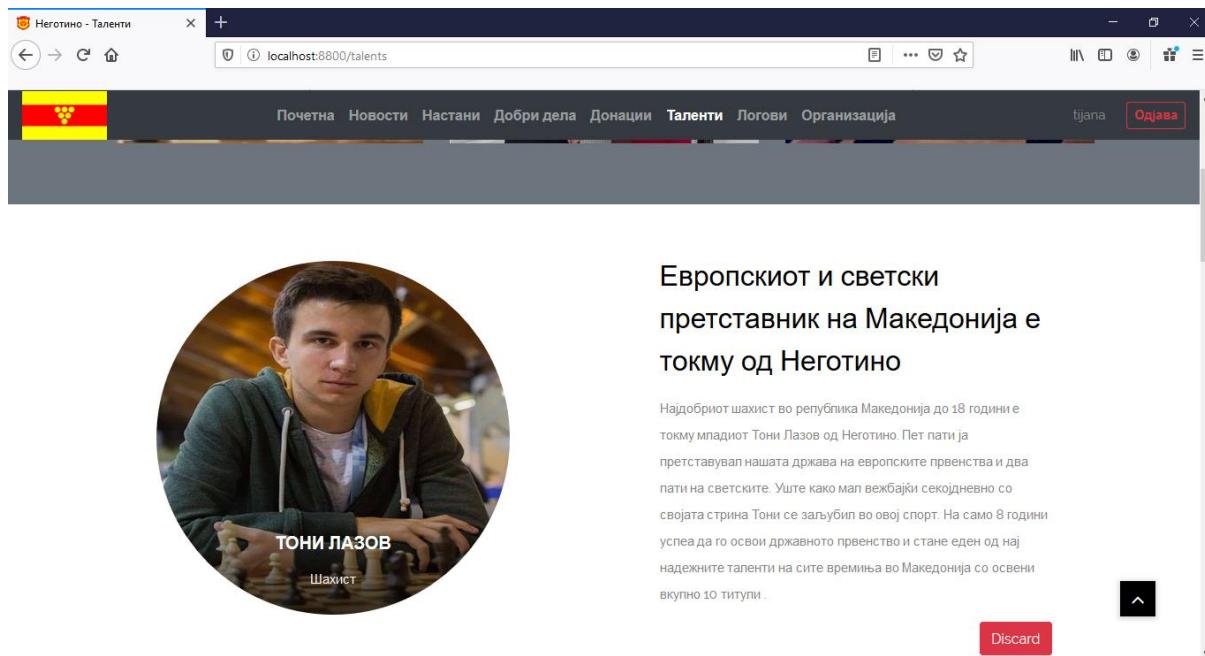


**Европскиот и светски
представник на Македонија е
токму од Неготино**

Најдобриот шахист во република Македонија до 18 години е токму младиот Тони Лазов од Неготино. Пет пати ја претставува нашата држава на европските првенства и два пати на светските. Уште како мал вежбајќи секојдневно со својата стрина Тони се запуѓбил во овој спорт. На само 8 години успеа да го освои државното првенство и стане еден од најнадежните таленти на сите времиња во Македонија со освени

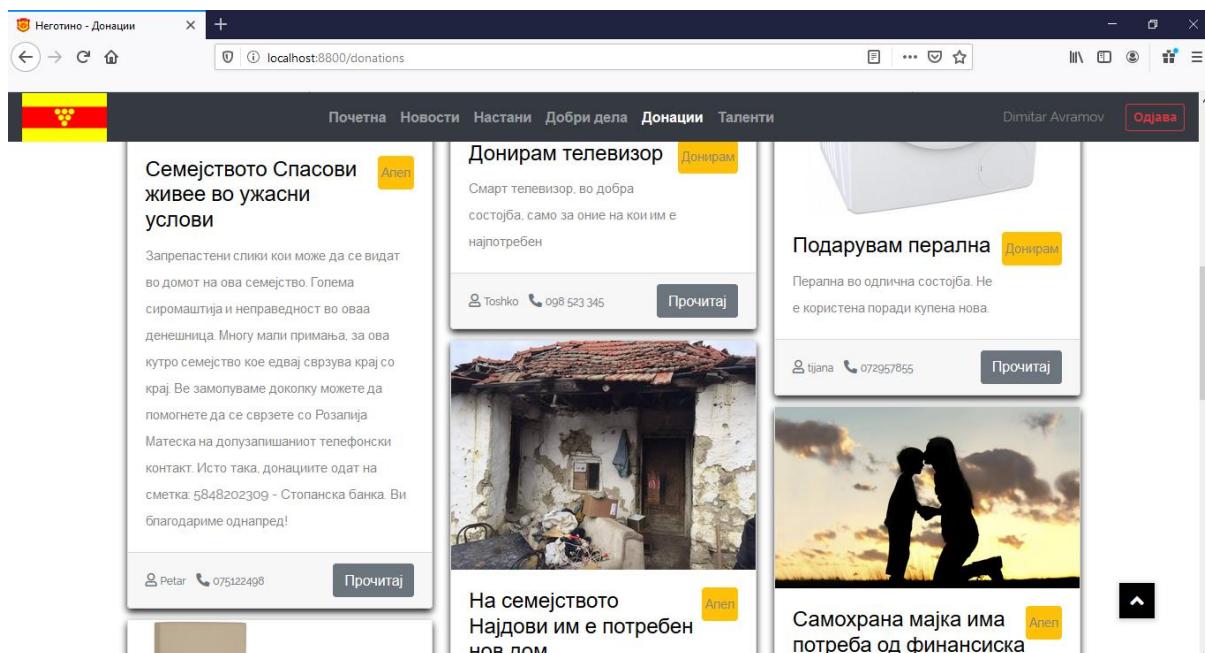
Фиг. 5.17 – Approve на поста и съответното публикуване

На ред идва да се логнем с потребителя `tijana` който притежава ролята `admin`. На фигура 5.18 вече се вижда разликата с която администраторите може да трият вече публикувани информации на сайта с подобно цъкане върху Discard бутона показан на самия пост. Този, както и принципа на модераторите е приложен при всички други секции.

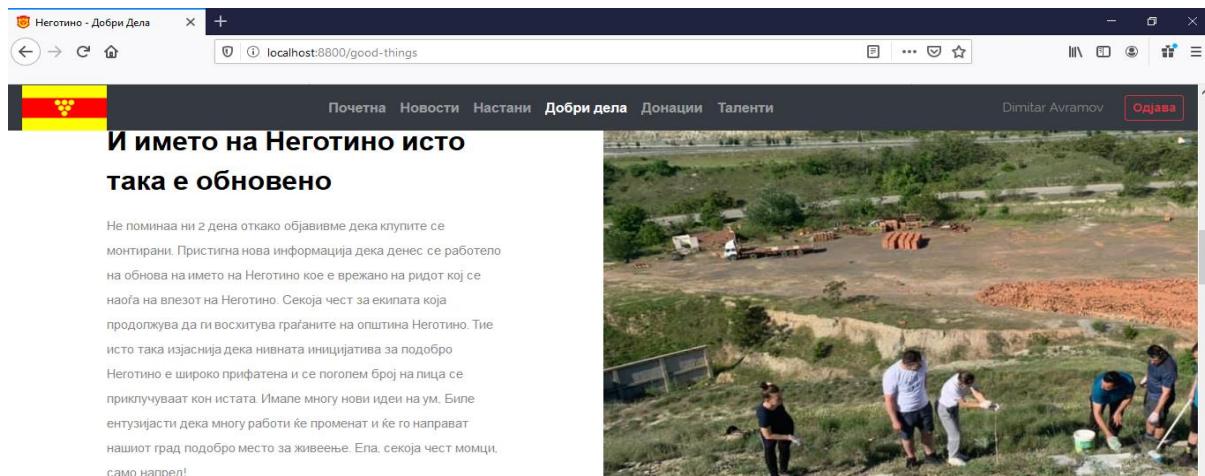


Фиг. 5.18 – Discard на публикуван пост (право на админа)

Сега остана да прегледаме част от изгледите на останалите секции които аналогично имплементират живота на постовете посочен в проектирането. Тези неща ще ги направим с логнат акаунт от Google за да докажем, че са достъпни и такива видове на автентификация.



Фиг. 5.19 – Секция Донации (Дарения)



Фиг. 5.20 – Секция Добри дела

Турнир во одбојка
Ми претставува огромна чест и задоволство да Ви го претставам 5-тиот јубилеен хуманитарен турнир во одбојка кој веќе е дел од новиот проект наречен Хуманитарен маратон - „Неготино се буди“. Повторно, сите заедно на едно исто место преку дружење, радост и лъбубов да продолжиме ги создаваме оние мали, но сепак толку убави мигови.

Театарска Претстава „Умри Машки“
Представата ќе се одржи на ден Четвртток (07.02.2019г.) од 19 часот во центарот на културата „Ацо Ѓорчев“ во Неготино.

Фиг. 5.21 – Секция Настани (Събития)

Во Белград Лидл, во Неготино КАМ манија: Битка за евтини
Неготино, градот каде Тоше ги заработи своите први 100 евра

Барај

Младиот научник Давидов е добитник на општинската награда во Неготино
18/06/2020 | [реко](#)

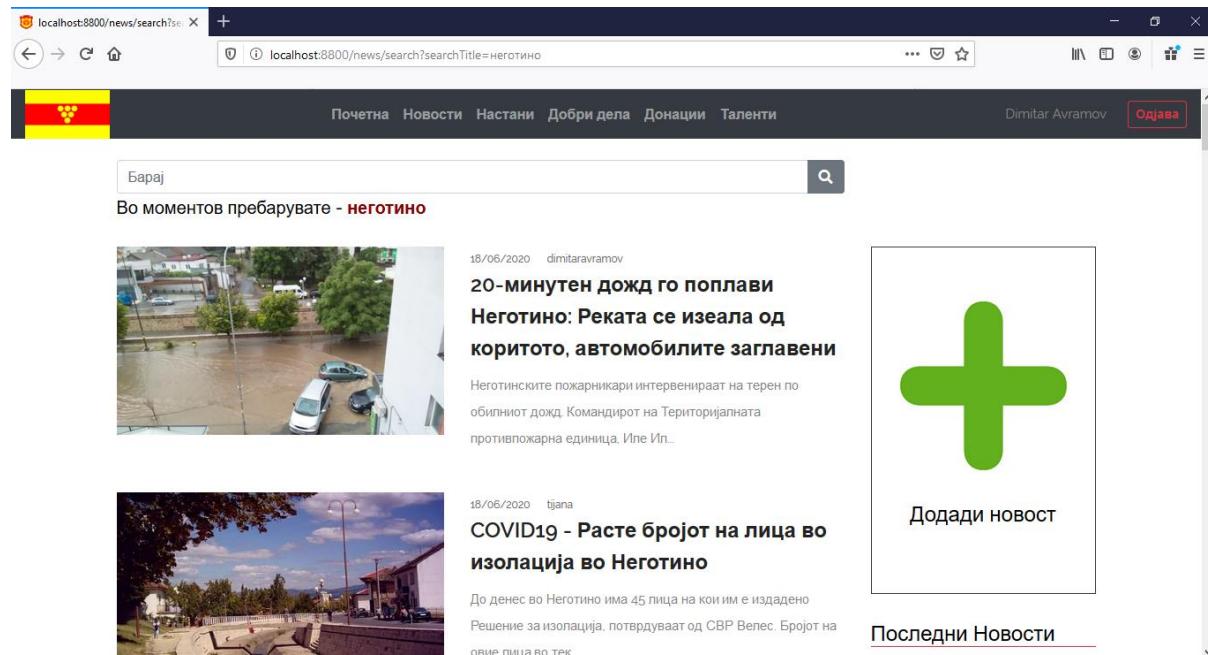
Тиквешките лозари ќе протестираат со земјоделска механизација во Неготино
18/06/2020 | [тјана](#)

Ограбил казино во Неготино, мајка му го пријавила во полиција!
18/06/2020 | [реко](#)

Стiven Gerrard ќе купува фудбалер од Неготино
18/06/2020 | [dimitaravramov](#)

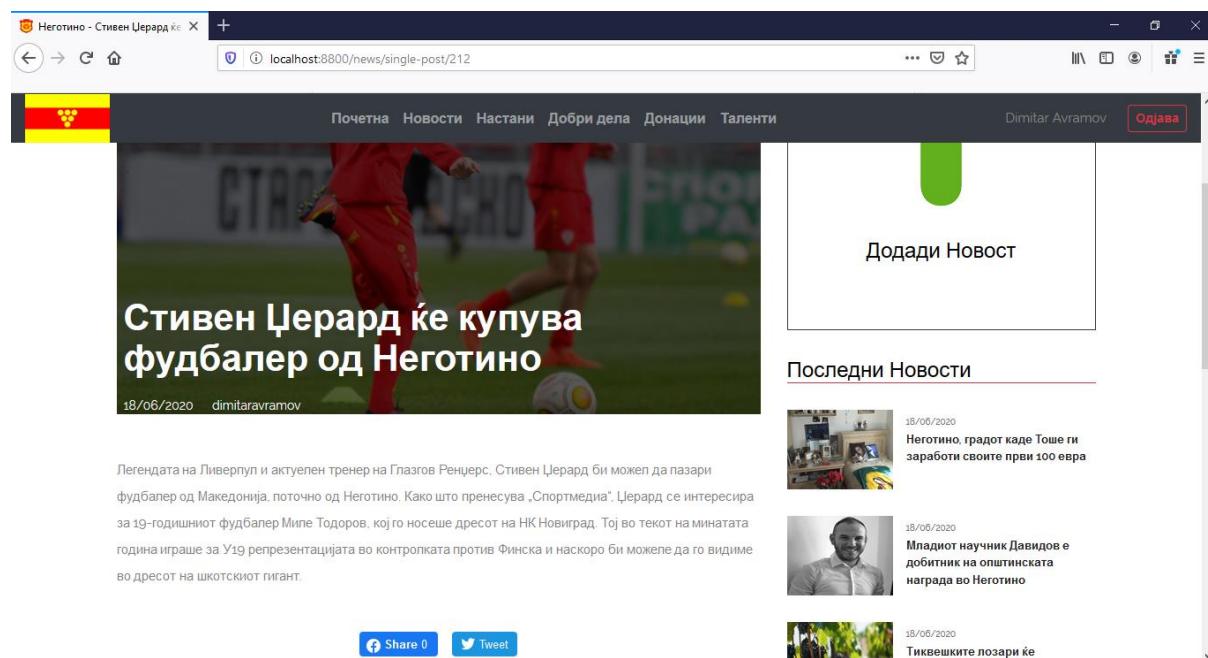
Фиг. 5.22 – Секция Новости (Новини)

На фиг 5.23 е показана функционалността за търсене на новина по ключова дума.



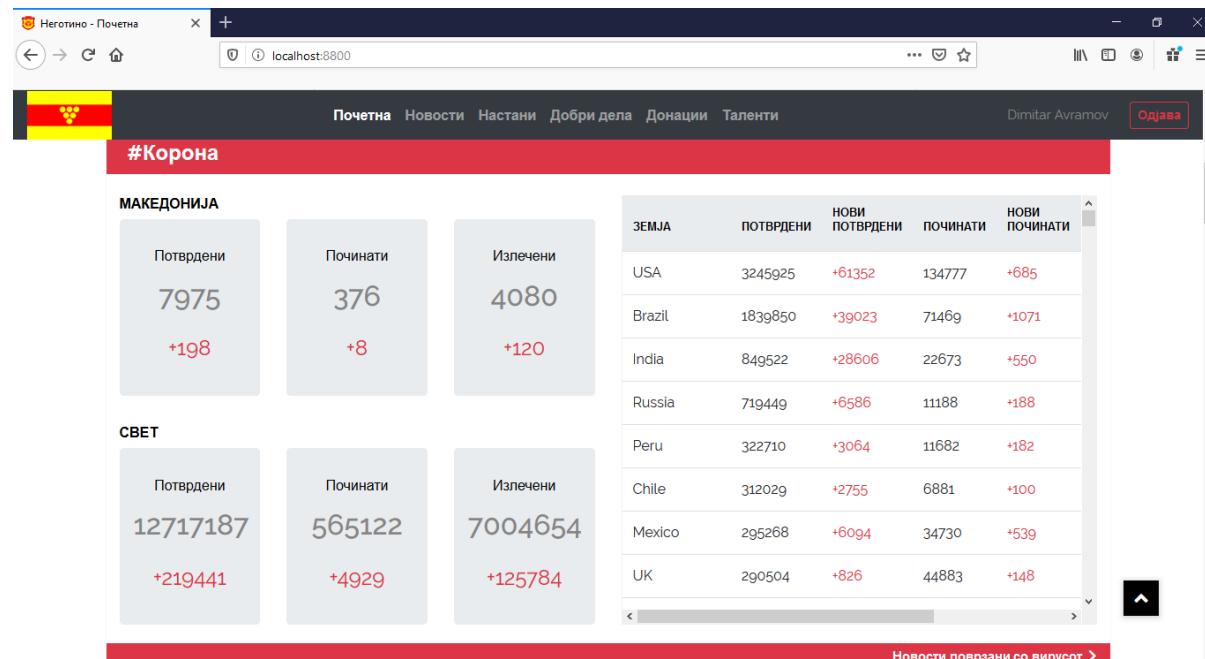
Фиг. 5.23 – Търсене на новина по ключова дума

За хубаво преглеждане на целия текст във всичките секции е имплементирана възможност с която с цъкане върху заглавието на публикацията, сайта не носи към определен изглед който се съдържа само от посочения пост. Там имаме възможност като отделен линк да шернем публикацията директно в Facebook или Twitter. На фигура 5.24 е показано как може да достъпиме текста то една новина и при това да имаме възможността да я споделим в социалните мрежи.

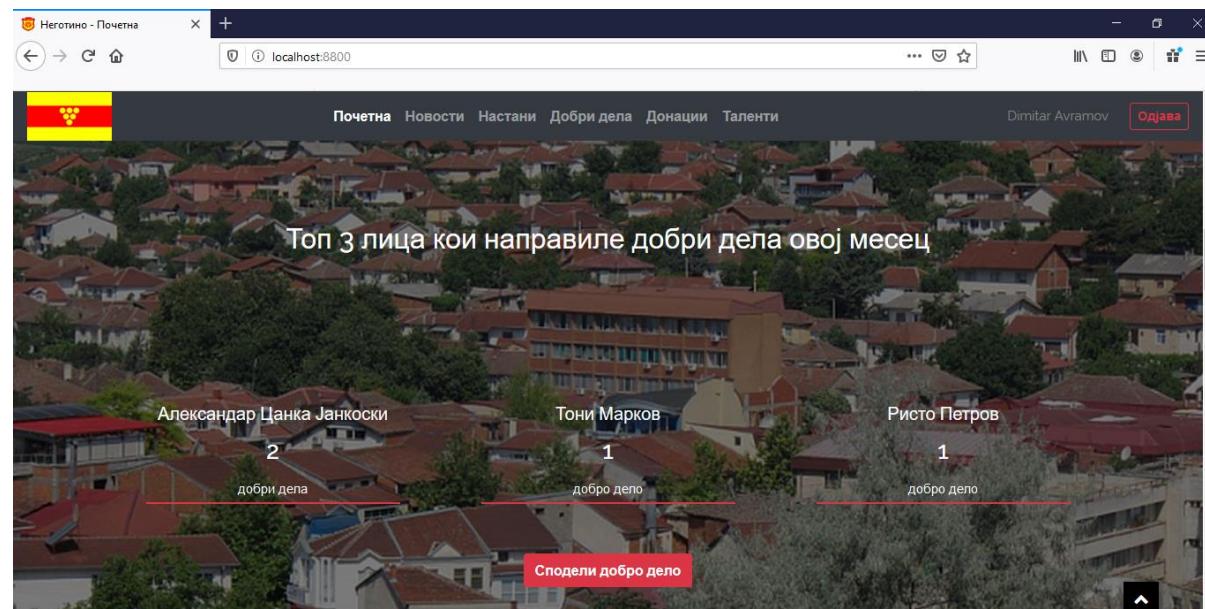


Фиг. 5.24 – Отделен пост от разните секции

При достъпване на началната страница може да видим информация относно корона вируса както низ света, така и в държавата в която се намира Неготино.

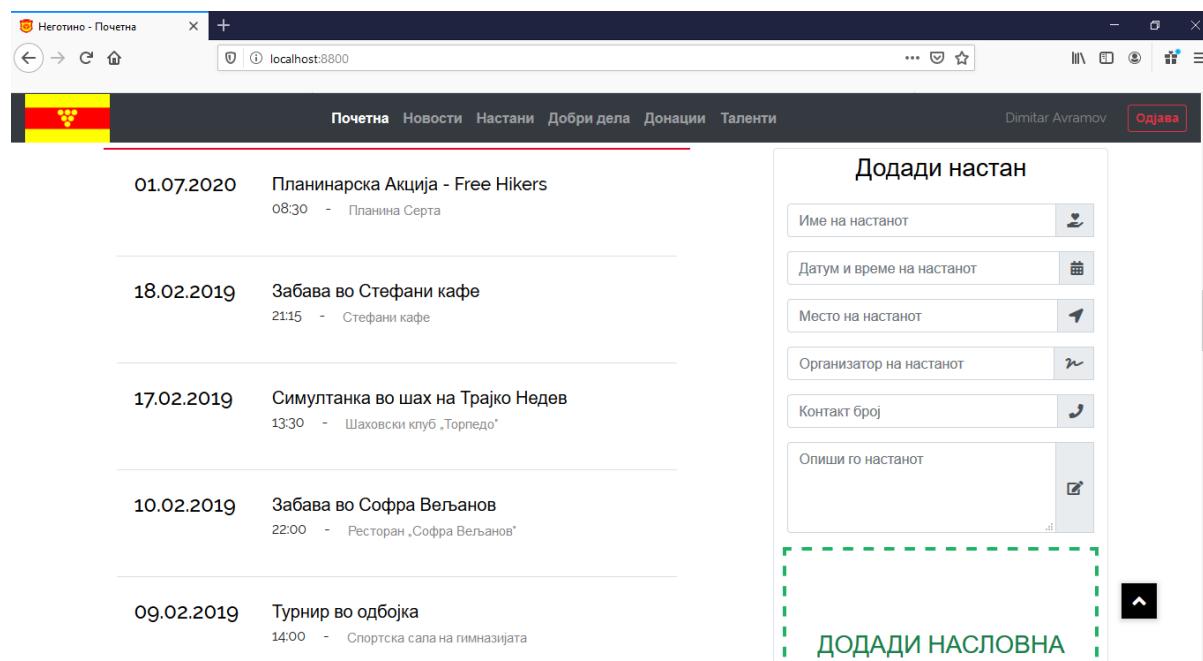


Фиг. 5.25 – Информация относно корона вируса

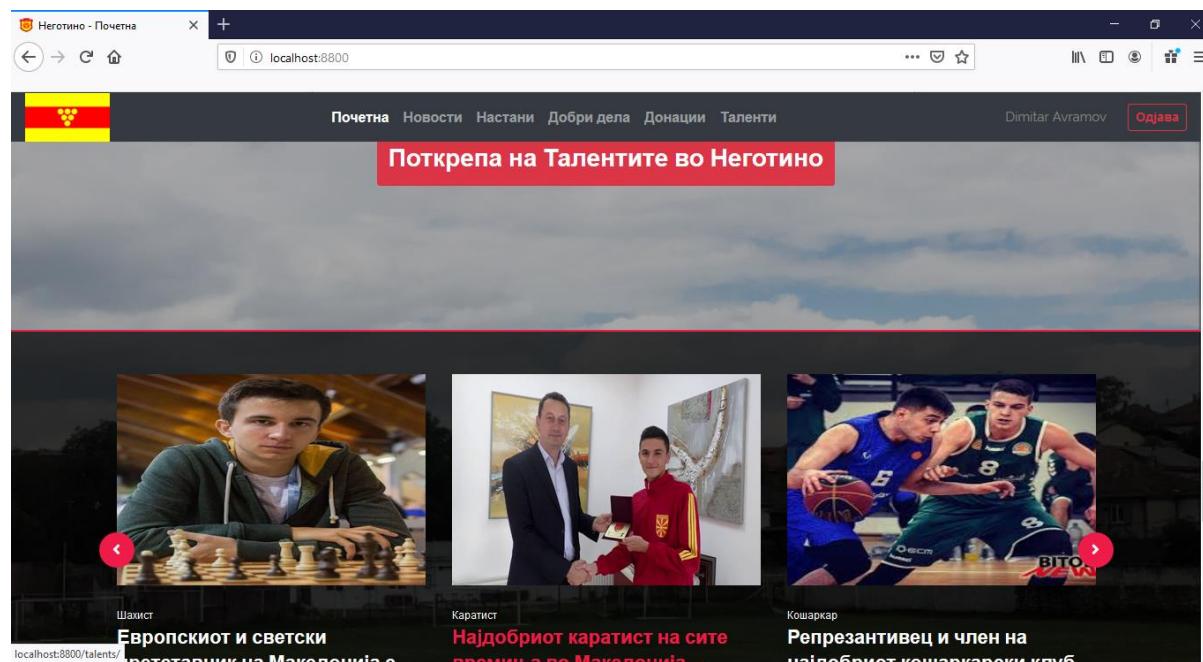


Фиг. 5.26 – Топ 3 автори на добри дела

На началната страница можем да видим съобразение от публикации от всички секции. Една част е да се видят хората които са направили най-много добри дела за Неготино на фигура 5.26. Подобни изгледи са изкарани на фигураните 5.27 и 5.28.



Фиг. 5.27 – Листнати настани (събития)



Фиг. 5.28 – Подкрепа на талантите

Последната секция която ще разгледаме е достъпна при минимум роля модератор и е наречена „Логове“. Това е частта в която са листнати всички потребителски истории. Особеното тук е, че само модераторите може да преглеждат логовете, докато само owner-а може да тряб определени или всички истории. За тази цел ще се логнем с най-първо регистрирания акаунт – dimitaravramov. За сега сме правили само 2 пъти добавяне от потребител Petar в Таланти и съответно един път отхвърляне и един път прихващане от страна на потребителя с име moderator. Точно това си стои в момента в логовете, показано на фигура (5.29).

Username	Operation	Modified table	Time
Petar	Add	Talents	12/07/2020 17:35:44
moderator	Discard	Talents	12/07/2020 17:38:28
Petar	Add	Talents	12/07/2020 17:41:34
moderator	Approve	Talents	12/07/2020 17:45:43
Username	Operation	Modified table	Time

Showing 1 to 4 of 4 entries

Clear selected Clear all logs

Фиг. 5.29 – Потребителски истории

Тук може да сортираме по всичките колони което ни овъзможава да определяме логовете низ времето, най-активни потребители, най-често ползвани операции и най-популярни секции. Също така може да търсим по ключова дума отново по било коя колона за да намерим информация от за определен потребител, секция, операция или време.

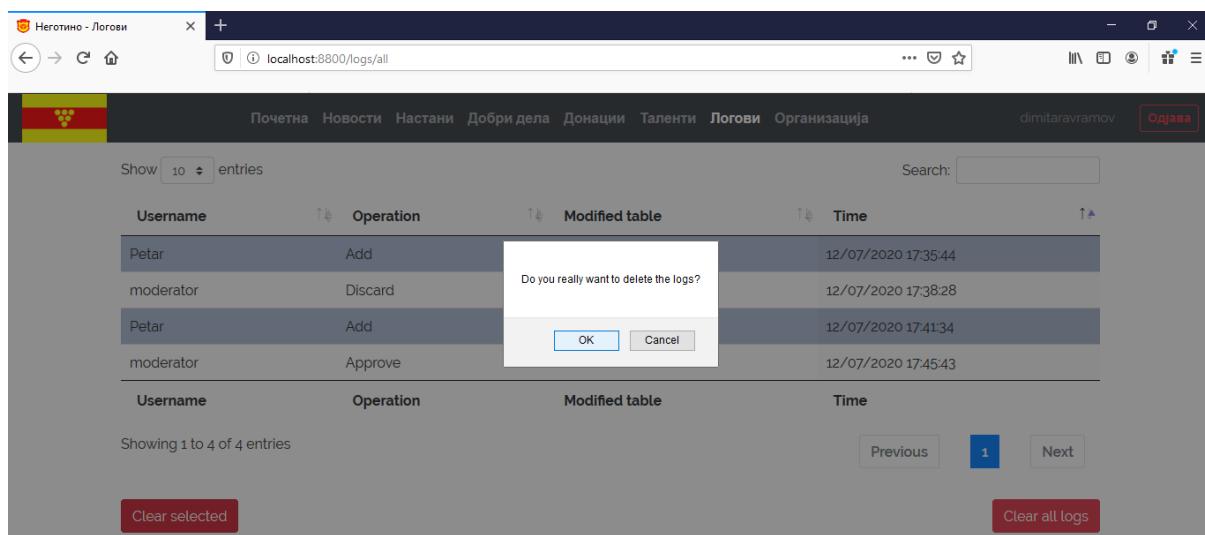
Username	Operation	Modified table	Time
moderator	Discard	Talents	12/07/2020 17:38:28
moderator	Approve	Talents	12/07/2020 17:45:43
Username	Operation	Modified table	Time

Showing 1 to 2 of 2 entries (filtered from 4 total entries)

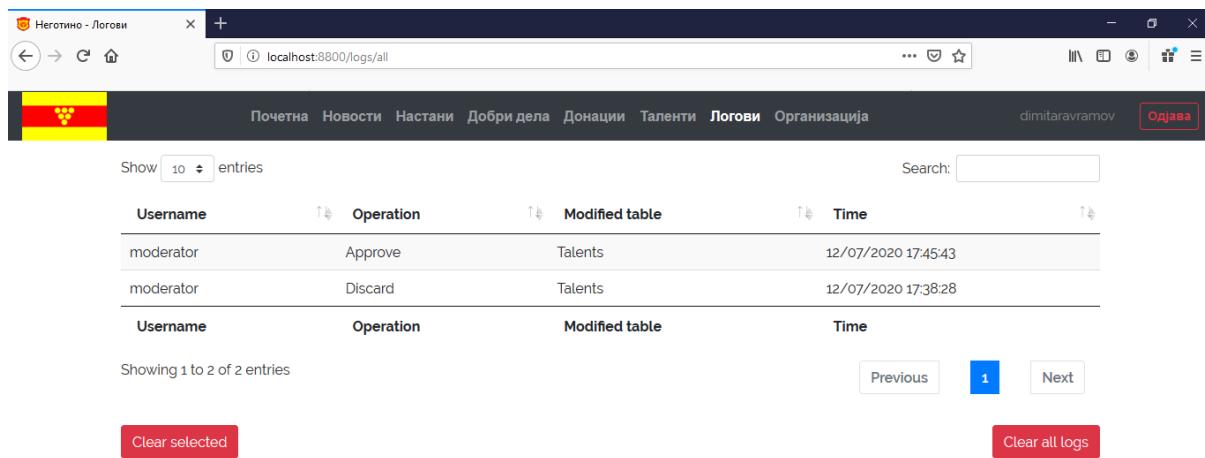
Clear selected Clear all logs

Фиг. 5.30 – Търсене по ключова дума

Както посочихме в логовете owner-а притежава права и да изтрие определени или всичките логове. На фигура 5.31 се вижда как се опитваме да изтрием логовете в които потребителя Petar е добавял публикация в секцията за таланти. При това системата не пита за това дали сме сигурни, и при потвърждаване резултата е видим на фигура 5.32.



Фиг. 5.31 – Опит за триене на определен лог



Фиг. 5.32 – Успешно триене на определените логове

Заключение

Уеб приложението представлява платформа, която има за цел да подобри вътрешното развитие на града Неготино. Основната идея на тази реализация се базира на участието на гражданите. Те имат достъп до възможностите на сайта след **регистрация** или логване с помощ на **други социални мрежи**. **Автентификацията** става чрез вписване на потребителското име и парола. За целите на **авторизацията** са създадени 4 на брой възможни роли: обикновен потребител, модератор, админ, главен админ (собственик), които са йерархично проектирани, с идеята всеки да има неговата роля и ролите под него (наследяване). Първия регистриран потребител е притежателя на всичките права т.е собственика на приложението. Всеки следващ регистриран потребител получава ролята на обикновен потребител, като тя може да му бъде променена от страна на собственика чрез потребителския интерфейс в частта **Организация**. Като обикновен потребител клиента има възможност да добавя публикации в отделните секции, които са: **таланти, дарения, добри дела, събития, новини**. Неговите публикации се изпращат за проверка към модераторите, които преглеждат и съответно публикуват или отхвърлят поста. Админите имат допълнителна възможност да трият вече публикувани постове. Всяка публикация в сайта за Неготино притежава съответен линк, където може да се прочете пълния и текст и при желание да се сподели във Facebook и Twitter. Възможностите за споделяне дават по-широка публичност на сайта. Реализирана е възможност за **търсене по ключова дума в новините**, преглеждане на **топ 3 автори на добри дела и най-новите постове** във всички секции. Също така приложението доставя информация относно текущото състояние на **корона вируса** както глобално, така и в различните държави. Направена е и възможност за преглеждане на **потребителските истории**, достъпна за потребители с роля модератор. Там може да се види история кой потребител какво е направил, коя секция е най-популярна, в кое време сайта е най-посещаван и т.н. Също така, за админите е достъпна частта за **организация**, където може да се промени роля на определен потребител или да се изtrie потребител от системата.

Към момента уеб приложението е готово за употреба, като са реализирани поставените цели и задачи. То среща нуждите и дава пълната възможност на нашите потребители да споделят информация за града и хората в него, имайки свободата явно да изказват тяхната визия. Приложението е конкурентноспособно на съществуващите решения, които са по-редки в тази специфична област. Част от бъдещото развитие на продукта включва: подобряване на потребителския интерфейс, възможност за споделяне на повече снимки, разширяване на областите, които обхваща, взаимодействие помежду потребителите, добавяне на коментари при публикациите, харесване на постове, споделяне на най-активни потребители и други.

Литература

- [1] Блогът на Явор Томов и Даниел Джолев, MySql и Java, <http://javac.bg/>
- [2] What is Framework, <https://gbksoft.com/blog/what-is-framework/>
- [3] What is Framework, <https://hackr.io/blog/what-is-frameworks>
- [4] Spring Overview, https://www.tutorialspoint.com/spring/spring_overview.htm
- [5] Java programming, http://www.tu-varna.bg/tu_varnaknt/images/learning/tutorials/OOP2/lectures/oop2_01.pdf
- [6] Java Tutorial, <https://www.tutorialspoint.com/java/index.htm>
- [7] Garbage Collector, <https://stackify.com/what-is-java-garbage-collection/>
- [8] MVC Tutorial, <https://www.guru99.com/mvc-tutorial.html>
- [9] Thymeleaf documentation, <https://www.thymeleaf.org/>
- [10] Tomcat documentation, <http://tomcat.apache.org/>

Приложения

1.Entities

Donation.java

```

@Entity
@Table(name = "donations")
public class Donation
{
    @Id
    @GeneratedValue
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private int id;
    @Column(name = "type")
    private DonationType type;
    @Column(name = "title")
    private String title;
    @Column(name = "contact")
    private String contact;
    @Column(name = "text", length = 14000)
    private String text;
    @Column(name = "author") // = username
    private String author;
    @Column(name = "img_name")
    private String imgName;
    @Column(name = "img_type")
    private String imgType;
    @Lob
    @Column(name = "img_data")
    private byte[] imgData;
    @Column(name = "approved")
    private boolean approved;
    public Donation() { }
    public int getId() { return this.id; }
    public void setId(int id){ this.id = id; }
    public DonationType getType() {return type; }
    public void setType(DonationType type) {this.type = type;}
    public String getTitle() {return title; }
    public void setTitle(String title) {this.title = title; }
    public String getContact() {return contact; }
    public void setContact(String contact) {this.contact = contact; }
    public String getText(){return text;}
    public void setText(String text){ this.text = text; }
    public boolean isApproved(){ return approved; }
    public void setApproved( boolean approved){this.approved = approved; }
    public String getAuthor(){ return author; }
    public void setAuthor(String author){ this.author = author; }
    public String getImgName(){ return imgName; }
    public void setImgName(String imgName){ this.imgName = imgName; }
    public String getImgType() { return imgType; }
    public void setImgType(String imgType){ this.imgType = imgType; }
    public byte[] getImgData(){ return imgData; }
    public void setImgData(byte[] imgData){ this.imgData = imgData; }
}

```

DonationType.java

```

public enum DonationType
{
    Апел,
    Донираам
}

```

Event.java

```

@Entity
@Table(name = "events")

```

```

public class Event
{
    @Id
    @GeneratedValue
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private int id;
    @Column(name = "name")
    private String name;
    @Column(name = "date")
    private Date date;
    @Column(name = "location")
    private String location;
    @Column(name = "organizer")
    private String organizer;
    @Column(name = "contact")
    private String contact;
    @Column(name = "text", length = 14000)
    private String text;
    @Column(name = "approved")
    private boolean approved;
    @Column(name = "img_name")
    private String imgName;
    @Column(name = "img_type")
    private String imgType;
    @Lob
    @Column(name = "img_data")
    private byte[] imgData;
    public int getId() { return this.id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
    public String getLocation() { return location; }
    public void setLocation(String location) { this.location = location; }
    public String getOrganizer() { return organizer; }
    public void setOrganizer(String organizer) { this.organizer = organizer; }
    public String getContact() { return contact; }
    public void setContact(String contact) { this.contact = contact; }
    public String getText() { return text; }
    public void setText(String text) { this.text = text; }
    public boolean isApproved() { return approved; }
    public void setApproved(boolean approved) { this.approved = approved; }
    public String getImgName() { return imgName; }
    public void setImgName(String imgName) { this.imgName = imgName; }
    public String getImgType() { return imgType; }
    public void setImgType(String imgType) { this.imgType = imgType; }
    public byte[] getImgData() { return imgData; }
    public void setImgData(byte[] imgData) { this.imgData = imgData; }
    public Event() { }
}
GoodThing.java
@Entity
@Table(name = "good_things")
public class GoodThing
{
    @Id
    @GeneratedValue
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private int id;
    @Column(name = "title")
    private String title;
    @Column(name = "text", length = 14000)
    private String text;
    @Column(name = "author")

```

```

private String author;
@Column(name = "approved")
private boolean approved;
@Column(name = "img_name")
private String imgName;
@Column(name = "img_type")
private String imgType;
@Lob
@Column(name = "img_data")
private byte[] imgData;
public String getImgName() { return imgName; }
public void setImgName(String imgName) { this.imgName = imgName; }
public String getImgType() { return imgType; }
public void setImgType(String imgType) { this.imgType = imgType; }
public byte[] getImgData() { return imgData; }
public void setImgData(byte[] imgData) { this.imgData = imgData; }
public int getId() { return this.id; }
public void setId(int id) { this.id = id; }
public boolean isApproved() { return approved; }
public void setApproved(boolean approved) { this.approved = approved; }
public String getTitle() { return title; }
public void setTitle(String title) { this.title = title; }
public String getAuthor() { return author; }
public void setAuthor(String author) { this.author = author; }
public String getText() { return text; }
public void setText(String text) { this.text = text; }
}

Log.java
@Entity
@Table(name="logs")
public class Log
{
    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    @Column(name = "id", nullable = false, unique = true, updatable = false)
    private String id;
    @Column(name="user")
    private String user;
    @Column(name="operation")
    @Enumerated(EnumType.STRING)
    private Operation operation;
    @Column(name="tableName")
    private String tableName;
    @Column(name="date")
    private String date;
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getUser() { return user; }
    public void setUser(String user) { this.user = user; }
    public Operation getOperation() { return operation; }
    public void setOperation(Operation operation) { this.operation = operation; }
    public String getTableName() { return tableName; }
    public void setTableName(String tableName) { this.tableName = tableName; }
    public String getDate() { return date; }
    public void setDate(String date) { this.date = date; }
    @Override
    public String toString() { return "Log{" + "id='" + id + '\'' + ", user='" + user +
    '\'' + ", operation=" + operation + ", tableName='" + tableName + '\'' + ", date='" + date
    + '\'' + '}'; }
}

NewsType.java
public enum NewsType
{

```

```

Актуелно,
Култура,
Историја,
Земјоделие,
Спорт,
Здравје,
Македонија
}

```

Operation.java

```

public enum Operation
{
    Add,
    Approve,
    Discard,
    DiscardApproved,
    ApproveAll,
    DiscardAll,
    Edit,
    Delete
}

```

Country.java

```

public class Country
{
    public int NewRecovered;
    public int NewDeaths;
    public int TotalRecovered;
    public int TotalConfirmed;
    public String Country;
    public String CountryCode;
    public String Slug;
    public int NewConfirmed;
    public int TotalDeaths;
    public String Date;
    public JsonNode Premium;
}

```

Global.java

```

public class Global
{
    public int NewConfirmed;
    public int TotalConfirmed;
    public int NewDeaths;
    public int TotalDeaths;
    public int NewRecovered;
    public int TotalRecovered;
}

```

2. User Model**UserRegisterBindingModel.java**

```

public class UserRegisterBindingModel
{
    private String username;
    private String password;
    private String confirmPassword;
    private String email;
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public UserRegisterBindingModel() { }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getPassword()
    public void setPassword(String password) { this.password = password; }
    public String getConfirmPassword() { return confirmPassword; }
    public void setConfirmPassword(String confirmPassword) { this.confirmPassword =
confirmPassword; }
}

```

3. UserRole Factory

UserRoleFactory.java

```
public final class UserRoleFactory
{
    @Autowired
    public UserRoleFactory() { }
    public final UserRole createUserRole(String authority)
    {
        UserRole userRole = new UserRole();
        userRole.setAuthority(authority);
        return userRole;
    }
}
```

4. Repositories

DonationRepository.java

```
public interface DonationRepository extends JpaRepository<Donation, Integer>
{
    List<Donation> findAllByApprovedFalse();
    List<Donation> findAllByApprovedTrue();
    List<Donation> findTop5ByApprovedTrueOrderByIdDesc();
}
```

EventRepository.java

```
public interface EventRepository extends JpaRepository<Event, Integer>
{
    List<Event> findAllByApprovedFalse();
    List<Event> findAllByApprovedTrue();
    List<Event> findTop5ByApprovedTrueOrderByIdDesc();
    List<Event> findTop6ByApprovedTrueOrderByIdDesc();
}
```

GoodThingRepository.java

```
public interface GoodThingRepository extends JpaRepository<GoodThing, Integer>
{
    List<GoodThing> findAllByApprovedFalse();
    List<GoodThing> findAllByApprovedTrue();
    List<GoodThing> findTop10ByApprovedTrueOrderByIdDesc();
    @Query("SELECT u.author, count(*) as v FROM GoodThing u WHERE u.approved=1 group by u.author order by v DESC")
    List<Object[]> getTopThreeAuthors();
}
```

Logs.java

```
public interface LogsRepository extends JpaRepository<Log, Integer>
{
    Log findLogById(String id);
    void deleteLogById(String id);
    void deleteLogsById(List<String> list);
}
```

NewsRepository.java

```
public interface NewsRepository extends JpaRepository<News, Integer>
{
    List<News> findAllByApprovedFalse();
    List<News> findAllByApprovedTrue();
    List<News> findAllByTitleContains(String title);
}
```

TalentRepository.java

```
public interface TalentRepository extends JpaRepository<Talent, Integer>
{
    List<Talent> findAllByApprovedFalse();
    List<Talent> findAllByApprovedTrue();
    List<Talent> findTop10ByApprovedTrueOrderByIdDesc();
}
```

UserRepository.java

```
public interface UserRepository extends JpaRepository<User, String>
{
    Optional<User> findByUsername(String username);
}

UserRoleRepository.java
public interface UserRoleRepository extends JpaRepository<UserRole, String>
{
    Optional<UserRole> findByAuthority(String authority);
}
```

5. Services

DonationService.java

```
public interface DonationService
{
    Donation getDonation(int id);
    void addDonation(Donation donation);
    void addDonation(Donation donation, MultipartFile file) throws IOException;
    List<Donation> getRequestedDonations();
    List<Donation> getApprovedDonations();
    boolean approveDonation(int id);
    boolean discardDonation(int id);
    boolean discardAllDonations();
    boolean approveAllDonations();
    List<Donation> getTop5ApprovedDonations();
}

```

DonationServiceImpl.java

```
@Service
public class DonationServiceImpl implements DonationService
{
    private final DonationRepository donationRepository;
    @Autowired
    public DonationServiceImpl(DonationRepository donationRepository) {
        this.donationRepository = donationRepository; }
    @Override
    public Donation getDonation(int id) { return
        this.donationRepository.findById(id).orElse(null); }
    @Override
    public void addDonation(Donation donation) { this.donationRepository.save(donation); }
    @Override
    public void addDonation(Donation donation, MultipartFile file) throws IOException
    {
        String imgName = file.getOriginalFilename();
        donation.setImgName(imgName);
        donation.setImgType(file.getContentType());
        donation.setImgData(file.getBytes());
        this.donationRepository.save(donation);
    }
    @Override
    public List<Donation> getRequestedDonations()
    {
        List<Donation> requestedDonations =
        this.donationRepository.findAllByApprovedFalse();
        Collections.reverse(requestedDonations);
        return requestedDonations;
    }
    @Override
    public List<Donation> getApprovedDonations()
    {
        List<Donation> approvedDonations = this.donationRepository.findAllByApprovedTrue();
        Collections.reverse(approvedDonations);
        return approvedDonations;
    }
    @Override
```

```
public boolean approveDonation(int id)
{
    Optional<Donation> getRequestedDonation = this.donationRepository.findById(id);
    if(getRequestedDonation.isPresent())
    {
        getRequestedDonation.get().setApproved(true);
        this.donationRepository.save(getRequestedDonation.get());
        return true;
    }
    return false;
}
@Override
public boolean discardDonation(int id)
{
    Optional<Donation> getRequestedDonation = this.donationRepository.findById(id);
    if(getRequestedDonation.isPresent())
    {
        this.donationRepository.delete(getRequestedDonation.get());
        return true;
    }
    return false;
}
@Override
public boolean approveAllDonations()
{
    List<Donation> getRequestedDonations =
this.donationRepository.findAllByApprovedFalse();
    if(getRequestedDonations != null)
    {
        for(Donation donation : getRequestedDonations)
        {
            donation.setApproved(true);
            this.donationRepository.save(donation);
        }
        return true;
    }
    return false;
}
@Override
public boolean discardAllDonations()
{
    List<Donation> getRequestedDonations =
this.donationRepository.findAllByApprovedFalse();
    if(getRequestedDonations != null)
    {
        for(Donation donation : getRequestedDonations)
        {
            this.donationRepository.delete(donation);
        }
        return true;
    }
    return false;
}
@Override
public List<Donation> getTop5ApprovedDonations()
{
    List<Donation> top5ApprovedDonations =
this.donationRepository.findTop5ByApprovedTrueOrderByIdDesc();
    return top5ApprovedDonations;
}
}
EventService.java
public interface EventService
{
    Event getEvent(int id);
```

```

    void addEvent(Event event);
    void addEvent(Event event, MultipartFile file) throws IOException;
    List<Event> getRequestedEvents();
    List<Event> getApprovedEvents();
    List<Event> getTop6ApprovedEvents();
    List<Event> getTop5ApprovedEvents();
    boolean approveEvent(int id);
    boolean discardEvent(int id);
    boolean discardAllEvents();
    boolean approveAllEvents();
}

```

EventServiceImpl.java

```

@Service
public class EventServiceImpl implements EventService
{
    private final EventRepository eventRepository;
    @Autowired
    public EventServiceImpl(EventRepository eventRepository) { this.eventRepository =
eventRepository; }
    @Override
    public Event getEvent(int id) { return this.eventRepository.findById(id).orElse(null); }
    @Override
    public void addEvent(Event event) { this.eventRepository.save(event); }
    @Override
    public void addEvent(Event event, MultipartFile file) throws IOException
    {
        String imgName = file.getOriginalFilename();
        event.setImgName(imgName);
        event.setImgType(file.getContentType());
        event.setImgData(file.getBytes());

        this.eventRepository.save(event);
    }
    @Override
    public List<Event> getRequestedEvents()
    {
        List<Event> requestedEvents = this.eventRepository.findAllByApprovedFalse();
        Collections.reverse(requestedEvents);

        return requestedEvents;
    }
    @Override
    public List<Event> getApprovedEvents()
    {
        List<Event> approvedEvents = this.eventRepository.findAllByApprovedTrue();
        Collections.reverse(approvedEvents);
        return approvedEvents;
    }
    @Override
    public List<Event> getTop5ApprovedEvents()
    {
        List<Event> top5ApprovedEvents =
this.eventRepository.findTop5ByApprovedTrueOrderByIdDesc();
        return top5ApprovedEvents;
    }
    @Override
    public List<Event> getTop6ApprovedEvents()
    {
        List<Event> top6ApprovedEvents =
this.eventRepository.findTop6ByApprovedTrueOrderByIdDesc();
        return top6ApprovedEvents;
    }
    @Override
    public boolean approveEvent(int id)

```

```

{
    Optional<Event> getRequestedEvent = this.eventRepository.findById(id);
    if(getRequestedEvent.isPresent())
    {
        getRequestedEvent.get().setApproved(true);
        this.eventRepository.save(getRequestedEvent.get());
        return true;
    }
    return false;
}
@Override
public boolean discardEvent(int id)
{
    Optional<Event> getRequestedEvent = this.eventRepository.findById(id);
    if(getRequestedEvent.isPresent())
    {
        this.eventRepository.delete(getRequestedEvent.get());
        return true;
    }
    return false;
}
@Override
public boolean approveAllEvents()
{
    List<Event> getRequestedEvents = this.eventRepository.findAllByApprovedFalse();
    if(getRequestedEvents != null)
    {
        for(Event event : getRequestedEvents)
        {
            event.setApproved(true);
            this.eventRepository.save(event);
        }
        return true;
    }
    return false;
}
@Override
public boolean discardAllEvents()
{
    List<Event> getRequestedEvents = this.eventRepository.findAllByApprovedFalse();
    if(getRequestedEvents != null)
    {
        for(Event event : getRequestedEvents)
        {
            this.eventRepository.delete(event);
        }
        return true;
    }
    return false;
}
}

GoodThingsService.java
public interface GoodThingsService
{
    GoodThing getGoodThing(int id);
    void addGoodThing(GoodThing goodThing);
    void addGoodThing(GoodThing goodThing, MultipartFile file) throws IOException;
    List<GoodThing> getRequestedGoodThings();
    List<GoodThing> getApprovedGoodThings();
    List<GoodThing> getTop10ApprovedGoodThings();
    boolean discardGoodThing(int id);
    boolean approveGoodThing(int id);
    boolean discardAllRequestedGoodThings();
    boolean approveAllRequestedGoodThings();
}

```

```

        LinkedHashMap<String, Long> getTopThreeAuthors();
    }
GoodThingsServiceImpl.java
@Service
public class GoodThingsServiceImpl implements GoodThingsService
{
    private final GoodThingRepository goodThingRepository;
    @Autowired
    public GoodThingsServiceImpl(GoodThingRepository goodThingRepository) {
this.goodThingRepository = goodThingRepository; }
    @Override
    public GoodThing getGoodThing(int id) { return
this.goodThingRepository.findById(id).orElse(null); }
    @Override
    public void addGoodThing(GoodThing goodThing)
{this.goodThingRepository.save(goodThing); }
    @Override
    public void addGoodThing(GoodThing goodThing, MultipartFile file) throws IOException
    {
        String imgName = file.getOriginalFilename();
        goodThing.setImgName(imgName);
        goodThing.setImgType(file.getContentType());
        goodThing.setImgData(file.getBytes());
        this.goodThingRepository.save(goodThing);
    }
    @Override
    public List<GoodThing> getRequestedGoodThings()
    {
        List<GoodThing> requestedGoodThings =
this.goodThingRepository.findAllByApprovedFalse();
        Collections.reverse(requestedGoodThings);
        return requestedGoodThings;
    }
    @Override
    public List<GoodThing> getApprovedGoodThings()
    {
        List<GoodThing> approvedGoodThings =
this.goodThingRepository.findAllByApprovedTrue();
        Collections.reverse(approvedGoodThings);
        return approvedGoodThings;
    }
    @Override
    public boolean approveGoodThing(int id)
    {
        Optional<GoodThing> getRequestedGoodThing = this.goodThingRepository.findById(id);
        if(getRequestedGoodThing.isPresent())
        {
            getRequestedGoodThing.get().setApproved(true);
            this.goodThingRepository.save(getRequestedGoodThing.get());
            return true;
        }
        return false;
    }
    @Override
    public boolean discardGoodThing(int id)
    {
        Optional<GoodThing> getRequestedGoodThing = this.goodThingRepository.findById(id);
        if(getRequestedGoodThing.isPresent())
        {
            this.goodThingRepository.delete(getRequestedGoodThing.get());
            return true;
        }
        return false;
    }
    @Override

```

```

public boolean approveAllRequestedGoodThings()
{
    List<GoodThing> getRequestedGoodThings =
this.goodThingRepository.findAllByApprovedFalse();
    if(getRequestedGoodThings != null)
    {
        for(GoodThing goodThing : getRequestedGoodThings)
        {
            goodThing.setApproved(true);
            this.goodThingRepository.save(goodThing);
        }
        return true;
    }
    return false;
}
@Override
public boolean discardAllRequestedGoodThings()
{
    List<GoodThing> getRequestedGoodThings =
this.goodThingRepository.findAllByApprovedFalse();
    if(getRequestedGoodThings != null)
    {
        for(GoodThing goodThing : getRequestedGoodThings)
        {
            this.goodThingRepository.delete(goodThing);
        }
        return true;
    }
    return false;
}
@Override
public LinkedHashMap<String,Long> getTopThreeAuthors()
{
    List<Object[]> list = this.goodThingRepository.getTopThreeAuthors();
    if(list.size() >= 3)
    {
        LinkedHashMap<String, Long> topThreeAuthors = new LinkedHashMap<String,
Long>();
        for (int i = 0; i < 3; i++)
        {
            String key = (String) list.get(i)[0];
            System.out.print(key + " - ");
            long value = (Long) list.get(i)[1];
            System.out.print(value + "\n");
            topThreeAuthors.put(key, value);
        }
        return topThreeAuthors;
    }
    return null;
}
@Override
public List<GoodThing> getTop10ApprovedGoodThings()
{
    List<GoodThing> top10ApprovedGoodThings =
this.goodThingRepository.findTop10ByApprovedTrueOrderByByIdDesc();
    return top10ApprovedGoodThings;
}
}

LogService.java
public interface LogService
{
    boolean insertLog(Log log);
    void removeLogsById(List<String> list);
    List<Log> getAll();
}

```

```

    void removeAllLogs();
}
LogServiceImpl.java
@Service
public class LogServiceImpl implements LogService
{
    private final LogsRepository logsRepository;
    @Autowired
    public LogServiceImpl(LogsRepository logsRepository) { this.logsRepository =
logsRepository; }
    @Override
    public boolean insertLog(Log log) { return this.logsRepository.save(log) != null; }
    @Override
    public void removeLogsById(List<String> list)
    {
        for(String uuid : list)
        {
            Log log = this.logsRepository.findLogById(uuid);
            this.logsRepository.delete(log);
        }
    }
    @Override
    public List<Log> getAll() { return
this.logsRepository.findAll().stream().collect(Collectors.toUnmodifiableList()); }
    @Override
    public void removeAllLogs() { this.logsRepository.deleteAll(); }
}

```

6. Additional bussines logic

CoronaInfo.java

```

public class CoronaInfo
{
    public static JSONObject getApiResponse() throws IOException
    {
        OkHttpClient client = new OkHttpClient().newBuilder()
            .build();
        Request request = new Request.Builder()
            .url("https://api.covid19api.com/summary")
            .method("GET", null)
            .build();
        Response response = client.newCall(request).execute();
        String jsonString = response.body().string();
        JSONObject obj = new JSONObject(jsonString);

        return obj;
    }
    public static Global getGlobalInfo(JSONObject obj, ObjectMapper mapper) throws
IOException
    {
        JSONObject globalJSON = obj.getJSONObject("Global");
        Global global = mapper.readValue(globalJSON.toString(), new
TypeReference<Global>(){});
        return global;
    }
    public static List<Country> getCountriesInfo(JSONObject obj, ObjectMapper mapper)
throws IOException
    {
        JSONArray arr = obj.getJSONArray("Countries");
        String countriesJSON = arr.toString();
        List<Country> countries = mapper.readValue(countriesJSON, new
TypeReference<List<Country>>(){});
        Collections.sort(countries, new CoronaConfirmedComparator());

        return countries;
    }
}

```

```

    }
    public static Country getMacedoniaInfo(JSONObject obj, ObjectMapper mapper) throws
IOException
{
    Country country;
    JSONArray arr = obj.getJSONArray("Countries");
    JSONObject jsonObject = arr.getJSONObject(100);
    String countryCode = jsonObject.getString("CountryCode");
    String slug = jsonObject.getString("Slug");
    if(countryCode.equals("MK") && slug.equals("macedonia"))
    {
        country = mapper.readValue(jsonObject.toString(), new
TypeReference<Country>(){});
        return country;
    }
    else
    {
        String countriesJSON = arr.toString();
        List<Country> countries = mapper.readValue(countriesJSON, new
TypeReference<List<Country>>(){});
        for(Country currCountry : countries)
        {
            if(currCountry.CountryCode.equals("MK") &&
currCountry.Slug.equals("macedonia"))
            {
                return currCountry;
            }
        }
        return null;
    }
}

```

CoronaConfirmedComparator.java

```

public class CoronaConfirmedComparator implements Comparator<Country>
{
    @Override
    public int compare(Country a, Country b)
    {
        if(a.TotalConfirmed > b.TotalConfirmed)
        {
            return -1;
        }
        else
        {
            return 1;
        }
    }
}

```

ImageEncoder.java

```

public class ImageEncoder
{
    public static final String noImageTalent = "...";
    public String generateBase64Image(byte [] img, String noImage)
    {
        if(img == null)
        {
            return noImage;
        }
        return Base64.getEncoder().encodeToString(img);
    }
    public static String getDefaultImage(String path) throws IOException
    {
        FileInputStream fis = new FileInputStream(path);

        String defaultImage = IOUtils.toString(fis, "UTF-8");

```

```

        return defaultImage;
    }
}
Logs.java
public class Logs
{
    private static final DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
    public static Log setupLog(Authentication authentication, String table, Operation
operation)
    {
        Log log = new Log();
        Date date = new Date();
        log.setUser(authentication.getName());
        log.setOperation(operation);
        log.setTableName(table);
        log.setDate(dateFormat.format(date));
        if(authentication != null)
        {
            String username = null;
            if(authentication instanceof OAuth2AuthenticationToken)
            {
                OAuth2AuthenticationToken oauthToken = (OAuth2AuthenticationToken)
authentication;
                Map<String, Object> details = (Map<String, Object>)
oauthToken.getPrincipal().getAttributes();

                username = details.get("name").toString();
            }
            else
            {
                username = authentication.getName();
            }
            log.setUser(username);
        }
        else
        {
            log.setUser("Guest"); // should never come here
        }
        return log;
    }
}
Username.java
public class Username
{
    public static String get(Authentication authentication)
    {
        String username = "Guest";
        if(authentication != null)
        {
            if(authentication instanceof OAuth2AuthenticationToken)
            {
                OAuth2AuthenticationToken oauthToken = (OAuth2AuthenticationToken)
authentication;
                Map<String, Object> details = (Map<String, Object>)
oauthToken.getPrincipal().getAttributes();
                username = details.get("name").toString();
            }
            else
            {
                username = authentication.getName();
            }
        }
        return username;
    }
}

```

```

    }
}
```

6. Controllers

AccessController.java

```

@Controller
public class AcessController
{
    @GetMapping("/unauthorized")
    public String unauthorized() { return "error/unauthorized.html"; }
}
```

DonationsController.java

```

@Controller
@RequestMapping("/donations")
public class DonationsController
{
    private final DonationService donationService;
    private final String defaultImageDonation;
    private final LogService logService;
    private static final DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
    @Autowired
    public DonationsController(DonationService donationService, String
defaultImageDonation, LogService logService)
    {
        this.donationService = donationService;
        this.defaultImageDonation = defaultImageDonation;
        this.logService = logService;
    }
    @RequestMapping(method = RequestMethod.GET)
    public String donations(Authentication authentication, Model model) throws IOException
    {
        String username = Username.get(authentication);
        model.addAttribute("username", username);
        ImageEncoder imageEncoder = new ImageEncoder();
        model.addAttribute("defaultImageDonation", defaultImageDonation);
        model.addAttribute("imageEncoder", imageEncoder);
        List<Donation> approvedDonations = donationService.getApprovedDonations();
        model.addAttribute("approvedDonations", approvedDonations);
        if(authentication != null)
        {
            Donation donation = new Donation();
            model.addAttribute("donation", donation);
        }
        if(authentication != null && authentication.getAuthorities().stream().anyMatch(a ->
a.getAuthority().equals("MODERATOR")))
        {
            List<Donation> requestedDonations = donationService.getRequestedDonations();
            model.addAttribute("requestedDonations", requestedDonations);
        }

        return "pages/donations.html";
    }
    @PostMapping("/add")
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public String addTalent(@ModelAttribute Donation donation, @RequestParam("files")
MultipartFile file, Model model, Authentication authentication) throws IOException
    {
        donation.setApproved(false);
        donation.setAuthor(Username.get(authentication));
        if(!file.isEmpty())
        {
            this.donationService.addDonation(donation, file);
        }
        else
    }
```

```

    {
        this.donationService.addDonation(donation);
    }
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (auth != null && auth.getAuthorities().stream().anyMatch(a ->
a.getAuthority().equals("MODERATOR")))
    {
        List<Donation> requestedDonations =
this.donationService.getRequestedDonations();
        model.addAttribute("requestedDonations", requestedDonations);
        ImageEncoder imageEncoder = new ImageEncoder();
        model.addAttribute("defaultImageDonation", defaultImageDonation);
        model.addAttribute("imageEncoder", imageEncoder);
    }
    Log log = Logs.setupLog(authentication, "Donations", Operation.Add);
    this.logService.insertLog(log);
    return "pages/donations.html :: #req-donations";
}
@PostMapping("/approve")
@PreAuthorize("hasAuthority('MODERATOR')")
public String approveDonation(@RequestParam("donationApproveId") int id, Authentication authentication)
{
    boolean approved = this.donationService.approveDonation(id);
    Log log = Logs.setupLog(authentication, "Donations", Operation.Approve);
    this.logService.insertLog(log);
    return "redirect:/donations";
}
@PostMapping("/discard")
@PreAuthorize("hasAuthority('MODERATOR')")
public String discardDonation(@RequestParam("donationDiscardId") int id, Authentication authentication)
{
    boolean discarded = this.donationService.discardDonation(id);
    Log log = Logs.setupLog(authentication, "Donations", Operation.Discard);
    this.logService.insertLog(log);
    return "redirect:/donations";
}
@PostMapping("/discard-all")
@PreAuthorize("hasAuthority('MODERATOR')")
public String discardAllDonations(Authentication authentication)
{
    boolean discardedAll = this.donationService.discardAllDonations();
    Log log = Logs.setupLog(authentication, "Donations", Operation.DiscardALL);
    this.logService.insertLog(log);
    return "redirect:/donations";
}
@PostMapping("/approve-all")
public String approveAllDonations(Authentication authentication)
{
    boolean approvedAll = this.donationService.approveAllDonations();
    Log log = Logs.setupLog(authentication, "Donations", Operation.ApproveALL);
    this.logService.insertLog(log);
    return "redirect:/donations";
}
@GetMapping("/single-donation/{id}")
public String getSingleDonation(@PathVariable(name = "id") int id, Model model,
Authentication authentication)
{
    String username = Username.get(authentication);
    model.addAttribute("username",username);

    Donation donation = this.donationService.getDonation(id);

    if(donation != null && donation.isApproved())
    {

```

```

        ImageEncoder imageEncoder = new ImageEncoder();
        model.addAttribute("defaultImageDonation", defaultImageDonation);
        model.addAttribute("imageEncoder", imageEncoder);
        model.addAttribute("donation", donation);
        List<Donation> top5ApprovedDonations =
this.donationService.getTop5ApprovedDonations();
        model.addAttribute("top5ApprovedDonations", top5ApprovedDonations);

        return "pages/single-donation.html";
    }
    else
    {
        return "error/400.html";
    }
}
EventsController.java
@Controller
@RequestMapping("/events")
public class EventsController
{
    private final EventService eventService;
    private final String defaultImageEvent;
    private final LogService logService;
    private static final DateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");
    @Autowired
    public EventsController(EventService eventService, String defaultImageEvent, LogService
logService)
    {
        this.eventService = eventService;
        this.defaultImageEvent = defaultImageEvent;
        this.logService = logService;
    }
    @RequestMapping(method = RequestMethod.GET)
    public String events(Model model, Authentication authentication)
    {
        String username = Username.get(authentication);
        model.addAttribute("username",username);
        model.addAttribute("sdf", sdf);
        ImageEncoder imageEncoder = new ImageEncoder();
        model.addAttribute("defaultImageEvent", defaultImageEvent);
        model.addAttribute("imageEncoder", imageEncoder);
        List<Event> approvedEvents = this.eventService.getApprovedEvents();
        model.addAttribute("approvedEvents", approvedEvents);
        if(authentication != null)
        {
            Event event = new Event();
            model.addAttribute("event", event);
        }
        if (authentication != null && authentication.getAuthorities().stream().anyMatch(a -
> a.getAuthority().equals("MODERATOR")))
        {
            List<Event> requestedEvents = this.eventService.getRequestedEvents();
            model.addAttribute("requestedEvents", requestedEvents);
        }
        return "pages/events.html";
    }
    @PostMapping("/add")
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public String addEvent(@ModelAttribute Event event, @RequestParam("files")
MultipartFile file, Model model, Authentication authentication) throws IOException
    {
        event.setApproved(false);
        if(!file.isEmpty())
        {

```

```

        this.eventService.addEvent(event,file);
    }
    else
    {
        this.eventService.addEvent(event);
    }
    if (authentication != null && authentication.getAuthorities().stream().anyMatch(a -
> a.getAuthority().equals("MODERATOR")))
    {
        model.addAttribute("sdf", sdf);
        List<Event> requestedEvents = this.eventService.getRequestedEvents();
        model.addAttribute("requestedEvents", requestedEvents);
        ImageEncoder imageEncoder = new ImageEncoder();
        model.addAttribute("defaultImageEvent", defaultImageEvent);
        model.addAttribute("imageEncoder", imageEncoder);
    }
    Log log = Logs.setupLog(authentication, "Events", Operation.Add);
    this.logService.insertLog(log);

    return "pages/events.html :: #req-events";
}
@PostMapping("/add-from-home")
@PreAuthorize("hasAuthority('ROLE_USER')")
public String addEventFromHome(@ModelAttribute Event event, @RequestParam("files")
MultipartFile file, Model model, Authentication authentication) throws IOException
{
    event.setApproved(false);
    if(!file.isEmpty())
    {
        this.eventService.addEvent(event,file);
    }
    else
    {
        this.eventService.addEvent(event);
    }
    Log log = Logs.setupLog(authentication, "Events", Operation.Add);
    this.logService.insertLog(log);
    return "pages/index.html :: #random";
}
@PostMapping("/approve")
@PreAuthorize("hasAuthority('MODERATOR')")
public String approveEvent(@RequestParam("eventApproveId") int id, Authentication
authentication)
{
    boolean approved = this.eventService.approveEvent(id);
    Log log = Logs.setupLog(authentication, "Events", Operation.Approve);
    this.logService.insertLog(log);
    return "redirect:/events";
}
@PostMapping("/discard")
@PreAuthorize("hasAuthority('MODERATOR')")
public String discardEvent(@RequestParam("eventDiscardId") int id, Authentication
authentication)
{
    boolean discarded = this.eventService.discardEvent(id);

    Log log = Logs.setupLog(authentication, "Events", Operation.Discard);
    this.logService.insertLog(log);

    return "redirect:/events";
}
@PostMapping("/discard-all")
@PreAuthorize("hasAuthority('MODERATOR')")
public String discardAllEvents(Authentication authentication)
{
    boolean discardedAll = this.eventService.discardAllEvents();
}

```

```

        Log log = Logs.setupLog(authentication, "Events", Operation.DiscardAll);
        this.logService.insertLog(log);

        return "redirect:/events";
    }
    @PostMapping("/approve-all")
    @PreAuthorize("hasAuthority('MODERATOR')")
    public String approveAllEvents(Authentication authentication)
    {
        boolean approvedAll = this.eventService.approveAllEvents();
        Log log = Logs.setupLog(authentication, "Events", Operation.ApproveAll);
        this.logService.insertLog(log);
        return "redirect:/events";
    }
    @GetMapping("/single-event/{id}")
    public String getSingleEvent(@PathVariable(name = "id") int id, Model model,
Authentication authentication)
    {
        String username = Username.get(authentication);
        model.addAttribute("username",username);

        Event event = this.eventService.getEvent(id);

        if(event != null && event.isApproved())
        {
            ImageEncoder imageEncoder = new ImageEncoder();
            model.addAttribute("sdf", sdf);
            model.addAttribute("defaultImageEvent", defaultImageEvent);
            model.addAttribute("imageEncoder", imageEncoder);
            model.addAttribute("event",event);
            List<Event> top5ApprovedEvents = this.eventService.getTop5ApprovedEvents();
            model.addAttribute("top5ApprovedEvents", top5ApprovedEvents);

            return "pages/single-event.html";
        }
        else
        {
            return "error/400.html";
        }
    }
}

```

HomeController.java

```

@Controller
public class HomeController
{
    private final NewsService newsService;
    private final EventService eventService;
    private final GoodThingsService goodThingsService;
    private final DonationService donationService;
    private final TalentService talentService;
    private final String defaultImageNews;
    private final String defaultImageGoodThing;
    private final String defaultImageEvent;
    private final String defaultImageDonation;
    private static final DateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    private static final DateFormat sdfEvents = new SimpleDateFormat("dd/MM/yyyy - HH:mm");
    private static final DateFormat sdfEventsDate = new SimpleDateFormat("dd.MM.yyyy");
    private static final DateFormat sdfEventsTime = new SimpleDateFormat("HH:mm");
    @Autowired
    public HomeController(GoodThingsService goodThingsService, NewsService newsService,
EventService eventService, DonationService donationService, TalentService talentService,
String defaultImageNews, String defaultImageGoodThing, String defaultImageEvent, String
defaultImageDonation)
    {
        this.goodThingsService = goodThingsService;
    }
}

```

```

    this.newsService = newsService;
    this.eventService = eventService;
    this.donationService = donationService;
    this.talentService = talentService;
    this.defaultImageNews = defaultImageNews;
    this.defaultImageGoodThing = defaultImageGoodThing;
    this.defaultImageEvent = defaultImageEvent;
    this.defaultImageDonation = defaultImageDonation;
}
@RequestMapping("/user")
public Principal user(Principal principal) { return principal; }
@GetMapping("/")
public String index(Model model, @AuthenticationPrincipal Authentication authentication) throws IOException
{
    String username = Username.get(authentication);
    model.addAttribute("username",username);
    ObjectMapper mapper = new ObjectMapper();
    JSONObject obj = CoronaInfo.getApiResponse();
    Global global = CoronaInfo.getGlobalInfo(obj, mapper);
    List<Country> countries = CoronaInfo.getCountriesInfo(obj, mapper);
    Country mkd = CoronaInfo.getMacedoniaInfo(obj, mapper);
    model.addAttribute("global",global);
    model.addAttribute("countries",countries);
    model.addAttribute("mkd",mkd);
    model.addAttribute("sdf", sdf);
    model.addAttribute("sdfEvents", sdfEvents);
    model.addAttribute("sdfEventsDate", sdfEventsDate);
    model.addAttribute("sdfEventsTime", sdfEventsTime);
    ImageEncoder imageEncoder = new ImageEncoder();
    model.addAttribute("defaultImageNews", defaultImageNews);
    model.addAttribute("defaultImageEvent", defaultImageEvent);
    model.addAttribute("defaultImageGoodThing", defaultImageGoodThing);
    model.addAttribute("defaultImageDonation", defaultImageDonation);
    model.addAttribute("defaultImageTalent", ImageEncoder.noImageTalent);
    model.addAttribute("imageEncoder", imageEncoder);
    List<News> approvedNews = this.newsService.getApprovedNews();
    model.addAttribute("approvedNews", approvedNews);
    LinkedHashMap<String, Long> topThreeAuthors =
this.goodThingsService.getTopThreeAuthors();
    model.addAttribute("topThreeAuthors", topThreeAuthors);
    List<Event> top6ApprovedEvents = this.eventService.getTop6ApprovedEvents();
    model.addAttribute("top6ApprovedEvents", top6ApprovedEvents);
    List<GoodThing> top10ApprovedGoodThings =
goodThingsService.getTop10ApprovedGoodThings();
    model.addAttribute("top10ApprovedGoodThings", top10ApprovedGoodThings);
    List<Donation> approvedDonations = donationService.getApprovedDonations();
    model.addAttribute("approvedDonations", approvedDonations);
    List<Talent> approvedTalents = this.talentService.getApprovedTalents();
    model.addAttribute("approvedTalents", approvedTalents);
    Event event = new Event();
    model.addAttribute("event", event);
    UserRegisterBindingModel user = new UserRegisterBindingModel();
    model.addAttribute("user",user);
    return "pages/index.html";
}
}

```

LoginController.java

```

@Controller
public class LoginController
{
    private final UserService userService;
    private boolean newRegister;
    @Autowired
    public LoginController(UserService userService)

```

```

{
    this.userService = userService;
    this.newRegister=false;
}
@GetMapping("/login")
@PreAuthorize("isAnonymous()")
public String loginPage(Model model)
{
    if(newRegister==true)
    {
        model.addAttribute("newRegister", newRegister);
        newRegister = false;
    }
    UserRegisterBindingModel user = new UserRegisterBindingModel();
    model.addAttribute("user", user);
    return "login/login.html";
}
@GetMapping("/register")
@PreAuthorize("isAnonymous()")
public String registerPage(Model model)
{
    String error = null;
    model.addAttribute("error", error);
    UserRegisterBindingModel user = new UserRegisterBindingModel();
    model.addAttribute("user", user);
    return "login/register.html";
}
@PostMapping("/register")
@PreAuthorize("isAnonymous()")
public String registerPost(@ModelAttribute("user") UserRegisterBindingModel
userRegisterBindingModel, HttpServletResponse response, Model model)
{
    if(!userRegisterBindingModel.getPassword().equals(userRegisterBindingModel.getConfirmPasswo
rd()))
    {
        response.setHeader("PASSWORDS_UNMATCH", "1");
        String error = "Пасвордите не се совпадаат.";
        model.addAttribute("error", error);
        return "login/register.html :: #error";
    }
    boolean created = this.userService.createUser(userRegisterBindingModel);
    if(created == false)
    {
        response.setHeader("TAKEN_USERNAME", "1");
        String error = "Корисничко име вже постои.";
        model.addAttribute("error", error);
        return "login/register.html :: #error";
    }
    this.newRegister = true;
    return "login/login.html";
}
}

LogsController.java
@Controller
@RequestMapping(value="/logs")
public class LogsController
{
    @Autowired
    private final LogService logService;
    public LogsController(LogService logService) { this.logService = logService; }
    @GetMapping("/all")
    @PreAuthorize("hasAuthority('MODERATOR')")
    public String getLogs(Model model, Authentication authentication)
    {
}

```

```

        String username = Username.get(authentication);
        model.addAttribute("username",username);
        List<Log> list = this.logService.getAll();
        model.addAttribute("list",list);
        return "pages/logs.html";
    }
    @GetMapping("/delete")
    @PreAuthorize("hasAuthority('ROOT-ADMIN')")
    public String deleteLogs(@RequestParam("logs") List<String> list)
    {
        this.logService.removeLogsById(list);
        return "redirect:/logs/all";
    }
    @GetMapping("/delete/all")
    @PreAuthorize("hasAuthority('ROOT-ADMIN')")
    public String deleteAllLogs()
    {
        this.logService.removeAllLogs();
        return "redirect:/logs/all";
    }
}

```

OrganizationController.java

```

@Controller
@RequestMapping("/organization")
public class OrganizationController
{
    private final UserService userService;
    private final UserRoleService userRoleService;
    @Autowired
    public OrganizationController(LogService logService, UserService userService,
    UserRoleService userRoleService)
    {
        this.userService = userService;
        this.userRoleService = userRoleService;
    }
    @RequestMapping(method = RequestMethod.GET)
    @PreAuthorize("hasAuthority('ADMIN')")
    public String organization(Model model, Authentication authentication)
    {
        String username = Username.get(authentication);
        model.addAttribute("username",username);
        Set<User> users = this.userService.getAllUsers();
        model.addAttribute("users", users);
        UserRole root_admin = this.userRoleService.findRole("ROOT-ADMIN");
        model.addAttribute("root_admin", root_admin);
        UserRole admin = this.userRoleService.findRole("ADMIN");
        model.addAttribute("admin", admin);
        UserRole moderator = this.userRoleService.findRole("MODERATOR");
        model.addAttribute("moderator", moderator);
        UserRole role_user = this.userRoleService.findRole("ROLE_USER");
        model.addAttribute("role_user", role_user);
        return "pages/organization.html";
    }
    @PostMapping("/change-role")
    @PreAuthorize("hasAuthority('ADMIN')")
    public String changeRole(@RequestParam("username") String username, String change)
    {
        boolean changed = this.userService.changeRole(username,change);
        return "redirect:/organization";
    }
    @PostMapping("/delete-user")
    @PreAuthorize("hasAuthority('ADMIN')")
    public String deleteUser(@RequestParam("username") String username)
    { this.userService.deleteUser(username); return "redirect:/organization"; } }

```

7. Views(изгледи)

Login.html – Логин форма

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Неготино - Најава</title>
    <th:block th:replace="fragments/head"></th:block>
    <link rel="stylesheet" href="/css/login-style.css">
</head>
<body>
<th:block th:replace="fragments/preloader"></th:block>
<th:block th:replace="fragments/navbar"></th:block>
<div class="container mt-2 w-50" th:if="${newRegister==true}">
    <p class="register register-success text-center">Успешно се регистриравте.<br/>Можете да се најавите со корисничкото име и пасвордот.</p>
</div>
<div class="container" style="margin-top: 80px;">
    <div class="d-flex justify-content-center h-100">
        <div class="card">
            <div class="card-header">
                <h3>Најава</h3>
                <div class="d-flex justify-content-end social_icon">
                    <a href="/oauth2/authorization/facebook"><span><i class="fab fa-facebook-square"></i></span></a>
                    <a href="/oauth2/authorization/google"><span><i class="fab fa-google-plus-square"></i></span></a>
                    <span><i class="fab fa-twitter-square"></i></span>
                </div>
            <div class="card-body">
                <form method="POST" th:action="@{/login}" th:object="${user}" class="needs-validation" novalidate>
                    <div class="input-group form-group">
                        <div class="input-group-prepend">
                            <span class="input-group-text"><i class="fas fa-user"></i></span>
                        </div>
                        <input type="text" class="form-control" th:field="*{username}" placeholder="Корисничко име" required>
                    </div>
                    <div class="input-group form-group">
                        <div class="input-group-prepend">
                            <span class="input-group-text"><i class="fas fa-key"></i></span>
                        </div>
                        <input type="password" class="form-control" th:field="*{password}" placeholder="Пасврд" required>
                    </div>
                    <div class="row align-items-center remember">
                        <input id="rememberMe" name="rememberMe" type="checkbox">
                        <label class="lblRemember" for="rememberMe"> Запомни ме </label>
                    </div>
                    <div class="csrf">
                        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
                    </div>
                    <div class="form-group">
                        <input type="submit" value="Најави се" class="btn float-right login_btn">
                    </div>
                    <div class="error mb-5" style="color: red;" th:if="${param.error}">
                        <th:with="errorMsg=${session['SPRING_SECURITY_LAST_EXCEPTION'].message}">
                            <span th:if="${errorMsg=='Bad credentials'}">Грешно име или пасврд</span>
                        </th:with>
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>
```

```

        </div>
    </form>
</div>
<div class="card-footer">
    <div class="d-flex justify-content-center links">
        Немаш акаунт?<a href="/register">Регистрирай се</a>
    </div>
    <div class="d-flex justify-content-center">
        <a href="#">Го заборави пасвордот?</a>
    </div>
</div>
</div>
</div>
<footer class="py-2 bg-dark" style="margin-top: 105px;">
    <div class="container">
        <p class="m-0 text-center text-white">Avramov © Your Website 2020</p>
    </div>
</footer>
<th:block th:replace="fragments/js"></th:block>
<script>
    (function() {
        'use strict';
        window.addEventListener('load', function() {
            var forms = document.getElementsByClassName('needs-validation');
            var validation = Array.prototype.filter.call(forms, function(form) {
                form.addEventListener('submit', function(event) {
                    if (form.checkValidity() === false) {
                        event.preventDefault();
                        event.stopPropagation();
                    }
                    form.classList.add('was-validated');
                }, false);
            });
        }, false);
    })();
</script>
</body>
</html>

```

Register.html - Регистрация

```

<body>
<th:block th:replace="fragments/preloader"></th:block>
<th:block th:replace="fragments/navbar"></th:block>
<div class="container" style="margin-top: 80px;">
    <div class="d-flex justify-content-center h-100">
        <div class="card-registration">
            <div class="card-header">
                <h3>Регистрација</h3>
                <div class="d-flex justify-content-end social_icon">
                    <span><i class="fab fa-facebook-square"></i></span>
                    <span><i class="fab fa-google-plus-square"></i></span>
                    <span><i class="fab fa-twitter-square"></i></span>
                </div>
            </div>
            <div class="card-body">
                <form method="POST" th:action="@{/}" th:object="${user}" class="needs-validation" novalidate>
                    <div class="input-group form-group">
                        <div class="input-group-prepend">
                            <span class="input-group-text"><i class="fas fa-user" th:text="${username}"></i></span>
                        </div>
                        <input id="username" type="text" class="form-control" th:field="*{username}" placeholder="Корисничко име" maxlength="50" minlength="3" required>
                        <div class="invalid-feedback">

```

Корисничкото име треба да е над 3 символи

```

</div>
</div>
<div class="input-group form-group">
    <div class="input-group-prepend">
        <span class="input-group-text"><i class="fas fa-envelope"></i></span>
    </div>
    <input type="email" class="form-control rounded-right" th:field="*{email}" placeholder="Майл" maxlength="100" required>
    <div class="invalid-feedback">
        Введете валиден email.
    </div>
</div>
<div class="input-group form-group">
    <div class="input-group-prepend">
        <span class="input-group-text"><i class="fas fa-key"></i></span>
    </div>
    <input type="password" id="pw" class="form-control rounded-right" th:field="*{password}" placeholder="Пасворт" minlength="6" maxlength="20" required>
    <div class="invalid-feedback">
        Пасвортот мора да е над 6 символи.
    </div>
</div>
<div class="input-group form-group">
    <div class="input-group-prepend">
        <span class="input-group-text"><i class="fas fa-key"></i></span>
    </div>
    <input type="password" id="confirm-pw" class="form-control rounded-right" th:field="*{confirmPassword}" placeholder="Потврди пасворт" minlength="6" maxlength="20" required>
    <div class="invalid-feedback">
        Пасвортот мора да е над 6 символи.
    </div>
</div>
<div class="mt-2 mb-2">
    <div class="form-group2">
        <a href="/login" class="btn float-left back_login_btn mb-2">Кон
        најава</a>
    </div>
    <div class="form-group">
        <input type="submit" value="Регистрирај се" class="btn float-
        right register_btn mb-2">
    </div>
</div>
</div>
</div>
<!-- Footer -->
<footer class="py-2 bg-dark" style="margin-top: 108px;">
    <div class="container">
        <p class="m-0 text-center text-white">Неготино © 2020</p>
    </div>
</footer>
```

Donations.html – Донации (Дарения)

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
```

```
<title>Неготино - Донации</title>
<th:block th:replace="fragments/head"></th:block>
</head>
<body>
<th:block th:replace="fragments/preloader"></th:block>
<th:block th:replace="fragments/navbar"></th:block>
    
    <div id="carouselExampleFade" class="carousel slide carousel-fade" data-ride="carousel">
        <div class="carousel-inner">
            <th:block th:each="donation,iter : ${approvedDonations}">
                <div class="carousel-item single-blog-post style-1" th:classappend="${iter.index == 0 ? 'active' : ''}">
                    <div class="view">
                        
                    </div>
                    <div class="carousel-caption">
                        <div class="blog-content">
                            <div class="blog-content-hovered">
                                <div class="container">
                                    <a th:href="/donations/single-donation/${donation.getId()}" class="post-title" style="font-size:20px" th:text="${donation.getTitle()}"/>
                                    <span class="post-date mt-2" th:text="${donation.getContact()}>June 20, 2018???">
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </th:block>
        </div>
        <a class="carousel-control-prev" href="#carouselExampleFade" role="button" data-slide="prev">
            <span class="carousel-control-prev-icon" aria-hidden="true"></span>
            <span class="sr-only">Previous</span>
        </a>
        <a class="carousel-control-next" href="#carouselExampleFade" role="button" data-slide="next">
            <span class="carousel-control-next-icon" aria-hidden="true"></span>
            <span class="sr-only">Next</span>
        </a>
    </div>
    <div class="container mt-4">
        <div class="text-center">
        </div>
        <div class="container">
            <div class="card-columns">
                <div class="card h-100" th:each="donation : ${approvedDonations}">
                    <a th:href="/donations/single-donation/${donation.getId()}">
                        
                    </a>
                    <div class="card-body">
                        <p class="rounded bg-warning float-right p-1" th:text="${donation.getType()}"/>
                        <h4 class="card-title" th:text="${donation.getTitle()}"/>
                        </h4>
                        <p class="card-text" th:text="${donation.getText()}"/>
                    </div>
                    <div class="card-footer text-right">
                        <p class="text-muted float-left mt-1">
                            <i class="far fa-user"></i>
                            <small th:text="${donation.getAuthor()}"/>
                        </small>
                        <i class="fas fa-phone-alt ml-2"></i>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
```

```

        <small th:text="${donation.getContact()}></small>
    </p>
    <form th:action="@{/donations/single-
donation/{path}(path=${donation.getId()}]}" >
        <button class="btn btn-secondary">Прочитај</button>
    </form>
</div>
<div>
    <form sec:authorize="hasAuthority('ADMIN')" method="POST" style="text-align: center" th:action="@{/donations/discard}" onsubmit="return confirm('Do you really want to discard the requested donation?');">
        <input type="hidden" name="donationDiscardId" id="lbl" th:value="${donation.getId()}" />
        <button type="submit" name="discard" id="discard" value="value" class="btn btn-danger">Discard</button>
    </form>
</div>
</div>
</div>
</div>
<div sec:authorize="hasAuthority('ROLE_USER')" class="bg-light">
    <div class="container p-4">
        <form class="needs-validation mt-2" th:method="post" th:action="@{/}" th:object="${donation}" enctype="multipart/form-data" novalidate>
            <div class="row ml-2">
                <div class="col-4 border border-darkgray rounded-lg">
                    <h3 class="pt-3">
                        <center>Донирај/Побарај</center>
                    </h3>
                    <div class="form-group mt-4 mb-2">
                        <label for="exampleFormControlSelect1">Одбери вид</label>
                        <select th:field="*{type}" class="custom-select" id="exampleFormControlSelect1">
                            <option th:value="Апел">Потребна е донација</option>
                            <option th:value="Донираам">Јас донирам</option>
                        </select>
                    </div>
                    <div class="form-row">
                        <div class="col-md-12">
                            <label for="validationCustom01">Наслов</label>
                            <div class="input-group">
                                <input type="text" th:field="*{title}" class="form-control" id="validationCustom01" autocomplete="off" placeholder="Наслов на донацијата" required>
                                <div class="input-group-append">
                                    <div class="input-group-text"><i class="fas fa-hand-holding-heart"></i></div>
                                </div>
                            <div class="invalid-feedback">
                                Введи име
                            </div>
                        </div>
                    </div>
                </div>
                <div class="form-row">
                    <div class="col-md-12 mt-2">
                        <label for="organizer">Контакт</label>
                        <div class="input-group">
                            <input type="text" th:field="*{contact}" class="form-control" id="organizer" placeholder="Телефонски број" autocomplete="off" required>
                            <div class="input-group-append">
                                <div class="input-group-text"><i class="fas fa-phone"></i></div>
                            </div>
                        <div class="invalid-feedback">
                            Введи број
                        </div>
                    </div>
                </div>
            </div>
        </form>
    </div>
</div>

```

```

        </div>
    </div>
</div>
<div class="form-row">
    <div class="col-md-12 mt-2">
        <label for="validationCustom04">Опис</label>
        <div class="input-group">
            <textarea rows="4" th:field="*{text}" class="form-control"
id="validationCustom04" autocomplete="off" placeholder="За што точно се бара донација или
имаш желба да подариш. Опис и начин на извршување на донацијата"></textarea>
            <div class="input-group-append">
                <div class="input-group-text"><i class="fas fa-edit"></i></div>
            </div>
            <div class="invalid-feedback">
                Описи ја донацијата
            </div>
        </div>
    </div>
<div class="form-row">
    <div class="col-md-12 mt-2">
        <div class="input-group">
            <input type="hidden" class="form-control" id="image" autocomplete="off"
required>
            <div class="input-group-append">
                </div>
            </div>
        </div>
    </div>
    <button class="btn btn-dark float-right mt-3 mb-2"
type="submit">Сподели</button>
</div>
<div class="col-2"></div>
<div class="col-6 d-flex align-items-center">
    <div class="p-3">
        <div class="image-upload-wrap-donations">
            <input class="file-upload-input-donations" type='file' name="files"
 onchange="readURL(this); accept="image/*" />
            <div class="drag-text-donations">
                <h3>Додади насловна фотографија</h3>
            </div>
        </div>
        <div class="file-upload-content-donations">
            <img class="file-upload-image-donations" alt="your image" />
            <div class="image-title-wrap-donations mt-2">
                <button type="button" id="removeImage" onclick="removeUpload()"
class="remove-image-donations">Remove <span class="image-title-donations">Uploaded
Image</span></button>
            </div>
        </div>
    </div>
</div>
</div>
</div>
<div id="req-donations" sec:authorize="hasAuthority('MODERATOR')" class="requested-
donations m-2">
    <div class="container mb-2 mt-3 border-bottom border-dark"
th:if="${requestedDonations.size() == 0}">
        <h3 style="text-align:center">Requested Donations</h3>
    </div>
    <div class="container mb-2 mt-3 border-bottom border-dark"
th:unless="${requestedDonations.size() == 0}">
        <h3 style="text-align:center" th:text="|Requested Donations -
${requestedDonations.size()}"></h3>
    </div>
</div>

```

```
</div>
<th:block class="p-3" th:if="${requestedDonations.size() == 0}">
    <p style="text-align:center">There are no requested Donations</p>
</th:block>
<th:block class="p-3" th:unless="${requestedDonations.size() == 0}">
<div class="container mt-4">
    <div class="card-columns">
        <div class="card h-100" th:each="donation : ${requestedDonations}">
            <a href="#">
                
            </a>
            <div class="card-body">
                <p class="rounded border bg-warning float-right p-1"
th:text="${donation.getType()}"/></p>
                <h4 class="card-title" th:text="${donation.getTitle()}"/>
                </h4>
                <p class="card-text" th:text="${donation.getText()}"/></p>
            </div>
            <div class="card-footer text-right">
                <p class="text-muted float-left mt-1">
                    <i class="far fa-user"></i>
                    <small th:text="${donation.getAuthor()}"/>
                </small>
                    <i class="fas fa-phone-alt ml-2"></i>
                    <small th:text="${donation.getContact()}"/></small>
                </p>
                <button class="btn btn-secondary">Прочитај</button>
            </div>
        <div class="row w-100 mr-0">
            <div class="col-3 float-left">
                <form method="POST" th:action="@{/donations/approve}" ;>
                    <input type="hidden" id="lbl15" name="donationApproveId"
th:value="${donation.getId()}" />
                    <button type="submit" name="approve" id="approve" value="value4" class="btn
btn-success">Approve</button>
                </form>
            </div>
            <div class="col-6"></div>
            <div class="col-3 float-right">
                <form method="POST" th:action="@{/donations/discard}" onsubmit="return
confirm('Do you really want to discard the requested donation?');">
                    <input type="hidden" name="donationDiscardId" id="lbl"
th:value="${donation.getId()}" />
                    <button type="submit" name="discard" id="discard" value="value" class="btn
btn-danger">Discard</button>
                </form>
            </div>
        </div>
    </div>
</div>
<div class="container mb-2 mt-3 border-top border-dark">
<div class="row mt-2">
    <div class="col-lg-12">
        <form method="POST" style="float:left;" th:action="@{/donations/approve-all}"
onsubmit="return confirm('Do you really want to approve all the requested donations?');">
            <button type="submit" name="approve-all" id="approve-all" value="value" class="btn
btn-outline-success">Approve All</button>
        </form>
        <form method="POST" style="float:right;" th:action="@{/donations/discard-all}"
onsubmit="return confirm('Do you really want to discard all the requested donations?');">
            <button type="submit" name="discard-all" id="discard-all" value="value" class="btn
btn-outline-danger">Discard All</button>
        </form>
    </div>
</div>
```

```
</div>
</div>
</th:block>
</div>
<th:block th:replace="fragments/modal"></th:block>
<th:block th:replace="fragments/footer"></th:block>
<th:block th:replace="fragments/js"></th:block>
<script>
(function() {
    'use strict';
    window.addEventListener('load', function() {
        // Fetch all the forms we want to apply custom Bootstrap validation styles to
        var forms = document.getElementsByClassName('needs-validation');
        // Loop over them and prevent submission
        var validation = Array.prototype.filter.call(forms, function(form) {
            form.addEventListener('submit', function(event) {
                if (form.checkValidity() === false) {
                    form.classList.add('was-validated');
                    event.preventDefault();
                    event.stopPropagation();
                }
                else
                {
                    event.preventDefault();
                    event.stopPropagation();
                    var formData = new FormData($('.needs-validation')[0]);
                    // var formData = new FormData("this");
                    $.ajax({
                        url: "/donations/add",
                        method: "POST",
                        data: formData,
                        success: function (data, textStatus)
                        {
                            $("#success_tic").modal();
                            $("#req-donations").replaceWith(data); // upd
                        },
                        contentType: false,
                        processData: false,
                        cache: false,
                    });
                    $("#removeImage").click();
                    form.classList.remove('was-validated');
                    form.reset();
                }
            }, false);
        });
    })();
    $(document).keydown(function(event) {
        if (event.keyCode == 27)
        {
            $('.close').click();
        }
    });
    function readURL(input)
    {
        if (input.files && input.files[0]) {
            var reader = new FileReader();
            reader.onload = function(e) {
                $('.image-upload-wrap-donations').hide();

                $('.file-upload-image-donations').attr('src', e.target.result);
                $('.file-upload-content-donations').show();

                $('.image-title-donations').html(input.files[0].name);
            };
        }
    }
})()
```

```
    reader.readAsDataURL(input.files[0]);
} else {
    removeUpload();
}
}
function removeUpload() {
    $('.file-upload-input-donations').replaceWith($('.file-upload-input-
donations').clone());
    $('.file-upload-content-donations').hide();
    $('.image-upload-wrap-donations').show();
}
$('.image-upload-wrap-donations').bind('dragover', function() {
    $('.image-upload-wrap-donations').addClass('image-dropping-donations');
});
$('.image-upload-wrap-donations').bind('dragleave', function() {
    $('.image-upload-wrap-donations').removeClass('image-dropping-donations');
});
</script>

</body>

</html>
```