|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Јован Попов

**Развој web апликацијa употребом .NET CORE 2.0 оквира**

МАСТЕР РАД

Нови Сад, 2018

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | | | | |
| КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА | | | | |
| Редни број, **РБР**: | | |  | | |
| Идентификациони број, **ИБР**: | | |  | | |
| Тип документације, **ТД**: | | | Монографска публикација | | |
| Тип записа, **ТЗ**: | | | Текстуални штампани документ | | |
| Врста рада, **ВР**: | | | Мастер рад | | |
| Аутор, **АУ**: | | | Јован Попов | | |
| Ментор, **МН**: | | | Милан Видаковић | | |
| Наслов рада, **НР**: | | | Развој web апликацијa употребом .NET CORE оквира | | |
| Језик публикације, **ЈП**: | | | Српски | | |
| Језик извода, **ЈИ**: | | | Српски / енглески | | |
| Земља публиковања, **ЗП**: | | | Република Србија | | |
| Уже географско подручје, **УГП**: | | | Војводина | | |
| Година, **ГО**: | | | 2018 | | |
| Издавач, **ИЗ**: | | | Ауторски репринт | | |
| Место и адреса, **МА**: | | | Нови Сад, Факултет тахничких наука  Трг Доситеја Обрадовића 6 | | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | | 6/53/ 0/0/14/0/16 | | |
| Научна област, **НО**: | | | Електотехничко и рачунарско инжењерство | | |
| Научна дисциплина, **НД**: | | | Web Програмирање | | |
| Предметна одредница/Кључне речи, **ПО**: | | | .NET Core, Angular, Twitter, Wikipedia, Allevents.in | | |
| **УДК** | | |  | | |
| Чува се, **ЧУ**: | | | Библиотека Факултета техничких наука  Трг Доситеја Обрадовића 6, 21000 Нови Сад | | |
| Важна напомена, **ВН**: | | |  | | |
| Извод, **ИЗ**: | | | У раду је анализиран .NET Core 2.0 оквир, и његова примена, у комбинацији са Angular оквиром, за развој web апликација. Специфицирани су захтеви и дизајн, имплементирана и тестирана је web апликација која прукупља податке о извођачима са Twitter, Wikipedia и Allevents.in API-ја. | | |
| Датум прихватања теме, **ДП**: | | |  | | |
| Датум одбране, **ДО**: | | |  | | |
| Чланови комисије, **КО**: | | Председник: | | Председник |  |
|  | | Члан: | | Члан | Потпис ментора |
|  | | Члан, ментор: | | др Милан Видаковић, редовни професор, Факултет техничких наука, Нови Сад |  |

Образац **Q2.НА.04-05** - Издање 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | UNIVERSITY OF NOVI SAD  **FACULTY OF TECHNICAL SCIENCES**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | | | |
| **KEY WORDS DOCUMENTATION** | | | |
| Accession number, **ANO**: | | |  | |
| Identification number, **INO**: | | |  | |
| Document type, **DT**: | | | Monographic publication | |
| Type of record, **TR**: | | | Textual material | |
| Contents code, **CC**: | | | Master thesis | |
| Author, **AU**: | | | Jovan Popov | |
| Mentor, **MN**: | | | Milan Vidaković | |
| Title, **TI**: | | | Development of web applications using .NET CORE framework | |
| Language of text, **LT**: | | | Serbian | |
| Language of abstract, **LA**: | | | Serbian / English | |
| Country of publication, **CP**: | | | Republic of Serbia | |
| Locality of publication, **LP**: | | | Vojvodina | |
| Publication year, **PY**: | | | 2018 | |
| Publisher, **PB**: | | | Author’s reprint | |
| Publication place, **PP**: | | | Novi Sad, Faculty of tehnical sciences,  Trg Dositeja Obradovica 2 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | | 6/53/ 0/0/14/0/16 | |
| Scientific field, **SF**: | | | Electrical and computer engineering | |
| Scientific discipline, **SD**: | | | Web app development | |
| Subject/Key words, **S**/**KW**: | | | .NET Core, Angular, Twitter, Wikipedia, Allevents.in | |
| **UC** | | |  | |
| Holding data, **HD**: | | | Library of the Faculty of technical sciences, Trg Dositeja Obradovića 6, Novi Sad | |
| Note, **N**: | | |  | |
| Abstract, **AB**: | | | This thesis analyses the .NET Core framework, and its use, in combination with the Angular framework, for the development of web applications. It specifies the requirements and the design and describes the implementation and the testing of a web app that collects data about artists from Twitter, Wikipedia and Allevents.in APIs. | |
| Accepted by the Scientific Board on, **ASB**: | | |  | |
| Defended on, **DE**: | | |  | |
| Defended Board, **DB**: | | President: | President |  |
|  | | Member: | Member | Menthor's sign |
|  | | Member, Mentor: | Milan Vidaković, PhD, full prof., Faculty of Technical Sciences, Novi Sad |  |

Obrazac **Q2.НА.04-05** - Izdanje 1

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Број: |
|  |
| **ЗАДАТАК ЗА МАСТЕР РАД** | Датум: |
|  |

*(Податке уноси предметни наставник - ментор)*

| СТУДИЈСКИ ПРОГРАМ: | Рачунарство и аутоматика |
| --- | --- |
| РУКОВОДИЛАЦ СТУДИЈСКОГ ПРОГРАМА: | Проф. Др Мирослав Поповић |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Јован Попов | Број индекса: | Е2 40/2016 |
| Област: | Веб програмирање | | |
| Ментор: | Др Милан Видаковић | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА  ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; | | | |

**НАСЛОВ МАСТЕР РАДА:**

|  |
| --- |
| Развој web апликацијa употребом .NET CORE 2.0 оквира |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Задатак рада је анализа .NET Core 2.0 оквира и његове примене, у комбинацији са Angular окружењем, за развој web апликација. Након анализе захтева биће реализован дизајн и урађена имплементација web апликације која прукупља податке о извођачима са Twitterа, Wikipediје и употребом Allevents.in API-ја. Спецификација система ће бити представљена употребом UML дијаграма. За реализацију серверског дела биће коришћен C# програмски језик, уз ослонац на .NET Core, а за реализацију клијентског дела биће коришћен програмски језик TypeScript, уз употребу Angular окружења. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

Образац **Q2.НА.04-03** - Издање 4

С А Д Р Ж А Ј

[1. Увод 9](#__RefHeading___Toc4946_1515439396)

[2. Коришћене технологије и терминологије 11](#__RefHeading___Toc1133_2716419765)

[2.1 .NET Core 11](#__RefHeading___Toc1530_1452494186)

[2.1.1 Композиција 11](#__RefHeading___Toc1532_1452494186)

[2.1.2. Језици 12](#__RefHeading___Toc1534_1452494186)

[2.1.3. .NET Стандард 12](#__RefHeading___Toc1536_1452494186)

[2.2 .NET Core 2.0 12](#__RefHeading___Toc4561_614418588)

[2.2.1. *Entity Framework Core* 13](#__RefHeading___Toc4563_614418588)

[2.2.2 Razor Pages 14](#__RefHeading___Toc4565_614418588)

[2.2.3 .NET Strandard 2.0 15](#__RefHeading___Toc4567_614418588)

[2.2.4 *SPA templates* 15](#__RefHeading___Toc4569_614418588)

[2.3 Angular 5 15](#__RefHeading___Toc4571_614418588)

[2.4 Twitter API 15](#__RefHeading___Toc9102_1878571053)

[2.5 Wikipedia API 16](#__RefHeading___Toc9104_1878571053)

[2.6 AllEvents.in API 16](#__RefHeading___Toc9106_1878571053)

[3. Спецификација апликације 17](#__RefHeading___Toc5919_4276884250)

[3.1. Дијаграм Случајева коришћења (*Use case*) 17](#__RefHeading___Toc4573_614418588)

[3.2. Дијаграм активности 18](#__RefHeading___Toc4575_614418588)

[3.3. Дијаграм шеме базе података 20](#__RefHeading___Toc4577_614418588)

[4. Имплементација 23](#__RefHeading___Toc5921_4276884250)

[4.1 *.NET Core 2.0 SPA* пројекат 23](#__RefHeading___Toc5923_4276884250)

[4.1.1 Структура апликације 24](#__RefHeading___Toc1143_2716419765)

[4.1.2 Конфигурација пројекта 26](#__RefHeading___Toc4579_614418588)

[4.1.3 Identity Framework 28](#__RefHeading___Toc4581_614418588)

[4.1.4 Рад са базом 30](#__RefHeading___Toc4583_614418588)

[4.1.5 Имплементација претраге 32](#__RefHeading___Toc4585_614418588)

[4.1.6 Wikipedia API 35](#__RefHeading___Toc4589_614418588)

[4.1.7 Twitter API 36](#__RefHeading___Toc4591_614418588)

[4.1.8 AllEvents.in API 36](#__RefHeading___Toc4593_614418588)

[4.1.8 Angular клијентска апликација 37](#__RefHeading___Toc4595_614418588)

[5. Демонстрација 45](#__RefHeading___Toc5951_4276884250)

[6. ЗАКЉУЧАК 49](#__RefHeading___Toc5953_4276884250)

[ЛИТЕРАТУРА 51](#__RefHeading___Toc4831_3121367626)

[БИОГРАФИЈА 53](#__RefHeading___Toc4833_3121367626)

# Увод

Микрософтова .NET платформа [1] постоји још од 2002. године, и њена библиотека ASP.NET [2] je тренутно једна од водећих технологија које се употребљавају за развој web апликација, поред и других функционалности које .NET пружа, као што су мобилне и десктоп апликације. Од 2016. године, паралелно са развојем тренутно стабилне верзије .NET bibliotekе, која је тренутно у употреби у комерцијалним пројектима, Микрософт је започео развој најновије верзије њихове библиотеке, .NET Core, а заједно са њиме новог стандарда за .NET платформе, који се зове .NET standard [3]. .NET Core је развијан од нуле, уместо да се надовезују на претходне верзије, и он је за разлику од претходних имплементација, *open source* и такође *cross-platform*. Тренутно, актуелна верзија .NET Core библиотеке je .NET Core 2.0, издата 14. августа 2017. године.

У првом поглављу је појашњен проблем који се посматра у пројекту. У следећем су описане технологије које се посматрају и имплементирају. У трећем је наведена спецификација и у четвртом је описана имплементација. У последњем поглављу је дат закључак и споменуте су могућности везане за даље истраживање ове технологије и дање проширење пројекта.

# Коришћене технологије и терминологије

У овом поглављу ће бити детаљније описане технологије коришћене за реализацију пројекта. Пројекат је у основи .NET Core 2.0 web апликација, са API сервисима, која за свој *front-end* користи Angular 5 апликацију. Најновија верзија Visual Studio окружења подржава темплејте приликом започињања новог пројекта, који већ у себи имају подешену овакву структуру пројекта.

## 2.1 .NET Core

.NET Core [4] је развојна платформа која се одржава од стране Микрософта и .NET заједнице. Cross-platform је, и подржава Windows, macOS и Linux. Основне карактеристике које дефинишу .NET Core су:

* ***Cross-Platform* подршка**. Тренутно подржава горе наведене оперативне системе, али временом је могуће даље проширење подршке од стране Микрософта и заједнице.
* **Командни алати**. Омогућена *command-line* подршка за све продукционе сценарије.
* **Компатибилност**. .NET Core је компатибилан са .NET framework-om, Xamarin-om и Mono-om, пошто подржава, односно имплементира .NET стандард.
* ***Open source***. .NET Core платформа је *open source*, и користи MIT и Apache 2 лиценце.
* **Подршка од стране Микрософта**. .NET Core платформа је подржана од стране Микрософта, према њиховој полиси .NET Core подршке.
* ***Dependency Injection.*** Претходно је било потребно користити додатну библиотеку за ову функционалност, сада се она налази у основи дизајна *framework*-a. Поред сервиса које нуди само окружење, који се региструју преко DI, као што су *Entity Framework* за рад са базом, на пример, корисник може лако своје сервисе да дода у исти *DI container.*
* ***ASP.NET* - *MVC* и *WebApi.*** У претходним верзијама, MVC и webApi су били раздвојени, сад су спојени у исти *namespace*, и такође контролери деле исту класу, дакле нема више засебних класа за MVC и WebApi контролере.

### 2.1.1 Композиција

.NET Core се састоји од следећих делова:

* ***.NET Runtime***, који пружа систем типова, учитавање асемблија, *JIT (just in time)* компајлер, *garbage collection*, и сличне основне сервисе.
* **Основни сет *framework* библиотека**, које пружају примитивне типове података, класе за колекције, и друге фундаменталне функционалности.
* **SDK алати и компајлери језика**, које омогућавају основно развојно искуство, који су доступни у .NET Core SDK.
* ***‘Dotnet’ app host,*** који се користи за покретање .NET Core апликација.

### 2.1.2. Језици

C#, Visual Basic и F# су језици који се могу користити за писање апликација за .NET Core. Компајлери се покрећу на .NET Core платформи, што омогућава развој за .NET Core где год он може да се покрене.

### 2.1.3. .NET Стандард

Један од изазова код тренутног развоја .NET-а је тај да се временом дошло до више .NET имплементација. Постоји .NET framework [5], Xamarin [6], Mono [7], Core, и сваки има своје API-је. Приликом развоја Core-a, дошло се до жеље да .NET буде конзистентан на свим платформама, па је развијен .NET стандард, који је спецификација за .NET. Што значи да уколико неко жели да направи .NET платформу која задовољава стандард, онда мора да имплементира сет API-ja прописаних стандардом, који програмери могу да очекују од .NET платформе. Употребом овог стандарда, код може лако да се преноси између различитих .NET платформи.

## 2.2 .NET Core 2.0

.NET Core 2.0 доноси са собом следећа побољшања и алате:

* ***Оптимизација.*** 2.0 верзија је за око 20% бржа од 1.1 верзије
* ***Entity Framework Core 2.0***
* ***Razor pages***
* ***ASP.NET Core metapackage,*** који садржи све Core библиотеке у себи, које су се у претходним верзијама морале засебно додавати у пројекат.
* ***.NET Standard 2.0***
* ***SPA templates***
* ***Kestrel web server improvements***
* ***Razor view engine подржава C# 7.1***

### 2.2.1. *Entity Framework Core*

*Entity Framework* је библиотека која је део ASP.NET оквира, и она се користи за рад са базом. *Entity Framework Core 2.0* је најновија верзија ове библиотеке, и она је такође cross-platform и такође подржава .NET 2.0 стандард, и доноси са собом генерално боље перформансе, а такође и додатна побољшања, као што су:

- Побољшан превод *Linq* израза у Sql упите.

- Употреба параметризације уколико се позива Sql упит написан помоћу string интерполације. На пример:

var city = "Redmond";

using (var context = CreateContext())

{

context.Customers.FromSql($@" SELECT \* FROM Customers WHERE City = {city}");

}

Листинг 1: Стринг интерполациција за SQL упит

Оно што ће се у овом случају десити у позадини, је да ће бити направљен параметар @p0 , који ће имати вредност „Redmond“, и затим Sql упит који ће изгледати овако:

SELECT \* FROM Customers WHERE City = @p0

Листинг 2: Добијен SQL упит са параметром

Ово нам пружа заштиту против *Sql injection*-a.

Да се не би правио нов DataBaseContext сваки пут кад дође захтев на сервер, додата је опција коришћења *pool-а* у ком се држе инстанцирани контексти, и из њега се добављају при сваком позиву. Потенцијална мана је, да ако корисник дода своје стање унутар контекста, које је везано за један захтев, то стање ће остати упамћено што може довести до проблема кад се та инстанца поново користи приликом следећег позива. Ово је разлог из којег ова опција није укључена одмах, већ корисник мора ручно да је подеси.

Мана рада са *Entity Framework Core 2.0 верзијом,* и такође највећа мана Core оквира тренутно, је да просто није још увек завршен, односно, нису још увек имплементиране све функционалности које је претходна верзија .NET оквира имала. Па тако, верзија *Entity Framework*-a која се користи у *.NET framework*-у има веома квалитетну подршку за *code-first* и такође *database-first* рад са базом, са веома корисним дизајнером, који може да прикаже све табеле и њихове везе унутар самог *Visual Studio* okruжења. *Entity Framework Core 2.0* са друге стране, тренутно првенствено подржава *code-first* приступ, док је могуће изгенерисати готове моделе и DatabaseContext од постојеће базе, али је тренутно та опција имплементирана само као конзолни алат, који се позива из фолдера у којем се пројекат налази.

Са друге стране, *code-first* приступ је одлично имплементиран. Приликом писања класа на које ће се мапирати модели, није неопходно користити анотације да би се назначиле карактеристике те табеле, пошто се у позадини користе разни чести шаблони да се аутоматски препозна да је, на пример, Id *property* примарни кључ у табели. Након што се направи модел табела, и DatabaseContext класа у којој се назначе сви модели, могу се вршити додатне измене и подешавања, потребно је употребом команди у *Package Manager* конзоли у *Visual Studiо* окружењу прво направити миграцију, чиме се добије C# фајл који се извршава позивом команде за освежавање базе, чиме се врше измене у бази наспрам модела који смо написали у коду. Оно што је овде корисно, је да се миграциони фајлови, који су C# класе, налазе у *migrations* фолдеру унутар пројекта, и тако се могу налазити на *Git*-у, заједно са осталим фајловима. Такође је могуће у конфигурацији пројекта додати поѕив методе EnsureCreated(), над DatabaseContext-ом, да би се база направила аутоматски према моделу, без ручне употребе миграција.

### 2.2.2 Razor Pages

*Razor pages* [8] представља нов начин за прављење web апликација. У позадини је заснован на добро познатом MVC-у, и основна замисао је да се додатно поједностави израда web страница, дакле да се на брз и једноставан начин прикажу подаци. Пошто је заснован на MVC-у, подржава исте функционалности, и такође може да се користи паралелно са MVC web апликацијом. Састоји се од страница, које динамички генеришу HTML садржај, и свака од тих страница је повезана са својом C# класом, која је слична контролеру из MVC-a. Страница има приступ промељивама које су дефинисане у класи, а класа сем тога, може и да садржи Http *endpoint*-e и помоћне функције.

### 2.2.3 .NET Strandard 2.0

.NET Core 2.0 долази заједно са најновијом и тренутно актуелном верзијом .NET стандарда, .NET standard 2.0. Око 70% NuGet пакета треутно доступних подржава овај стандард, што значи да могу да се користе приликом развоја .NET Core 2.0 апликација. Они који га не подржавају су углавном старији пакети, који референцирају *Windows form*-e, WPF, или сличне технологије које нису доступне на свим платформама.

### 2.2.4 *SPA templates*

ASP.NET библиотека је одлично решење за развој MVC web апликација. У исто време, *Single Page* апликацијепостају све популарнији pristup, па тако Микрософт жели да подржи и тај приступ заједно са већ постојећим MVC и *Razor Pages* опцијaмa, па прилим започињања новог пројекта у *Visual Studio* окружењу, постоје темплејти за развој .NET Core 2.0 апликација са свим овим приступима, дакле MVC, Razor Pages, и на крају, SPA темплејти. Конкренто, тренутно постоји темплејт који користи .NET Core 2.0 за REST API сервисе, а *front-end* jе реализован потребом *Angular* библиотеке, и такоће сличан темплејт са употребом *React* библиотеке. Дакле постоји подршка за неке од тренутно најпопуларнијих SPA библиотека.

## 2.3 Angular 5

*Angular* [9]је тренутно једна од најпопуларнијих JavaScript библиотека за прављење *Single Page* апликација. Прва верзија библиотеке се звала AngularJs, и она је и даље веома популарна и користи се веома често. Верзија 2.0 је донела много измена, за почетак, употребу *TypeScript*-а, затим контролери су замењени са компонентама, и уз доста додатних измена, Angular 2 (без „Js“ у свом називу) је постао практично нови оквир. Тренутно актуелна верзија је 5, мада од верзије 2 па на даље, измене нису толико радикалне.

## 2.4 Twitter API

Twitter API омогућава различите функционалности везане за Twitter платформу. Конретно, он омогућава операције над налозима, претраживање Tweet-ova по различитим параметрима, слање порука и додатне функционалности. Да би се користио, неопходно је регистровати се на њиховом порталу, након чега се добију кључеви помоћу којих ће се ауторизовати позиви API-ја.

## 2.5 Wikipedia API

Подаци са Wikipedije се могу преузети употребом MediaWiki API-ја. АPI се користи употребом *query* параметара, као што су „action“ параметар, који може да прима различите вредности, од којих је најбитнија „query“, који назначава да желимо да вршимо претрагу садржаја, затим постоји „format“ параметар, са којим се може назначити формат одговора. Следећи битан параметар је „prop“, што је скраћено од „property“, којим се може може назначити које конкретне информације о одређеној страници нам требају, док „titles“ параметар, служи да проследимо назив странице коју желимо да пронађемо.

## 2.6 AllEvents.in API

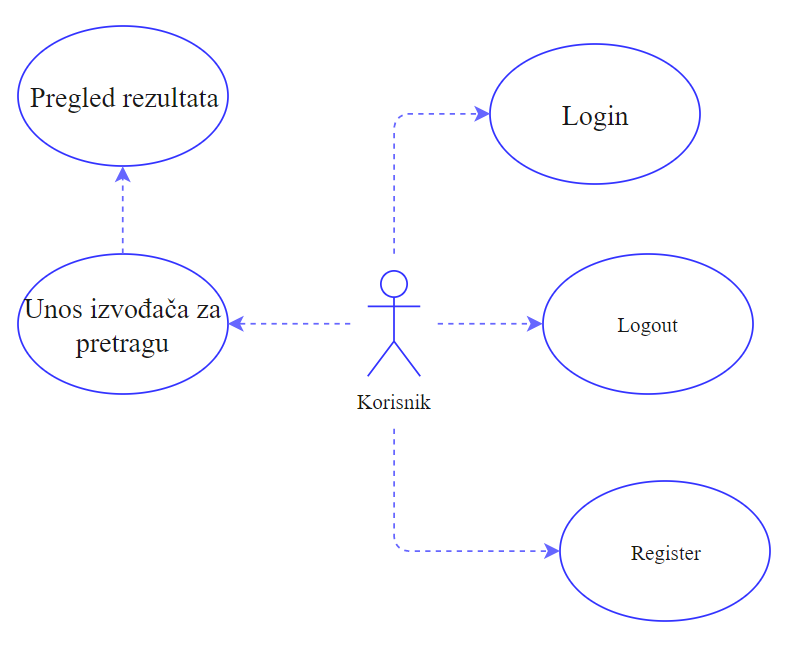
AllEvents.in је сајт који прикупља Facebook догађаје, и чува их у својој бази. Он је посебно користан звог свог API-ja, који је корисна алтернатива за Facebook API, са којим је већ неко време добављање догађаја изузетно тешко. AllEvents.in API је још увек у приватној бета фази, и потребно је регистровати се, и сачекати да се одобри регистрација. Затим се може добити API кључ који ће се користити приликом позива API-ја и такође Url за портал за рад са API-jem. Примери позива за више програмских језика се могу видети на порталу. API омогућава претрагу догађаја по називу, по географској локацији, по граду и по организатору.

# Спецификација апликације

Задатак обухвата израду апликације која је заснована на .NET Core 2.0 технологији, која се користи као основа web апликације, за REST API, и за чување података у базу, док је *front-end* реализован помоћу Angular 5 оквира. Корисник може да се региструје, пријави, одјави, и претражује извођаче помоћу поља за претрагу. Након уноса извођача, користе се Allevents.in [10], Twitter [11], и Wikipedia [12 ] API да би се добавиле информације о њима и приказале. Идеја пројекта је да се резултати претраге одређеног извођача чувају у базу, и следећи пут кад се исти извођач претражује унутар одређеног временског периода, уместо да се опет врше позиви свих API-ја, подаци се прочитају из базе и прикажу знатно брже.

## 3.1. Дијаграм Случајева коришћења (*Use case*)

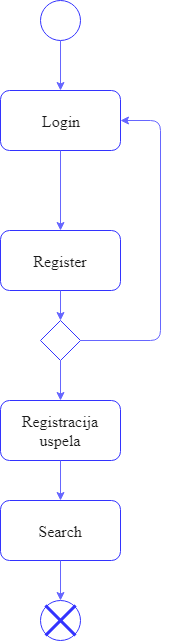
Дијаграм случајева коришћења представља све акције корисник може да изврши и приказан је на слици 1.



Слика 1. Дијаграм случајева коришћења

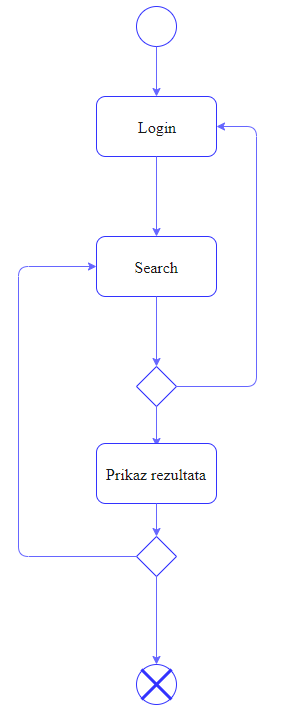
## 3.2. Дијаграм активности

Дијаграми активности су намењени моделирању понашања програма. Они приказују секвенцијалне и конкуретне кораке у извршавању програма. Активности везане за коришћење апликације се налазе на сликама 2 и 3.



Слика 2: Дијаграм регистрације

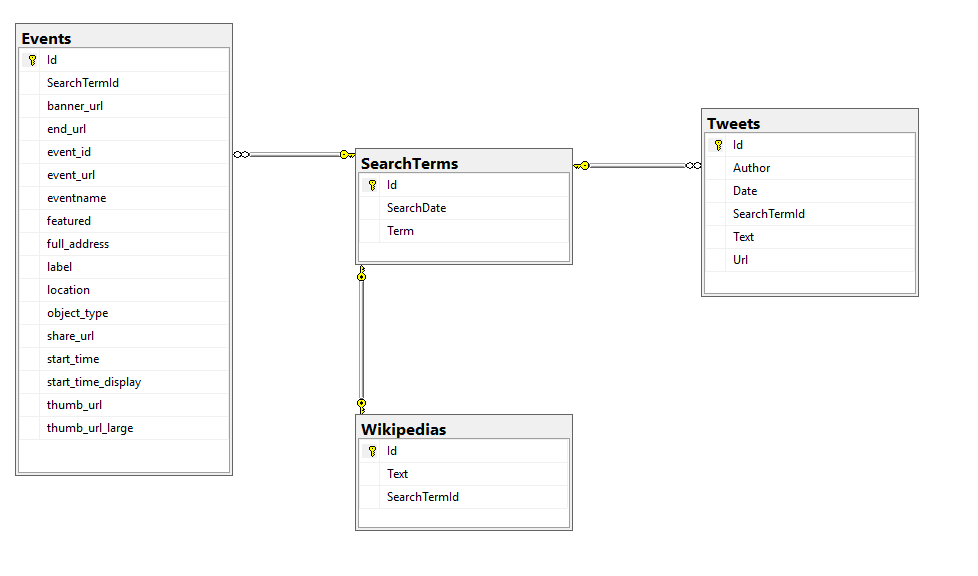
Да би се могле користити функционалности сајта, корисник мора бити регистрован у систему, и затим пријављен. Уколико покуша да приступи страници за претрагу а није пријављен, биће преусмерен на страницу за пријаву, на којој се налази линк за регистрацију. Уколико је регистрација успешна, корисник се преусмерава на страницу за претрагу.



Слика 4. Дијаграм претраживања извођача

Након успешне пријаве, корисник може да приступи страници за претрагу извођача. На њој уноси назив извођача у поље за претрагу и резултати се приказују испод поља. Резултати за сваки засебни API су подељени по страницама уколико их има више, и увек је могућ унос новог извођача што ће освежити табеле са резултатима. Корисник се може у било ком момeнту вратити на страницу за пријаву кликом на Logout.

## 3.3. Дијаграм шеме базе података

 За потребе пројекта коришћена је MS SQL Server база података. Она је потребна за управљање корисничким налозима, и такође за чување података које смо добили за различитих API-ja приликом претраге. Управљање корисничким налозима је реализовано упоребом *Identity Framework* библиотеке, које је део ASP.NET-а, и која са собом доноси све табеле неопходне за кориснике, улоге, и остале функционалности. Модел података који се користи за чување резултата претраге се налази на слици 5.

Слика 5. Дијаграм модела података

**SearchTerms** је основни ентитет, који представља једну претрагу корисника. Ова табела се састоји од SearchDateпоља, које представља време кад је претрага извршена, и Termпоља, које представља унос претраге корисника.

За ову табелу су везане остале табеле, које представљају податке добављене са различитих API-ja.

**Events** ентитетипредстављају модел једног догађаја добављеног са Allevents.in API-ja. У овој табели се чувају различити подаци који се добаве везани за догађај, као што су адреса, локација, Url, време почетка догађаја и тако даље. Један SearchTerm може имати више догађаја везаних за њега. SearchTermId је страни кључ.

**Tweets** ентитети представљају модел једног tweet-a добављеног преко Twitter API-ja, и састоји се од назива аутора, Датума, текста и Url адресе самог tweet-a. Један SearchTerm може имати више Tweet-ova, SearchTermId је страни кључ.

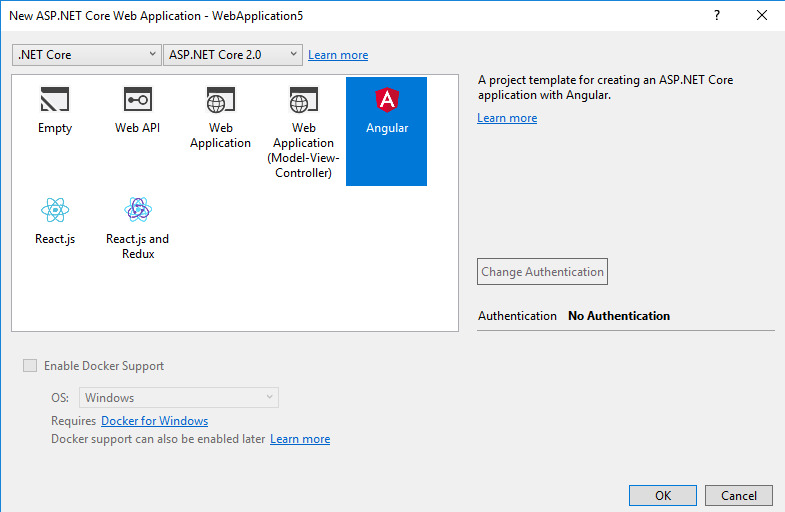
**Wikipedias** ентитети представљају модел податка добављеног преко Wikipedia API-ја. Састоји се од текстра сажетка са wikipedia странице о извођачу. Један SearchTerm може имати један Wikipedia објекат. SearchTermId је страни кључ.

# Имплементација

Ово поглавље описује кључне делове имплементације пројекта. Пројекат је у основи .NET Core 2.0 web апликација, и основна намена му је истраживање ове технологије. Core се првенствено користи као *back-end*, дакле за имплементацију REST API сервиса, док се за *front-end* користи Angular клијентска апликација. Користи се MS SQL Server база, за рад са базом се користи *Entity Framework Core* 2.0, дакле најновија верзија Микрософтовог *Entity Framework* оквира за рад са базом. За управљање корисничким налозима, регистрацију, пријаву, чување корисника у бази, користи се *Identity Framework*, дакле Микрософтова библиотека за ову сврху, која је део ASP.NET-a, и чија најновија верзија долази са Core 2.0 издањем. Пројекат користи шаблон који такође долази са Core 2.0 издањем, који омогућава лако започињање .NET Core web пројеката који користи Angular као клијентску апликацију. Пројекат је развијан у *Visual Studio* развојном окружењу.

## *.NET Core 2.0 SPA* пројекат

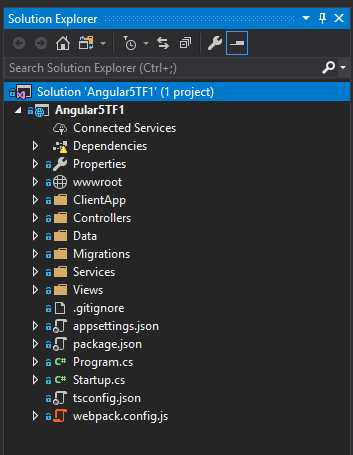
Приликом започињања новог ASP.NET Core Web Application пројекта, најновиja верзија *Visual Studio* окружења нуди следеће опције:

Слика 6. Нови ASP.NET Core web application пројекат

Овде се може одабрати Angular шаблон, који ће нам дати основу апликације која користи ове две технологије. Конрентно, начин на који овај шаблон функционише, је да је се унутар Core 2.0 web апликације налази фолдер са једноставном Angular клијентском апликацијом, који се састоји од TypeScrypt фајлова који морају да се *transpile*-ују, односно преведу у JavaScript фајлове, што корисник не мора ручно да ради, већ се ово аутоматски уради приликом покретања, односно компајлирања .NET пројекта који се налази у основи. Начин на који је ово урађено је употребом .NET CORE *middleware*-a која се зове *Node Services*, који омогућава комуникацију са Node.js-om из C# кода, који врши превођење TypeScript фајлова. Овај шаблон такође корiсти WebPack, који се користи да би оптимизовао учитавање JavaScript, CSS и HTML фајлова са сервера на претраживач. Ова библиотека такође омогућава „*hotswap*“ функционалност приликом рада са пројектом, која док је пројекат покренут, слуша у позадини да ли је дошло до неке промене у коду, и у колико јесте, преко *Node Services* middleware-a поново искомпајлира клијентски део апликације и аутоматски освежи web апликацију, без потребе да се сам пројекат рестартује. Ово је веома корисна функционалност која додатно олакшава рад. Једина мана овог официјалног шаблона је што користи Angular 4, мада постоји и неофицијална верзија шаблона која користи најновију, Angular 5, верзију Angular библиотеке.

### 4.1.1 Структура апликације

На слици 7 се може видети структура завршене апликације.

Слика 7. Структура апликације

Битни елементи ове структуре су:

* *Dependencies –* Садржи зависности, односно референце на библиотеке које се користе. *NuGet* се користи за добављање библиотека за серверски део апликације.
* *Properties* – Садржи подешавања везана за покретање пројекта, као што је покретање претраживача приликом покретања пројекта.
* *wwwroot –* основни фолдер web апликације, који корсти IIS апликативни сервер. Улога овог фолдера је да садржи статичке фајлове, дакле HTML, CSS, JavaScript фајлове, слике и тако даље.
* *ClientApp –* Angular клијентска апликација се налази у овом фолдеру.
* *Controllers –* Садржи контролере, које се користе за WebApi *endpoint*-ове, и такође за почетну страницу апликације, унутар које се приказује Angular садржај.
* *Data –* Садржи фајлове везане за рад са базом, и такође DTO моделе за webApi. Овде се налазе модели, *Database context* који користи *Identity Framework* за рад са корисницима, и такође *Database context* за чување података добављених са API-ја.
* *Migrations* – Садржи миграционе фајлове који су генерисани коришћењем команди за миграцију.
* *Services –* Садржи сервис који у себи има логику позива различитих API-ја који се позивају у овом пројекту. Овај сервисе се користи преко уграђеног *Dependecy Injecton* контејнера.
* *Views* – Фолдер који садржи Razor view странице MVC оквира ASP.NET-a. У овом пројекту, потребна је само једна, почетна страница, која ће садржати Angular апликацију у себи.
* *appsettings.json –* json фајл који се користи за додатну конфигурацију .NET Core апликације. Такође служи за дефинисање константи у пројекту, као што су кључеви и токени који се користе приликом позива API-ja. Ове константе се после добављају употребом постојећих сервиса за добављање константи.
* *package.json –* фајл који користи NPM да би добавио библиотеке које се користе за клијентски део апликације.
* *Program.cs -*  је улазна тачка сваког .NET Core пројекта, у њему се дефинише покретање самог пројекта.
* *Startup.cs –* конфигурација пројекта. Овде се, на пример, дефинишу различити сервиси и функционалности које се користе, везане за базу, за рад са корисницима, додају се ручно направљени сервиси у DI контејнер.
* *tsconfig.json –* конфигурацију TypeScript компајлера, ово је потребно окружењу да би Angular апликација могла да се компајлира.
* *webpack.config.js –* конфигурациони фајл webpack алата.

### 4.1.2 Конфигурација пројекта

Почетна конфигурација пројекта се врши у Startup.cs фајлу. У њему се додају функционалности, као што су употреба MVC-а на пример, и такође се региструју сервиси употребом *Dependency Injection* функционалности, која је у .NET Core оквиру већ уграђена. Битнe методе у овој класи који се позивају од стране *runtime*-a су Configuration и ConfigureServices. ConfigureServices метод је место где се региструју сервиси, контексти за базу, функционалности као што су MVC, *Indentity Framework* који се користи за рад са корисницима, а Configuration метод се користи за додатно конфигурисање сервиса, на пример за додатно подешавање MVC функционалности. Осим наведеног, користи се и за реализацију *hotswap* функционалности, која нам омогућава аутоматско освежавање пројекта кад се код у позадини промени, помоћу *webpack* библиотеке, је такође овде конфигурисана, што се може видети на листингу 3.

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseWebpackDevMiddleware(new WebpackDevMiddlewareOptions

{

HotModuleReplacement = true

});

}

Листинг 3: *hotswap* конфигурација

Пројекат користи два контекста за базу, један је повезан са Identity оквиром (ApplicationDbContext), који сам направи табеле у бази преко којих чува регистроване кориснике и друге операције везане за њих, док је други контекст везан за податке који се чувају у базу за потребе овог пројекта (DataContext), што су подаци који се добављају преко API-ja. Регистровање контекста се може видети на листингу 4.

// ===== ApplicationDbContext ========

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connection));

// ===== DataContext ========

services.AddDbContext<DataContext>(options =>

options.UseSqlServer(connection));

Листинг 4: Регистровање контекста за базу

Такође, сервиси направљени за потребе пројекта се могу регистровати у овој секцији, да би били доступни кроз DI механизам. У овом пројекту, зарад примера, направљен је сервис који у себи садржи све позиве спољних API-ја, и он се овде у конфигурацији региструје, што се може видети на листингу 5.

services.AddTransient<IApisService, ApisService>();

Листинг 5: Регистровање сервиса

Након тога, сервис нам је доступан преко DI контејнера, што значи да му можемо приступити инјектовањем преко конструктора, што се може видети на листингу 6.

public SearchController(IConfiguration configuration, DataContext dataContext, IApisService apisService)

{

context = dataContext;

\_configuration = configuration;

\_apisService = apisService;

}

Листинг 6: Dependency Injection

### 4.1.3 Identity Framework

Да би Identity сервис функционисао, потребно је да га додатно конфигуришемо у методи за конфигурисање, што се може видети на листингу 7, и такође направити методе за регистровање и пријаву. Након тога, он омогућава аутентификацију и ауторизацију корисника додавањем [Authorize] анотација на API контролере, које такође као параметре могу да имају, на пример, неопходну улогу корисника.

// ===== Add Identity ========

services.AddIdentity<ApplicationUser, IdentityRole>( options => {

options.Password.RequiredLength = 6;

options.Password.RequireLowercase = false;

options.Password.RequireUppercase = false;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequireDigit = false;

})

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

// ===== Add Jwt Authentication ========

JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();

services

.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(cfg =>

{

cfg.RequireHttpsMetadata = false;

cfg.SaveToken = true;

cfg.TokenValidationParameters = new TokenValidationParameters

{

ValidIssuer = Configuration["JwtIssuer"],

ValidAudience = Configuration["JwtIssuer"],

IssuerSigningKey = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(Configuration["JwtKey"])),

ClockSkew = TimeSpan.Zero

};

});

Листинг 7: Конфигурација Identity сервиса и JWT

Након тога, потребно је направити методе за пријаву и регистрацију у контролеру. Метод за регистрацију се може видети на листингу 8, а метод за пријаву на листингу 9.

[HttpPost]

public async Task<IActionResult> Register([FromBody] UserDto model)

{

var user = new ApplicationUser

{

UserName = model.Username,

Email = model.Username

};

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false);

return Ok(GenerateJwtToken(model.Username, user));

}

else

{

string errors = "";

foreach(var item in result.Errors)

{

errors += item.Description + " ";

}

return BadRequest(errors);

}

}

Листинг 8: Метод за регистрацију

[HttpPost]

public async Task<IActionResult> Login([FromBody] UserDto model)

{

var result = await \_signInManager.PasswordSignInAsync(model.Username, model.Password, false, false);

if (result.Succeeded)

{

var appUser = \_userManager.Users.SingleOrDefault(r => r.Email == model.Username);

var token = GenerateJwtToken(model.Username, appUser);

return Ok(new

{

Id = appUser.Id,

Username = appUser.Email,

FirstName = appUser.UserName,

Token = token

});

}

return Unauthorized();

}

Листинг 9: Метод за пријаву корисника

### 4.1.4 Рад са базом

* + 1. За рад са базом се користи *Entity Framework* ORM (*Оbject-relational mapping*) оквир. Основна класа преко које се врши интеракција је DatabaseContext. Као што је већ речено, постоје две врсте контекста у .NET оквиру, IdentityDbContext и DbContext. IdentityDbContext користи Identity framework за рад са корисницима, и он се додаје у пројекат тако што се направи класа која наслеђује IdentityDbContext класу, и она се региструје у startup конфигурацији. Ова класа се може видети на листингу 10.
    2. public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

// Customize the ASP.NET Identity model and override the defaults if needed.

}

* + 1. }
    2. Листинг 10: ApplicationDbContext класа
    3. IdentityDbContext класа овде прима као генерички параметар класу ApplicationUser, са којом смо наследили класу IdentityUser, која представља основну класу за корисника у Identity библиотеци. Уколико бисмо желели да додамо неке додатне параметре везане за корисника, поред основних које садржи IdentityUser класа, можемо их додати у ApplicationUser класи. Након што је ова класа додата у пројекат и регистрована, потребно је урадити миграцију, да бисмо добили табеле везане за кориснике и њихове улоге у бази.

Други тип контекста у .NET оквиру је DbContext, и он се користи за ентитете у које се чувају подаци у апликацији који нису везани за кориснике. Оно што је прво битно направити, је класу која наслеђује DbContext, и у њој је неопходно додати као *property*-је колекције типа DbSet, које представњају ентитете који ће се чувати у бази. Такође је могуће додатно конфигурисати моделе у овој класи, на пример, подесити каскадно брисање као понашање приликом брисања ентитета. Ова класа се може видети на листингу 11.

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options) { }

public DbSet<SearchTerm> SearchTerms { get; set; }

public DbSet<Event> Events { get; set; }

public DbSet<Wikipedia> Wikipedias { get; set; }

public DbSet<Tweet> Tweets { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<SearchTerm>()

.HasOne(p => p.Wikipedia)

.WithOne(b => b.SearchTerm)

.OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<SearchTerm>()

.HasMany(p => p.Events)

.WithOne(b => b.SearchTerm)

.OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<SearchTerm>()

.HasMany(p => p.Tweets)

.WithOne(b => b.SearchTerm)

.OnDelete(DeleteBehavior.Cascade);

}

}

Листинг 11: DataContext класа

Класа која представља модел податка је стандардна POCO (*Plain old c# object*) класа, и притом, није потребно користити анотације да би се тачно назначило који property је кључ, или страни кључ, пошто је библотека у стању да сама протумачи садржај класе и из ње закључи њене карактеристике, користећи често коришћене шаблоне и конвенције за називе, мада се могу такође и анотације користити уколико постоји потреба. Пример једне овакве класе се може видети на листингу 12.

public class Tweet

{

public int Id { get; set; }

public DateTime Date { get; set; }

public string Text { get; set; }

public string Url { get; set; }

public string Author { get; set; }

public SearchTerm SearchTerm { get; set; }

}

Листинг 12: Tweet класа

Из ове класе, Identity је у стању сам да закључи како да назове поља и ког су типа. Затим, може да закључи да је property који се зове Id, и типа је integer, примарни кључ. Такође, из property-ја „public SearchTerm SearchTerm { get; set; }“, библиотека може да закључи везу између Tweet и SearchTerm ентитета, и у позадини ће направити додатно поље у Tweet табели, које ће се звати SearchTermId и које ће бити страни кључ. Након овога, потребно је у Package Manager конзоли прво укуцати „Add-Migration“, да би се миграција направила, и затим „Update-database“ да би се промене извршиле над базом. Након што смо ово све подесили, објектима из базе се приступа преко контекста, дакле преко DataContext класе, која нам је доступна преко Dependency Injection механизма. Овај објекат затим у себи садржи све DbSet-ове који представљају ентитете из базе, и којима можемо приступити, и затим вршити CRUD операције над њима, и такође приступати објектима са којима постоје везе. Све ове операције су већ имплементиране као део Identity библиотеке, помоћу Linq оператора, као што су Select, OrderBy, и тако даље. Пример једног упита се може видети на листингу 13.

SearchTerm searchTerm = context.SearchTerms.Where(x => x.Term == data).FirstOrDefault();

Листинг 13: Пример упита

Ово ће се у позадини претворити у Sql упит који ће се извршити над базом, који ће добавити SearchTerm чије „Term” поље је једнако вредности „data” стринга. На сличан начин можемо брисати ентитете, додавати нове, и тако даље.

### 4.1.5 Имплементација претраге

Претрага је имплементирана у класи SearchController, који садржи API *endpoint* који враћа резултате претраге. Претрага функционише тако што се за унесен појам врши претрага три позната API-ја: Wikipedia, Twitter и Allevents.in. Логика је да се све претраге чувају у бази, и уколико се поново тражи исти појам унутар одређеног временског оквира, приказаће се резултати из базе, што је знатно брже него да се поново подаци добављају преко API-ја. Уколико појам не постоји у бази, или постоји али је превише стар, прво ће се појам обрисати, и онда ће се извршити позив API-ja. Имплементација ове логике се може видети на листигну 14. Помоћне методе за чување података у базу се могу видети на листингу 15.

SearchTerm searchTerm = context.SearchTerms.Include(x => x.Wikipedia).Include(x => x.Events).Include(x => x.Tweets).Where(x => x.Term == data).FirstOrDefault();

if(searchTerm != null &&

(DateTime.UtcNow - searchTerm.SearchDate).TotalMinutes > 1)

{

context.SearchTerms.Remove(searchTerm);

searchTerm = null;

}

if (searchTerm == null)

{

//search term does not exist in database

searchTerm = new SearchTerm();

searchTerm.Term = data;

searchTerm.SearchDate = DateTime.UtcNow;

//apis

List<Data.DTO.AlleventsDto.Event> events = \_apisService.Allevents(data);

string wikiText = \_apisService.Wikipedia(data).Result;

List<TwitterStatus> tweets = \_apisService.Twitter(data);

//database

WikiDb(searchTerm, wikiText);

EventsDb(searchTerm, events);

TweetesDb(searchTerm, tweets);

context.SearchTerms.Add(searchTerm);

context.SaveChanges();

}

Листинг 14: Имплементација претраге у Search контролеру

private void WikiDb(SearchTerm newTerm, string wikiText)

{

Wikipedia wiki = new Wikipedia();

wiki.Text = wikiText;

newTerm.Wikipedia = wiki;

}

private void EventsDb(SearchTerm newTerm, List<AlleventsDto.Event> events)

{

List<Data.Model.Event> dbEvents = new List<Data.Model.Event>();

dbEvents = events.Select(x => new Data.Model.Event {

banner\_url = x.banner\_url,

end\_url = x.end\_url,

eventname = x.eventname,

event\_id = x.event\_id,

event\_url = x.event\_url,

featured = x.featured,

full\_address = x.venue != null ? x.venue.full\_address : "",

label = x.label,

location = x.location,

object\_type = x.object\_type,

share\_url = x.share\_url,

start\_time = x.start\_time,

start\_time\_display = x.start\_time\_display,

thumb\_url = x.thumb\_url,

thumb\_url\_large = x.thumb\_url\_large

}).ToList();

newTerm.Events = dbEvents;

}

private void TweetesDb(SearchTerm newTerm, List<TwitterStatus> tweets)

{

List<Tweet> newTweets = new List<Tweet>();

newTweets = tweets.Select(x => new Tweet

{

Author = x.User.Name,

Date = x.CreatedDate,

Text = x.Text,

Url = "https://twitter.com/" + x.Author.ScreenName + "/status/" + x.IdStr

}).ToList();

newTerm.Tweets = newTweets;

}

Листинг 15: Помоћне методе за обраду и чување података у базу

Наког тога, потребно је додатно обрадити и форматирати резултате пре прослеђивања на *front-end*, ради лакшег приказа.

Као што се може видети на листингу 14, код за позив Аpi-ja је издвојен у посебну сервис класу, ApisService, која дата као пример употребе *Dependency Injection* механизма који Core поседује. Прво је неопходно направити интерфејс, који ће садржати методе које класа жели да имплементира, и онда класа мора да имплементира тај интерфејс, који ће се користити приликом регистровања сервиса унутар DI контејнера, што је већ приказано у поглављу за конфигурацију пројекта. Интерфејс ове класе се може видети на листингу 16.

public interface IApisService

{

List<Event> Allevents(string data);

Task<string> Wikipedia(string data);

List<TwitterStatus> Twitter(string data);

}

Листинг 16: IАpisService интерфејс

### 4.1.6 Wikipedia API

За добављање података са Wikipedia страница je коришћен MediaWiki API [13]. Потребна су два корака, први да се претраже постојеће странице на Wikipediji, и затим се узме тачан назив странице и за њу се, помоћу другог позива, добави скраћена верзија уводног пасуса са странице. Имплементација се може видети на листингу 17.

public async Task<string> Wikipedia(string data)

{

var client = new HttpClient();

string searchUrl = $"<https://en.wikipedia.org/w/api.php>? action=query&format=json&list=search&utf8=1&srsearch=intitle:{data}";

HttpResponseMessage searchResponse = await client.GetAsync(searchUrl);

string stringData = searchResponse.Content.ReadAsStringAsync().Result;

WikipediaSearchDto.RootObject apiData =

JsonConvert

.DeserializeObject<WikipediaSearchDto.RootObject>(stringData);

List<Search> pages = new List<Search>();

if(apiData != null && apiData.query != null)

{

pages = apiData.query.search;

}

string searchTermFromResult = pages[0] != null ? pages[0].title : data;

string uri = $"https://en.wikipedia.org/w/api.php[? format=json&action=query&prop=extracts&titles](https://en.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&titles)={searchTermFromResult}

&exintro&redirects=1&utf8";

HttpResponseMessage response = await client.GetAsync(uri);

var resultText = response.Content.ReadAsStringAsync().Result;

var index = resultText.IndexOf("extract");

if (index == -1)

{

return "no results";

}

else

{

string result = resultText.Substring(index + 10);

result = result.Remove(result.Length - 5);

result = result.Replace(@"\n", "<br />");

return result;

}

}

Листинг 17. Имплементација Wikipedia API-ја

### 4.1.7 Twitter API

За добављање података преко Twitter API-ја, прво је неопходно регистровати налог на њиховом сајту, да би се добили кључеви који се морају користити приликом позивања API-ја. Ове вредности се чувају као константе у appsettings.json фајлу, и затим се добављају употребом Configuration сервиса. Затим се користи библиотека TweetSharp [14], која пружа веома ефикасан начин да се позову *endpoint-*и Twitter Rest API-ја. Имплементација се може видети на листингу 18.

public List<TwitterStatus> Twitter(string data)

{

var service = new TwitterService(\_configuration["TwConsumerKey"], \_configuration["TwConsumerSecret"]);

service.AuthenticateWith(\_configuration["TwAccessToken"], \_configuration["TwAccessTokenSecret"]);

TwitterSearchResult tweets = service.Search(options:

new SearchOptions{ Q = data, Count = 100 });

List<TwitterStatus> status = tweets.Statuses.ToList();

return status;

}

Листинг 18: Имплементација Twitter API-ја

### 4.1.8 AllEvents.in API

Добављање догађаја преко AllEvents.in API-ја се може видети на листигну 19, и састоји се од позива њиховог Search *endpoint*-a, који је одговарао у својој функционалности потребама пројекта.

public List<Event> Allevents(string data)

{

var client = new HttpClient();

var queryString = HttpUtility.ParseQueryString(string.Empty);

string key = \_configuration["AllEventsKey"];

// Request headers

client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", key);

// Request parameters

queryString["query"] = data;

//queryString["latitude"] = "{string}";

//queryString["longitude"] = "{string}";

//queryString["city"] = "{string}";

//queryString["page"] = "{string}";

string q = String.Join("&", queryString.AllKeys.Select(a => a + "=" + HttpUtility.UrlEncode(queryString[a])));

var uri = "https://api.allevents.in/events/search/?" + q;

HttpResponseMessage response;

// Request body

byte[] byteData = Encoding.UTF8.GetBytes("{body}");

using (var content = new ByteArrayContent(byteData))

{

content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

response = client.PostAsync(uri, content).Result;

List<Event> allevents = new List<Event>();

if ((int)response.StatusCode == 200)

{

string stringData = response.Content.ReadAsStringAsync().Result;

List<Event> apiData = JsonConvert.DeserializeObject<AlleventsDto.RootObject>(stringData).data;

if (apiData != null) allevents = apiData;

}

return allevents;

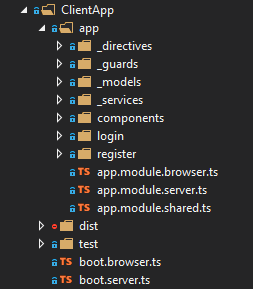
}

}

Листинг 19: Имплементација Allevents.in API-ја

### 4.1.8 Angular клијентска апликација

Структура Angular клијентске апликације се може видети на слици 8.



Слика 8: Аngular клијентска апликација

Фајл app.module.shared.ts садржи конфигурацију целе апликације, и у њему се, између осталог, врши подешавање рута апликације унутар *imports* објекта, и такође декларација сервиса који ће се касније користити преко *Dependency Injection* механизма, унутар *providers* објекта. Ово се може видети на листингу 20.

imports: [

CommonModule,

HttpModule,

FormsModule,

NgxPaginationModule,

RouterModule.forRoot([

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent, canActivate: [AuthGuard] },

{ path: 'login', component: LoginComponent },

{ path: 'register', component: RegisterComponent },

{ path: '\*\*', redirectTo: 'home' }

])

],

providers: [

AuthGuard,

AlertService,

AuthenticationService,

UserService,

SearchService

]

Листинг 20: Конфигурација Angular апликације

Овиме смо дефинисали руте, које се састоје од назива и компоненте која ће се позвати кад се оде на тај url. Овде се такође дефинише управљање корисничким налозима. Дакле Користи се AuthGuard класа, која се додаје у дефиницију руте, и помоћу које вршимо заштиту. Начин на који ово функционише је да се приликом пријаве проследи токен преко API-ја, и он се сачува локално, на претраживачу, и приликом покушаја да се отвори страница, провери се да ли постоји сачуван токен, уколико постоји кориснику се дозволи приступ, уколико не постоји, редиректује се на страницу за пријаву. Уколико токен постоји сачуван али је истекао, приликом претраге ће се добити 401 одговор са *back-end*-a, и након тога ће се корисник редиректовати на страницу за пријаву. Имплементација Authguard класе се може видети на листингу 21.

@Injectable()

export class AuthGuard implements CanActivate {

constructor(private router: Router) { }

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {

if (typeof window !== 'undefined' && localStorage.getItem('currentUser')) {

// logged in

return true;

}

// not logged in

this.router.navigate(['/login'], { queryParams: { returnUrl: state.url }});

return false;

}

}

Листинг 21: AuthGuard класа

Од верзије 2.0 Angular библиотеке, контролери су замењени са компонентама, тако да се свака страница Angular апликације састоји од компоненте, која у основи садржи TypeScript класу компоненте, засебни фајл са HTML садржајем странице, и такође може имати засебни CSS фајл који садржи CSS класе за стил. Ови фајлови се затим рефенцирају унутар компоненте. Улога компоненте је веома слична претходно коришћеним контролерима. Она садржи *listener*-e, затим се користи да би добавили податке са сервера, који се сместе у променљиве, које се затим, користећи функционалности Angular оквира, повезују са елементима на HTML страници, на пример, да би се приказали унутар табеле. Компонента за страницу на којој се врши претрага извођача се може видети на листингу 22, a њена HTML страница се може видети на листингу 23. Такође, унутар ове компоненте се користи класа SearchService да би се добавили резултати претраге преко нашег *back-end* API-ja, а њена имплементација се може видети на листингу 24.

@Component({

selector: 'home',

templateUrl: './home.component.html'

})

export class HomeComponent {

searchResult: SearchResult;

key: string = 'name';

reverse: boolean = false;

p1: number = 1;

p2: number = 1;

loading = false;

constructor(@Inject('BASE\_URL') private baseUrl: string, private router: Router, private searchService: SearchService) { }

ngOnInit() {

this.searchResult = {

wiki: "",

allEvents: [] as Event[],

tweets: [] as Tweet[]

};

}

search(title: string) {

if (title){

this.loading = true;

this.searchService.search(title)

.subscribe(result => {

this.searchResult = result.json() as SearchResult;

this.loading = false;

}, error => {

console.error(error);

this.loading = false;

if (error.status == 401) {

this.router.navigate(['login']);

}

}

);

}

}

}

Листинг 22: Компонента за претрагу

<h1>Search for an artist:</h1>

<div class="row">

<div class="col-xs-6 col-md-4">

<div class="input-group">

<input type="text" class="form-control" placeholder="Search" (keyup.enter)="search(searchInput.value)" #searchInput />

<div class="input-group-btn">

<button class="btn btn-default" type="submit" (click)="search(searchInput.value)" [disabled] = "loading">

<span class="glyphicon glyphicon-search" \*ngIf="!loading"></span>

<span \*ngIf="loading"> <img \*ngIf="loading" src="loadingImage" /></span>

</button>

</div>

</div>

</div>

</div>

<br>

<div class="panel panel-default" \*ngIf="searchResult.wiki">

<div class="panel-heading"><b>Wikipedia</b></div>

<div class="panel-body"><p innerHTML="**{{**searchResult.wiki**}}**"></p></div>

<div class="panel-footer"><a href="https://en.wikipedia.org/w/index.php?search=**{{**searchInput.value**}}**" target="\_blank">See more</a></div>

</div>

<div class="panel panel-default" \*ngIf="searchResult.allEvents?.length > 0">

<div class="panel-heading"><b>AllEvents.in</b></div>

<div class="panel-body">

<table class="table">

<thead>

<tr>

<th (click)="sort('Name')">

Name

</th>

<th (click)="sort('time')">

Time

</th>

<th (click)="sort('location')">

location

</th>

<th (click)="sort('url')">

Url

</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let event of searchResult.allEvents | paginate: { id: 'event', itemsPerPage: 10, currentPage: p1 };">

<td>**{{**event.name**}}**</td>

<td>**{{**event.time**}}**</td>

<td>**{{**event.location**}}**</td>

<td> <a href="**{{**event.url**}}**" target="\_blank">Event url</a> </td>

</tr>

</tbody>

<pagination-controls (pageChange)="p1 = $event" id="event"></pagination-controls>

</table>

</div>

</div>

<div class="panel panel-default" \*ngIf="searchResult.tweets?.length > 0">

<div class="panel-heading"><b>Tweeter</b></div>

<div class="panel-body">

<table class="table">

<thead>

<tr>

<th (click)="sort('Author')">

Author

</th>

<th (click)="sort('time')">

Time

</th>

<th (click)="sort('Text')">

Text

</th>

<th (click)="sort('url')">

Url

</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let tweet of searchResult.tweets | paginate: { id: 'tweet', itemsPerPage: 10, currentPage: p2 };">

<td>**{{**tweet.author**}}**</td>

<td>**{{**tweet.date**}}**</td>

<td>**{{**tweet.text**}}**</td>

<td> <a href="**{{**tweet.url**}}**" target="\_blank">Tweet url</a> </td>

</tr>

</tbody>

<pagination-controls (pageChange)="p2 = $event" id="tweet"></pagination-controls>

</table>

</div>

</div>

Листинг 23: HTML страница претраге

@Injectable()

export class SearchService {

constructor(private http: Http, @Inject('BASE\_URL') private baseUrl: string) { }

search(title: string) {

return this.http.get(this.baseUrl + 'api/search/search', { search: this.params(title), headers: this.jwt() });

}

private params(title: string) {

let params: URLSearchParams = new URLSearchParams();

params.set('data', title);

return params;

}

private jwt() {

// create authorization header with jwt token

const userStorage = localStorage.getItem('currentUser');

let currentUser = userStorage !== null ? JSON.parse(userStorage) : null;

if (currentUser && currentUser.token) {

let headers = new Headers({ 'Authorization': 'Bearer ' + currentUser.token });

return headers;

}

}

}

Листинг 24: SerchService klasa

Angular подржава различите начине повезивања, па тако на пример:

* (keyup.enter)="search(searchInput.value)" значи да повезујемо догађај притиска ентер тастера на позив функције search, којој такође прослеђујемо као параметар вредност коју је корисник унео у поље за претрагу.
* [disabled] = "loading" значи да вредност disabled параметра унутар, на пример дугмета, динамички добија предност променљиве коју смо дефинисали у компоненти, и која се у овом случају зове loading. Па кад код се у позадини извршава добављање података са сервера, треба поставити вредност променљиве loading на „true”, и кад се операција заврши на „false“, и Angular ће аутоматски доделити исту ту вредност параметру ком смо ову променљиву доделили, што у овом случају значи да ће дугме бити онемогућено док се подаци добављају.
* [(ngModel)]="model.username", је повезивање у оба смера, што значи да можемо у компоненти да променимо вредност променљиве, и видеће се промена на страници, а такође корисник може на страници да унесе неку вредност у поље за унос и то ће променити вредност променљиве у компоненти. Овај приступ се користи у страницама за пријаву и регистровање, које се могу видети на листингу 25 и 26.

<div class="col-md-6 col-md-offset-3">

<h2>Login</h2>

<form name="form" (ngSubmit)="f.form.valid && login()" #f="ngForm" novalidate>

<div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">

<label for="username">Username</label>

<input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />

<div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>

</div>

<div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">

<label for="password">Password</label>

<input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />

<div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>

</div>

<div class="form-group">

<button [disabled]="loading" class="btn btn-primary">Login</button>

<img \*ngIf="loading" src="loadingImage" />

<a [routerLink]="['/register']" class="btn btn-link">Register</a>

</div>

</form>

</div>

Listing 25: HTML странице за пријаву

<div class="col-md-6 col-md-offset-3">

<h2>Register</h2>

<form name="form" (ngSubmit)="f.form.valid && register()" #f="ngForm" novalidate>

<div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">

<label for="username">Username</label>

<input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />

<div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>

</div>

<div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">

<label for="password">Password</label>

<input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />

<div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>

</div>

<div class="form-group">

<button [disabled]="loading" class="btn btn-primary">Register</button>

<img \*ngIf="loading" src="loadingImage" />

<a [routerLink]="['/login']" class="btn btn-link">Cancel</a>

</div>

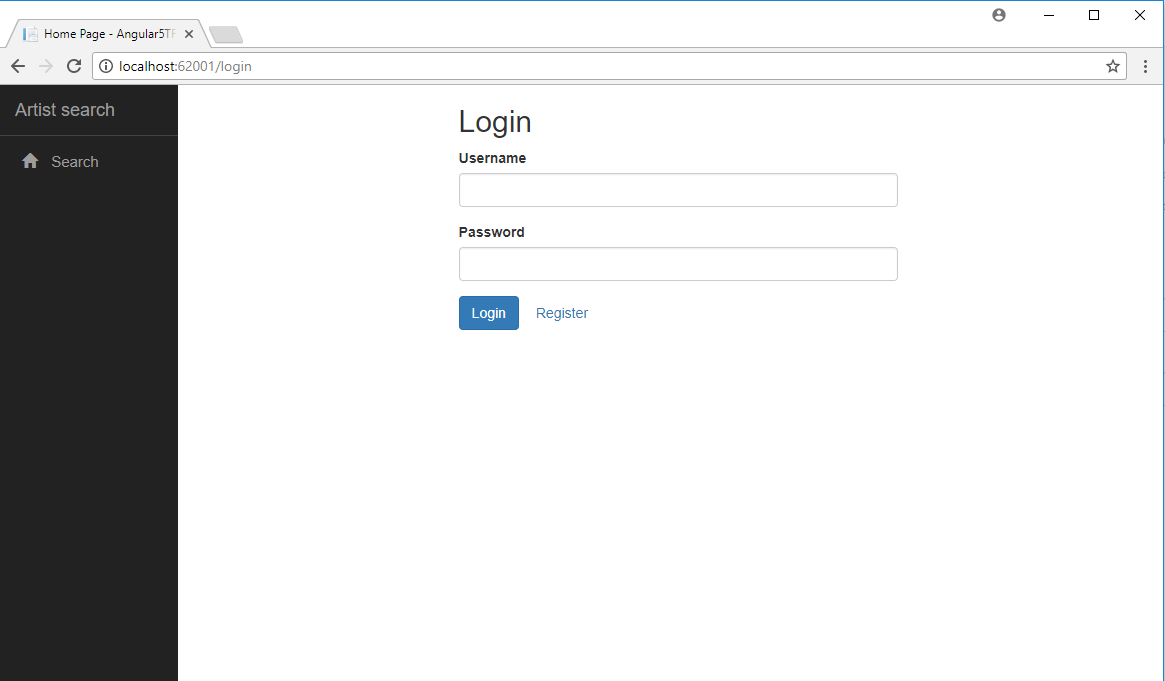
</form>

</div>

Листинг 26: HTML странице за регистрацију

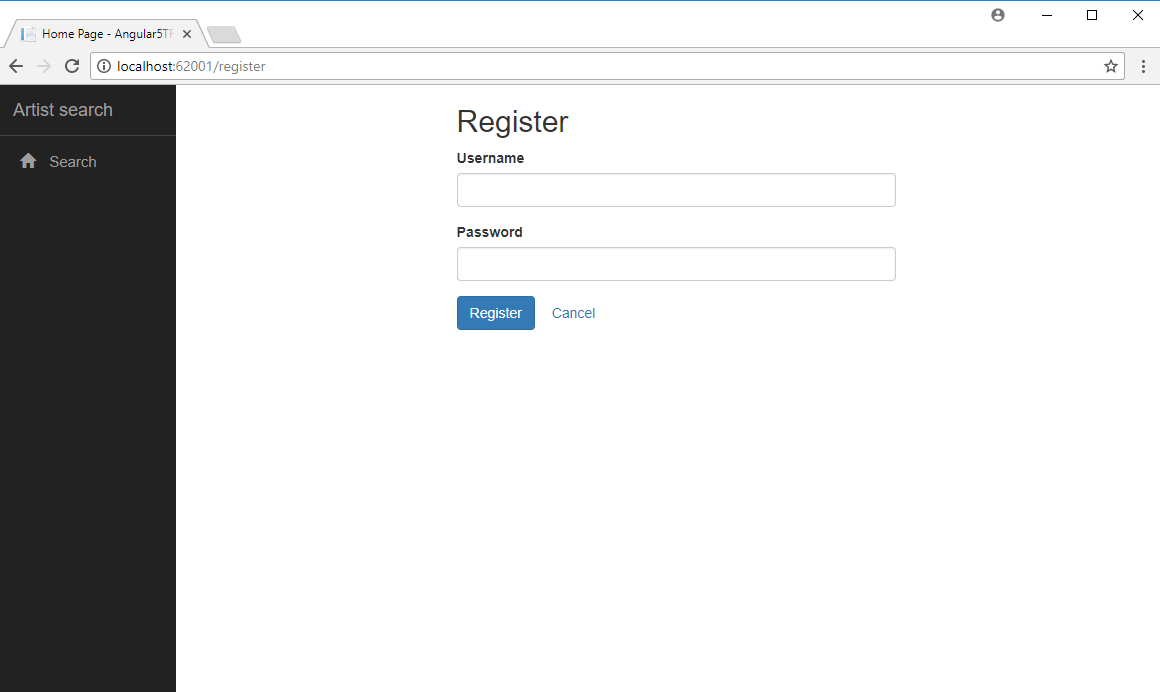
# Демонстрација

У овом поглављу ће бити демонстрирана апликација и њене основне функционалности. Страница за пријаву је прва коју корисник види, и она се може видети на слици 9.



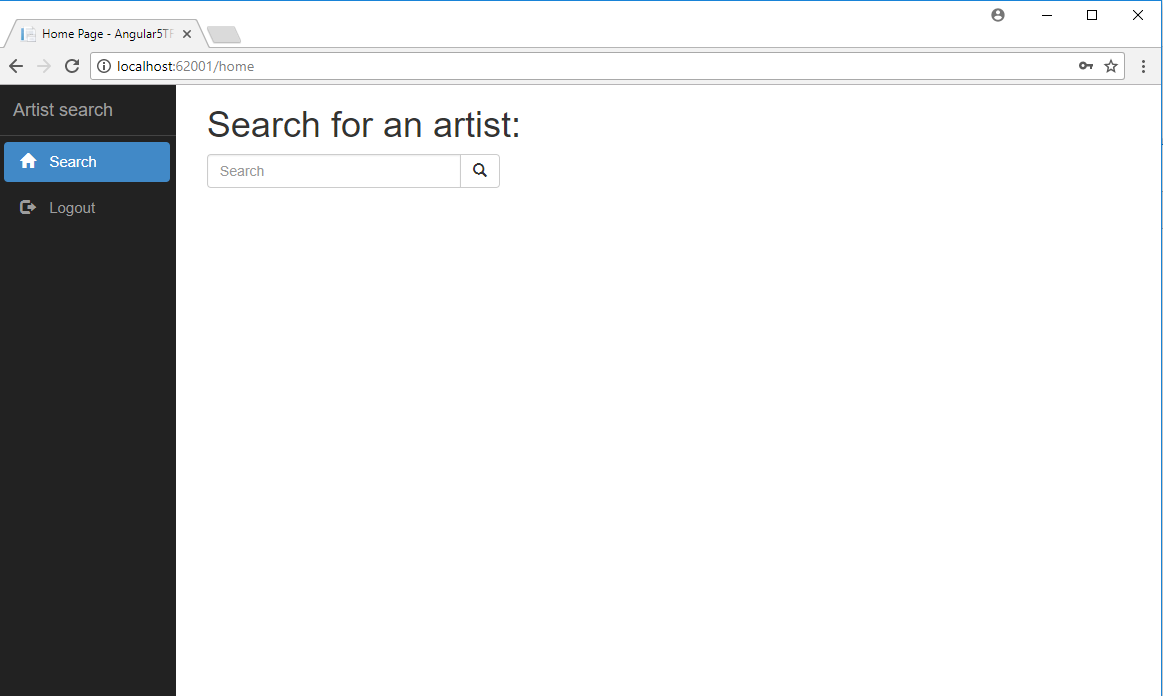
Слика 9: Страница за пријаву

Са ње се може прећи на страницу за регистрацију кликом на линк „Register“, која се може видети на слици 10.



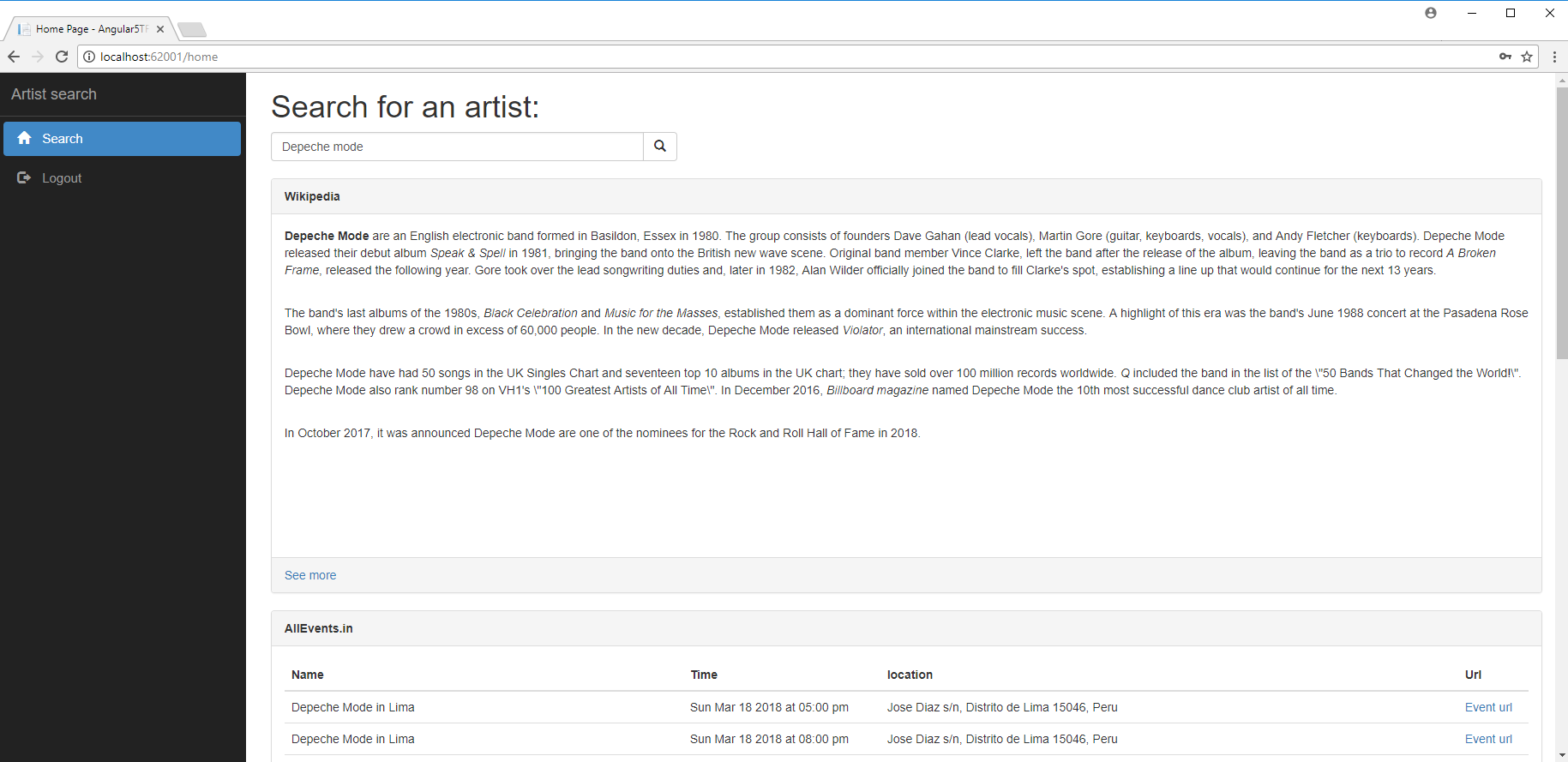
Слика 10: Страница за регистрацију

Након успешне регистрације, корисник је преусмерен на страницу за претрагу, која се може видети на слици 11.

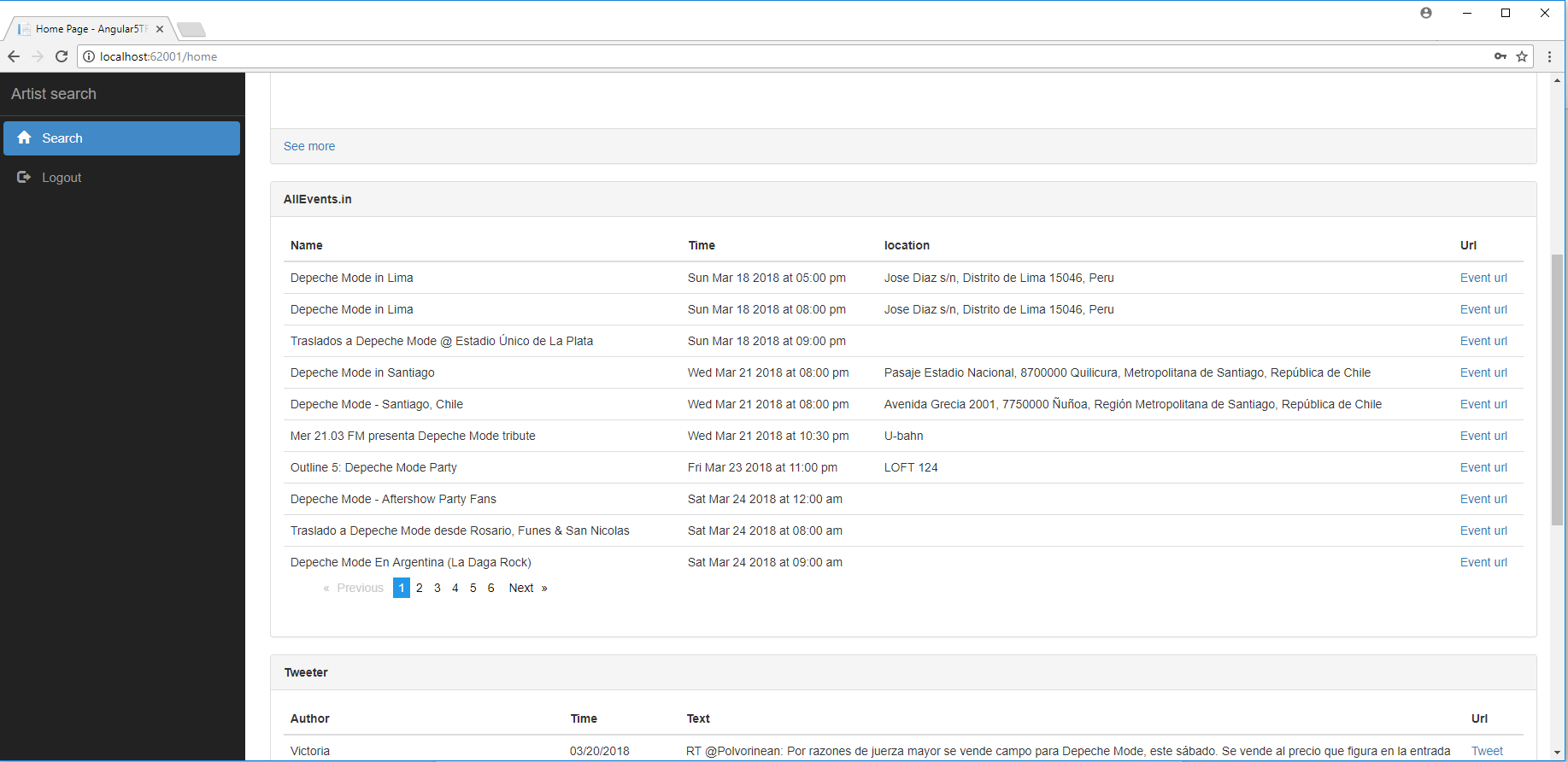


Слика 11: Страница за претрагу

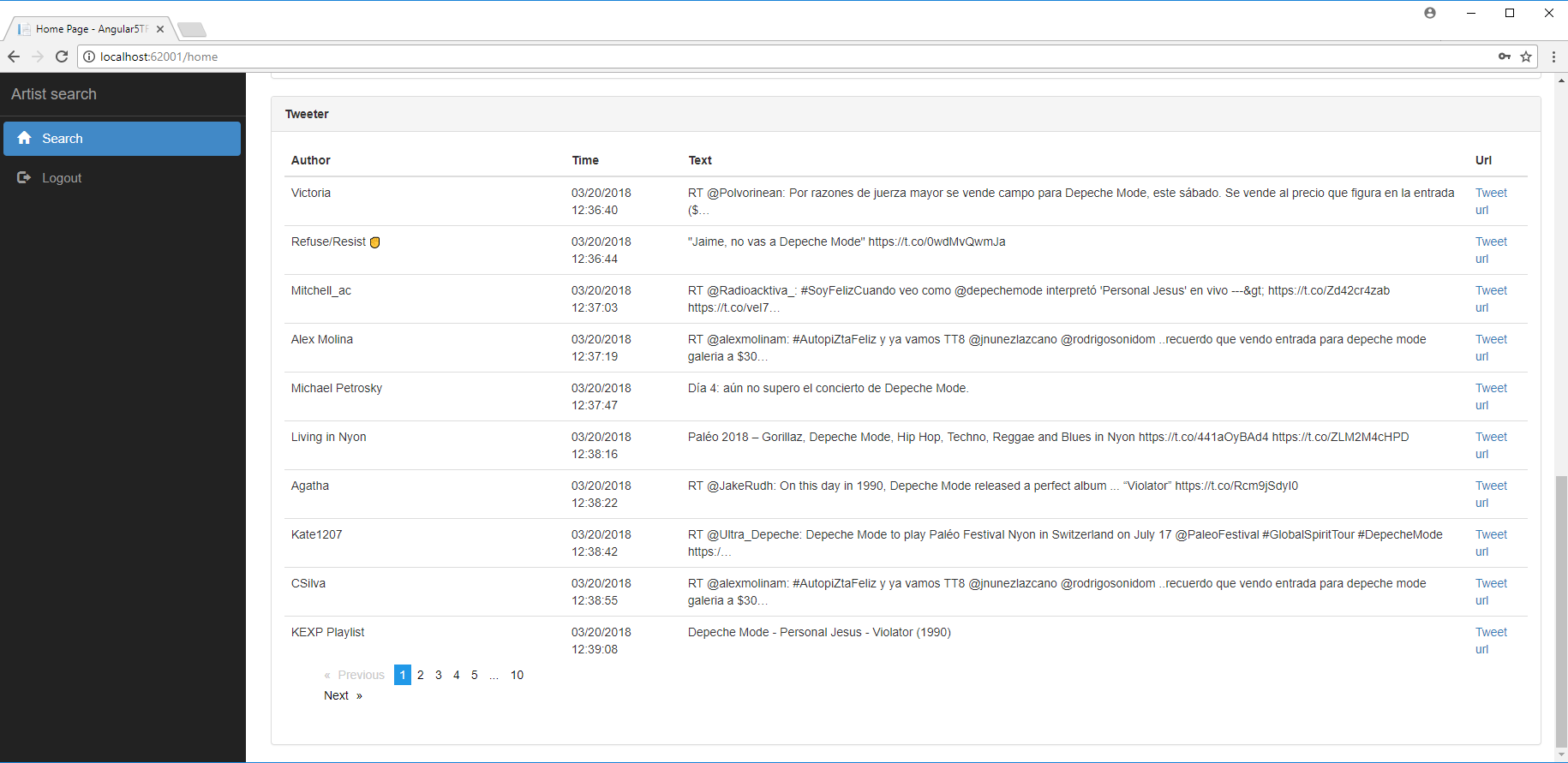
У поље за претрагу се уноси насив извођача, и након тога ће се појавити резултати претраге, што се може видети на сликама 12, 13 и 14.



Слика 12: Резултати са Wikipedia сајта



Слика 13: Резултати са AllEvents.in сајта



Слика 14: Резултати са Twitter сајта

Након тога, могуће је отићи на линк пронађене Wikipedia странице кликом на „See more“, и такође је могуће отићи на страницу за догађај на AllEvents сајту, и такође отићи на страницу конкретног Tweet-а.

# ЗАКЉУЧАК

Задатак овог рада је био да се истражи .NET Core 2.0 оквир, да се имплементира апликација кроз коју ће се демонстрирати његова употреба, и да се упореди његов рад, имплементација, предности и мане у односу на претходну верзију .NET оквира. Апликација је имплементирана употребом *Single Page Application* приступа, па је у ту сврху коришћен Angular 5 оквир за клијентску апликацију пројекта, кроз шаблон који Микрософт пружа уз .NET Core 2.0 оквир као њихов савет за брзу реализацију SPA апликација.

Главна предност Core оквира, у односу на .NET Framework оквир, је то да у основи подржава више платформи, уместо само Windows платформе. Осим тога, боље перформансе, специфичне могућности и генерално израда оквира од нуле, са применом свега што је научено кроз израду претходне верзије оквира, су такође предности.

Мана Core оквира је првенствено то да је још увек је у изради, која већ више година траје, и нису имплементиране све функционалности. Ово се може видети приликом рада са базом. Верзија *Entity Framework*-а која је била део претходне верзије .NET оквира је била веома користан ORM, пошто је веома квалитетно подржавао различите приступе рада са базом и такође је био веома лак за употребу, док тренутна верзија *Entiy Framework Core* библиотеке, која је део .NET Core-a има само део тих функционалности. Пре израде апликације употребом .NET Core 2.0 оквира, саветује се да се истражи које функционалности су тренутно подржане и да се из тога процени да ли тренутно има смисла његова употреба за тај конкретан пројекат.

Што се тиче даљег истраживања на ову тему, првенствено је битно пратити развој Core оквира. Нове верзије редовно излазе, и већ је најављена верзија 2.1. Што се проширења апликације тиче, њена основна идеја је прикупљање података са више извора, и могу се лако додати нови извори за прикупљање података поред тренутно коришћених. Потребно би било само додати нову табелу у базу и везати је за основни SearchTerm ентитет, додати нов метод у сервис за API-је, написати метод који ће да сачува податке у ту табелу и приказати резултате на Search страници. Идеја је била да је овај део апликације модуларан, тако да се лако може проширити.

# ЛИТЕРАТУРА

[1] Microsoft .NET, <https://www.microsoft.com/net/>

[2] ASP.NET, <https://www.asp.net/>

[3] .NET standard, <https://docs.microsoft.com/en-us/dotnet/standard/net-standard>

[4] .NET Core, <https://docs.microsoft.com/en-us/dotnet/core/>

[5] .NET framework, <https://www.microsoft.com/net/download/dotnet-framework-runtime>

[6] Xamarin, https://www.xamarin.com/

[7] Mono, <https://www.mono-project.com/>

[8] Razor pages, <https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio>

[9] Angular, <https://angular.io/>

[10] Allevents.in, https://allevents.in/

[11] Twitter, https://twitter.com/

[12] Wikipedia, <https://www.wikipedia.org/>

[13] MediaWiki API, <https://www.mediawiki.org/wiki/API:Main_page>

[14] TweetSharp, https://github.com/shugonta/tweetsharp

# БИОГРАФИЈА

Јован Попов, рођен 1.9.1988, Нови Сад, општина Нови Сад, Србија. Завршио основну школу „Вук стефановић Караџић“ у Дероњима, средњу електротехничку школу „Михајло Пупин“ у Новом Саду. Након тога уписао се на Факултет Техничких Наука, одсек Рачунарство и Аутоматика.

Контакт адреса: [jovan.popov.88@gmail.com](mailto:jovan.popov.88@gmail.com)

Контакт телефон: 063/7174791