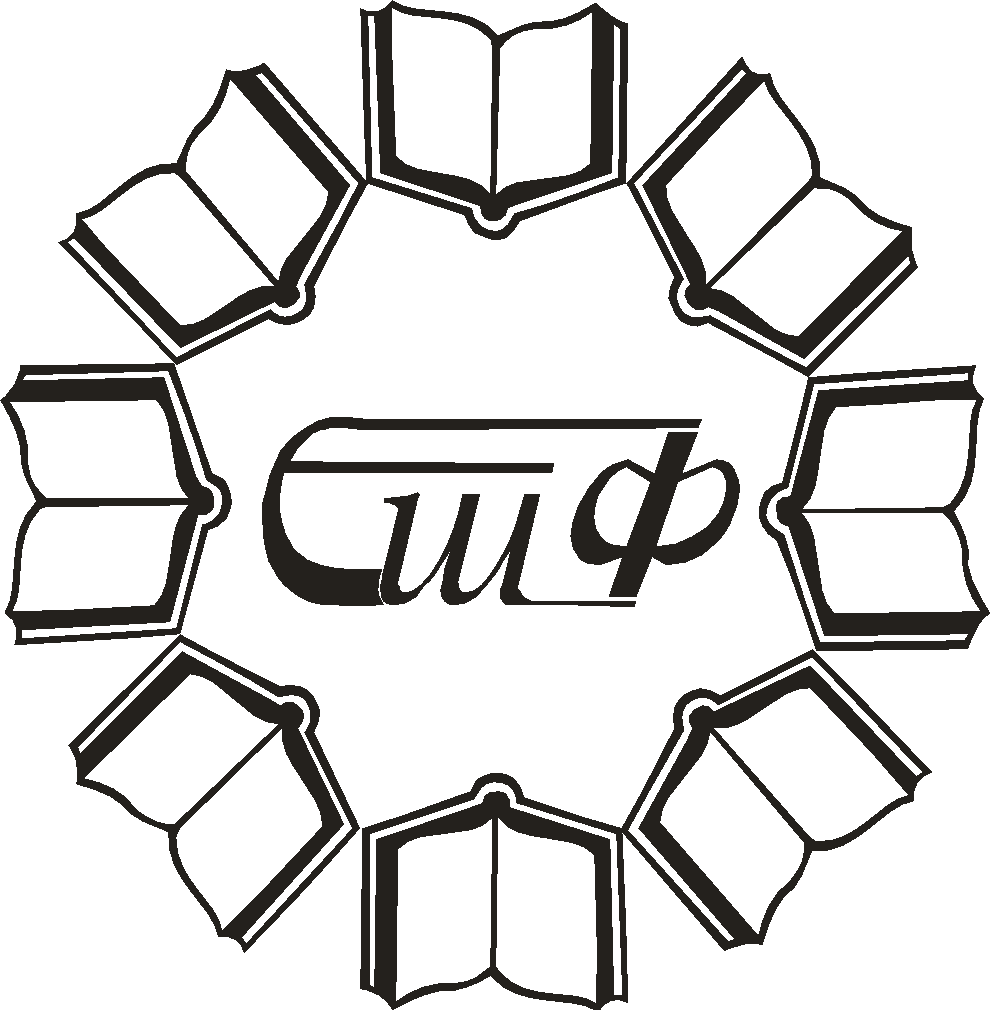
**УНИВЕРЗИТЕТ У ИСТОЧНОМ САРАЈЕВУ**

**ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ**

**Смјер: Рачунарство и информатика**

**Online систем за куповину аутомобила**

**ЗАВРШНИ РАД**

Студент: Никола Шаровић Ментор: Проф. др Срђан Ного

**Источно Сарајево, Октобар 2022**

# Садржај

[Садржај 2](#_Toc114583175)

[Сажетак 5](#_Toc114583176)

[Abstract 6](#_Toc114583177)

[Списак слика 7](#_Toc114583178)

[Списак табела 9](#_Toc114583179)

[Списак скраћеница 10](#_Toc114583180)

[Кратак преглед рада 11](#_Toc114583181)

[1 UВОД 12](#_Toc114583182)

[1.1 Електорнска трговина 12](#_Toc114583183)

[1.1.1 Модели електронске трговине 13](#_Toc114583184)

[1.2 Вeб апликације 14](#_Toc114583185)

[1.2.1 *WWW* концепт 15](#_Toc114583186)

[1.2.2 Архитектура веб апликација 16](#_Toc114583187)

[1.2.3 Типови вeб апликације 17](#_Toc114583188)

[1.3 Oбјашњење *SPA* (*Single page application*) 17](#_Toc114583189)

[1.3.1 Разлике између *SPA* и *MPA (Multi-Page Apps)* 19](#_Toc114583190)

[2 СПЕЦИФИКАЦИЈА И МОДЕЛОВАЊЕ СИСТЕМА 22](#_Toc114583191)

[2.1 Функционални захтјеви система 23](#_Toc114583192)

[2.2 Use-case model 25](#_Toc114583193)

[2.3 Koнцептуални модел 26](#_Toc114583194)

[3 ПРЕГЛЕД КОРИШТЕНИХ ТЕХНОЛОГИЈА 27](#_Toc114583195)

[*3.1* *Microsoft SQL Server* 27](#_Toc114583196)

[3.2 Технологије на серверској страни 28](#_Toc114583197)

[*3.3* *ASP.NET Core* 29](#_Toc114583198)

[3.3.1 Web Api 31](#_Toc114583199)

[*3.4* *Entity Framework* 32](#_Toc114583200)

[3.4.1 *Entity Framework Architecture* 35](#_Toc114583201)

[3.4.2 Приступи развоју 36](#_Toc114583202)

[3.4.3 Сценарио развоја 38](#_Toc114583203)

[*3.5* *Swagger UI* 39](#_Toc114583204)

[3.6 *Microsoft* Azure 40](#_Toc114583205)

[3.7 Технологије на клијентској страни 42](#_Toc114583206)

[3.8 *HTML* 42](#_Toc114583207)

[3.8.1 *HTML* елементи 42](#_Toc114583208)

[3.8.2 Начин употребе 43](#_Toc114583209)

[3.9 *CSS* 44](#_Toc114583210)

[3.9.1 Синтакса 44](#_Toc114583211)

[3.10 *JavaScript* 45](#_Toc114583212)

[3.10.1 Ajax 46](#_Toc114583213)

[3.11 *TypeScript* 47](#_Toc114583214)

[*3.12* *React* 48](#_Toc114583215)

[3.12.1 Virtuelni *DOM* 49](#_Toc114583216)

[3.12.2 Животни циклуси компоненти 50](#_Toc114583217)

[3.12.3 *React* *Hooks* 51](#_Toc114583218)

[*3.12.4* *Material UI* 52](#_Toc114583219)

[*3.13* *Redux* 52](#_Toc114583220)

[4 ИМПЛЕМЕНТАЦИЈА БАЗЕ ПОДАТАКА И БЕКЕНДА 54](#_Toc114583221)

[*4.1* *Repository pattern* 55](#_Toc114583222)

[4.2 Модел базе 58](#_Toc114583223)

[4.3 Складиштење фајлова 59](#_Toc114583224)

[5 ИМПЛЕМЕНТАЦИЈА ФРОНТЕНДА 62](#_Toc114583225)

[5.1 Корисничка хијерархија 62](#_Toc114583226)

[5.2 Структура веб апликације 62](#_Toc114583227)

[5.3 Приступ систему 63](#_Toc114583228)

[5.4 Претрага аутомобила 64](#_Toc114583229)

[5.5 Приказ аутомобила 65](#_Toc114583230)

[5.6 Регистрација корисника 65](#_Toc114583231)

[5.7 Пријава на систем 66](#_Toc114583232)

[5.7.1 Кориснички профил 67](#_Toc114583233)

[5.7.2 Измјена шифре 69](#_Toc114583234)

[5.7.3 Управљање објавама 69](#_Toc114583235)

[5.7.4 Креирање аутомобила 71](#_Toc114583236)

[6 ЗАКЉУЧАК 72](#_Toc114583237)

[ЛИТЕРАТУРА 73](#_Toc114583238)

# Сажетак

У раду је приказан цјелокупан процес развоја система за продају аутомобила, исти је базиран на *C2C* (*енгл.consumer to consumer*) моделу електронског пословаља. У оквиру рада дат је осврт на концепт електронског пословања, као и на различите моделе пословања дефинисане у оквиру истог. Под развојем система подразумијевају се како спецификација и моделовање система, тако и приказ технологија кориштених за имплеметацију самог система, закључно са приказом готовог апликативног рјешења са свим елементима истог.

# Abstract

The paper presents the entire process of developing the car sales system, based on the C2C model of electronic business. Within the scope of the paper, an overview of the concept of electronic business, as well as the various business models defined in it, is given. The development of the system includes the specification and modeling of the system, as well as the presentation of the technologies used to implement the system itself, including the presentation of the finished application solution with all its elements.

# Списак слика

[Слика 1.1 -Функционисање типичне вeб апликације 14](#_Toc113923997)

[Слика 1.2 -Познате вeб апликације 17](#_Toc113923998)

[Слика 1.3 -Како ради SPA 18](#_Toc113923999)

[Слика 1.4 -Разлика између SPA и MPA 19](#_Toc113924000)

[Слика 2.1 -Модел водопада 2](#_Toc113924001)3

[Слика 2.2 -Use Case модел 2](#_Toc113924002)5

[Слика 2.3 -Концептуални модел 2](#_Toc113924003)6

[Слика 3.1 -Backend развој 2](#_Toc113924004)8

[Слика 3.2 -Приказ ASP.NET Core 2](#_Toc113924005)9

[Слика 3.3 -Архитектура .NET CORE радног оквира 3](#_Toc113924006)1

[Слика 3.4 -Приказ Web API 3](#_Toc113924007)2

[Слика 3.5 -Entity Framework 3](#_Toc113924008)3

[Слика 3.6 -Entity Framework API 3](#_Toc113924009)5

[Слика 3.7 -Архитектура Entity радног оквира 3](#_Toc113924010)6

[Слика 3.8 -Дијаграм одлучивања код развоја базе података (EF) 3](#_Toc113924011)7

[Слика 3.9 -Приступ „База прво“ 3](#_Toc113924012)7

[Слика 3.10 -Приступ „Код прво“ 3](#_Toc113924013)7

[Слика 3.11 -Приступ „Модел прво“ 3](#_Toc113924014)8

[Слика 3.12 -Повезани сценарио 3](#_Toc113924015)8

[Слика 3.13 -Неповезани сценарио 3](#_Toc113924016)9

[Слика 3.14 -Изглед Swagger UI у апликацији](#_Toc113924017) 40

[Слика 3.15 -Microsoft Azure 4](#_Toc113924018)1

[Слика 3.16 -HTML element 4](#_Toc113924019)3

[Слика 3.17 -Декларација HTML документа 4](#_Toc113924020)3

[Слика 3.18 -CSS синтакса 4](#_Toc113924021)4

[Слика 3.19 -Приказ ЈЅ у HTML документу 4](#_Toc113924022)6

[Слика 3.20 -Рад Аjax-а 4](#_Toc113924023)6

[Слика 3.21 -Структура TypeScript-а 4](#_Toc113924024)7

[Слика 3.22 -Рад React-а 4](#_Toc113924025)8

[Слика 3.23 -Примјер JSX 4](#_Toc113924026)8

[Слика 3.24 -DOM веб апликације 4](#_Toc113924027)9

[Слика 3.25 -Животни циклуси компоненте](#_Toc113924028) 50

[Cлика 3.26 -Сегменти Flux дизајн шаблона 5](#_Toc113924029)3

[Cлика 3.27 -Сегменти Redux шаблона 5](#_Toc113924030)3

[Слика 4.1 -Примјер модела за табелу Car 5](#_Toc113924031)4

[Слика 4.2 -Примјер миграције за креирање табеле Car 5](#_Toc113924032)5

[Слика 4.3 -Repository pattern 5](#_Toc113924033)6

[Cлика 4.4 -Repository Pattern Interface 5](#_Toc113924034)7

[Cлика 4.5 -Repository pattern - SQL Server 5](#_Toc113924035)7

[Cлика 4.6 -Табела Identity и база података 5](#_Toc113924036)8

[Cлика 4.7 -Resource group 5](#_Toc113924037)9

[Слика 4.8 -Storage account](#_Toc113924038) 60

[Слика 4.9 -Azure Container Instances](#_Toc113924039) 60

[Cлика 4.10 -Програмски код за рад са сервисом Azure Blob storage service 6](#_Toc113924040)1

[Слика 5.1 -Почетна страница са заглављем и главним дијелом 6](#_Toc113924041)3

[Слика 5.2 -Крај странице 6](#_Toc113924042)3

[Cлика 5.3 -Претрага по бренду аутомобила 6](#_Toc113924043)4

[Слика 5.4 -Претрага по моделу аутомобила 6](#_Toc113924044)4

[Слика 5.5 -Приказ аутомобила 6](#_Toc113924045)5

[Слика 5.6 -Форма за регистрацију 6](#_Toc113924046)6

[Слика 5.7 -Форма за пријаву на систем 6](#_Toc113924047)6

[Слика 5.8 -Навигациони мени 6](#_Toc113924048)7

[Слика 5.9 -Преглед података о профилу 6](#_Toc113924049)7

[Слика 5.10 -Преглед података о креираном профилу 6](#_Toc113924050)7

[Слика 5.11 -Додавање или измјена података о профилу 6](#_Toc113924051)8

[Слика 5.12 -Приказ поруке о измјенама на профилу 6](#_Toc113924052)8

[Слика 5.13 -Форма за измјену шифре 6](#_Toc113924053)9

[Слика 5.14 -Преглед креираних аутомобила од корисника 6](#_Toc113924054)9

[Слика 5.15 -Измјена података креираног аутомобила](#_Toc113924055) 70

[Слика 5.16 -Брисање одређеног креираног аутомобила](#_Toc113924056) 70

[Слика 5.17 -Креирање аутомобила 7](#_Toc113924057)1

[Слика 5.18 -Поруке о креирању аутомобила 7](#_Toc113924058)1

# Списак табела

[Табела 2.1 -Приказ функционалних захтјева 25](#_Toc114583630)

[Табела 3.1 -Историја .*NET* CORE радног оквира 30](#_Toc114583631)

[Табела 3.2 -Приказ разлика измеђуе *Web API* и *ASP.NET* MVC 32](#_Toc114583632)

# Списак скраћеница

|  |  |
| --- | --- |
| **EF** | **E**ntity **F**ramework |
| **SQL** | **S**tructured **Q**uery **L**anguage |
| **DOM** | **D**ocument **O**bject **M**odel |
| **HTTP** | **H**yperText **T**ransport **P**rotocol |
| **JSON** | **J**ava **S**cript **O**bject **N**otation |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **HTML** | **H**yper **T**ext **M**arkup **L**anguage |
| **URL** | **U**niform **R**esource **L**ocator |
| **JS** | **J**ava **S**cript |
| **OS** | **O**perating **S**ystem |
| **MSSQL** | **M**icrosoft **S**QL **Server** |
| **CSS** | **C**ascading **S**tyle **S**heets |
| **CRUD** | **C**reate **R**ead **U**pdate **D**elete |
| **ORM** | **O**bject **R**elational **M**apping |
| **SPA** | **S**ingle **P**age **A**pplication |
| **MPA** | **M**ulti **P**age **A**pplication |
| **UI** | **U**ser **I**nterface |
| **ASP** | **A**ctive **S**erver **P**ages |
| **WEB** | **W**orld **W**ide **W**eb |
| **MVC** | **M**odel **V**iew **C**ontroller |
| **SSMS** | **S**QL **S**erver **M**anagement **S**tudio |
| **SЕО** | **S**earch **Е**ngine **О**ptimization |

# Кратак преглед рада

Овај рад састоји се од увода, три поглавља, закључка, литературе, списка слика, списка табела и списка скраћеница који су коришћени у овом раду.

У уводу је обрађен појам саме веб (*енгл.Web)* апликације, такође и појам електронске трговине кроз њен развој и њену подјелу на моделе електорнске трговине. Такође, дат је опис *SPA (енгл.*Single *page application)* са свим предностима и недостацима коришћења поменутог начина имплементације веба корисничких интерфејса.

Прво поглавље приказаје спецификацију и моделовање система. Ту се детаљно разрађују функционални захтјеви система који се у каснијем раду требају имплементирати кроз апликацију. Модели случајева коришћења гдје се препознају одговарајући корисници, којима се додјељују захтјеви са којима они располажу у апликацији. Модел базе приказан кроз концептулани модел који ће омогућити даљи развој саме базе у апликацији.

У другом поглавњу приказују се техноглогије које су коришћене приликом реализације завршног рада. Детаљно је дат приказ тих технологија кроз њихову историју, верзије и слично. Дат је опис *MSSQL,* као за подршку базе података.Tехнологије на серверској страни *ASP.NET Core* као *sеrvеr – sidе* технологију и његов *Web API.* Опис коришћења *Еntity Framework*, *Swagger,Microsoft Azure.* Технологије на клијентској страни *HTML* (*енгл*. *HyperText Markup Language*), *CSS* (*енгл*. *Cascading Style Sheets*) и *JavaScript*, *TypeScript, React* и *Redux.*

Треће поглавље је приказ апликативног рјешења. Ту је описана имплементација базе података и бекенда, уз детаљнији опис реализације модела базе и повезивања са *ASP.NET Core* – ом. Дат је приказ складиштења фајлова помоћу *Microsoft Azure-*а. Имлементација фронтеда система, односно опис приказа података које корисник може да види на свом екрану, те како може да користи дате податке, али и како управљати одређеним дијеловима система, да би кориснику обезбиједили што лакшу и једноставнију употребу.

Поједини оригинални називи на енглеском језику као и скраћенице, иако су набројане у номенклатурама које се налазе на почетку овог завршног рада, представљени су и у заградама ради лакшег разумијевања и праћења.

# UВОД

Са напредовањем технологија повећава се употреба интернета и он постаје један од главних видова комуникација, али и систем продаје. *Оnline* продаја је све заступљенија у свијету, из године у годину расте број сајтова за куповину. Због ових података веб апликације су се временом прилагођавале уређајима, тако да сада на својим телефонима можемо приступити готово свим веб апликацијама и оне ће бити прилагођене мобилном уређају или таблету односно њиховим екранима. Систем куповине преко интернета самим тим је омогућен свима, односно особама које посједују бар неки од уређаја преко којих се могу повезати на интернет. *Online* продаја је вид електронске трговине (*енгл. electronic commerce*) која омогућава куповину да преко интернета одаберете и наручите производ који желите да купите и да производ буде достављен у зависности од начина доставе.

## Електорнска трговина

Електронска трговина (*енгл.e-commerce)* јесте један подразумјевани дио електронског пословања. Креће се од појма трговине уопште. Ријеч трговина води поријекло од енглеске ријечи *commerce* која означава различите типове друштвених односа као што су нпр. размјена мишљења, идеја, размјене робе и транспорт робе од мјеста до мјеста. Главне улоге у трговини су: купци-особе или институције са новцем које желе да га замијене за производ или услугу, продавци-особе или институције које нуде производе или услуге купцима, призвођачи-особе или институције које креирају производе или услуге који се даље продају.

Електронска трговина има исте карактеристике као трговина као што су елементи: производ, мјесто, маркетинг, начин за пријем наруџбина, испорука, могућност враћања производа итд. Разлика од класичне трговине је само по средствима рада. Електронска трговина или Е-трговина представља пословну комуникацију и пренос добара и услуга (куповина и продаја), преко мреже и компјутера, као и пренос капитала коришћењем дигиталне комуникације. Елементи електорнске трговине:

* **Производ** могуће је продати двије ствари: производ и услугу. Може бити проширење класичне трговине.
* **Мјесто** продаје код електронске трговине је веб сајт, односно простор за продавницу и представља дио веб простора.
* **Маркетинг** циљ маркетинга је повећање саобраћаја купаца кроз продавницу и повећање процента посјетилаца који се одлучују да купе производ. За електронску продавницу потребно је да корисници интернета сазнају за њену веб адресу, да је посјете и сазнају шта продавница нуди.
* **Начин за пријем наруџба** електронска продавница омугућава пријем поруџбина аутоматски, путем формулара и без потребе за особљем које би се бавило контактима са муштеријама.
* **Начин за пријем новца** могуће је путем картица, али ту је потребна велика сигурност података.
* **Испорука** обично је предата одређеним фирмама које се баве транспортом од продавца до купца
* **Гаранција** могућност сервисирања производа
* **Техничка подршка** као живог телефонског контакта са особљем, обезбјеђују фирме које продају техничке производе који нису једноставни за коришћење

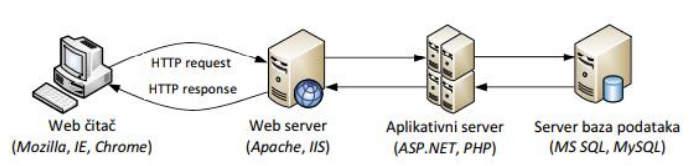
### Модели електронске трговине

Избор и примјена одговарајућег бизнис модела на одређени електронски бизнис (нпр. интернет продавница) представља велики изазов али и битно утиче на пословање самог бизниса. Користе се различити модели или типови електронске трговине. При томе приход од *B2B* електронске трговине износи 85% од укупне е-трговине, док приход од осталих модела износи око 15%. Пословни модели е-трговине се генерално могу категорисати у сљедеће категорије:

* Пословање за посао (*B2B*) продаје своје производе посредном купцу који затим продаје производ крајњем купцу.
* Бизнис – потрошачима (*B2C*) продаје своје производе директно купцу.
* Од потрошача до потрошача (*C2C*) помаже потрошачима да продају своју имовину као што су стамбена имовина, аутомобили, мотоцикли итд., или изнајме сервер тако што објављују своје информације на веб локацији.
* Од потрошача до предузећа (*C2B*) потрошач прилази веб страници која приказује више пословних организација за одређену услугу.
* Пословање за државу (*B2G*) владе користе такве веб странице за трговину и размјену информација са различитим пословним организацијама.
* Влада за пословање (*G2B*) да приступе пословним организацијама.
* Влада – грађанину (*G2C*) да би уопште приступиле грађанима.
* Пословање до посла.

## Вeб апликације

Веб апликације су настале развојем веб сајтова који у својој основи представљају скупове. Веб страница која се налази на одређеном веб серверу. Веб апликација представља софтверски производ који се извршава на веб серверу, а приказује корисника помоћу веб *читач* (*енгл. browser*). Да би веб апликација функционисала потребно је да се реализује у програмском језику који подржава веб сервер као софтверски „*engine*“ (односно *hosting* веб апликације коју технологију подржава), да би он могао да ту апликацију преведе и изврши. Веб апликација омогућава корисницима лакши приступ интернету и интерактивност између интернета и корисника. Десктоп апликације морају бити инсталиране на клијент рачунару да би се уопште могле користити. Код веб апликације то није случај јер се оне налазе на једном месту (серверу или групи сервера). Овим се минимизирају многи проблеми, прије свега проблеми са дистрибуцијом, надоградњом апликација и генерално у случајевима када koд десктоп апликација корисник мора да преузме и инсталира/ажурира апликацију. Битна ствар је да код веба корисник никада не добија саму апликацију, већ само интерфејс ка њој што је у суштина све што му и треба. На Слици 1.1 дат је приказ типичног функционисања веб апликације.



Слика .1 -Функционисање типичне вeб апликације

Принцип по коме функционишу веб апликације је прилично једноставан. Корисников веб *browser* шаље захтјев веб серверу, који податке просљеђује веб апликацијама, она их обради, а резултат враћа серверу, па напокон и *browser-u*. У тој ситуацији *browser* уопште не зна да ли је корисник услужен на страници или је код који претраживач приказује генерисан од стране веб апликације. Није му ни битно, докле год прослијеђени код разумије и може да прикаже.Увођењем програмских елемената у приказ веб страница настала су два типа веб страница,a то су:

* Статичке: изглед им је дефинисан већ приликом прве израде
* Динамичке: изглед им се дефинише кроз интеракцију са сервером

Најпознатије технологије – програмски језици за развој веб апликација су:

* *ASP/ASP.NET (C#)*
* *Java*
* *PHP*

Једна од основних мјера квалитета веб апликације су перформансе. Главна перформанса која утиче на успјешност једне веб апликације је вријеме одзива, што првенствено утиче на квалитет и употребну вриједност апликације. Ако је некој веб апликацији потребно много времена да би се учитала, корисник ће одустати од датог сајта и потражити сљедећи. Због тога је веома битно обратити пажњу на вријеме одзива приликом креирања веб апликација да би апликација била квалитетна и посјећена. Вријеме одзива представља период који прође од упућеног захтјева.

### *WWW* концепт

Концепт свјетска мрежа (*енгл.World Wide Web*) или краће веб је први пут заживио раних деведесетих година прошлог вјека. Веб се може дефинисати као скуп међусобно повезаних хипертекст докумената којима се приступа помоћу интернета. Развијен у оквиру *CERN-a* ( европска организација за нуклеарно истраживање) у Женеви, првобитно је служио научницима који су имали потребу да лако приступе одређеним информацијама које су биле садржане у виду хипертекст докумената. Од тог периода па до данас вeб није промијенио своје темеље на којима се базира:

* **Сервер-клијент архитектура** базирана на *HTTP* протоколу – представља подјелу улога међу рачунарима. Рачунари клијенти су оне машине које путем мреже шаљу одређене захтјеве за податке серверских машина које их опслужују. У контексту *WWW-а*, клијент је претраживач, најчешће лоциран на клијент-машини помоћу којег корисник шаље захтјев за одређену страну сервер-машину на којој је инсталиран веб сервер. Тражени документ се проналази а потом као одговор прослијеђује назад клијенту. Претходно описани поступак не би био могућ без *HTTP* комуникационог протокола помоћу којег клијент и сервер рачунари размјењују захтјеве и одговоре.
* **Начин именовања и референцирања ресурса (*URL*)** – *Unified Resoruce Locator*, или краће *URL*, представља текстуални низ који репрезентује веб адресу одређених ресурса (докумената, слика…) који се похрањују на сервер рачунарима широм интернета.
* **Језик за генерисање хипертекст докумената (*HTML*)** – *HyperText Markup Language* или *HTML* је језик намијењен изради *HTML* докумената. Његову синтаксу чине тзв. ознаке (тзв. ‘тагови’) помоћу којих се уређује садржај и структура *HTML* докумената. Претраживачи не приказују ове ознаке или их тумаче како би се садржај *HTML* докумената приказао на жељени начин.

### Архитектура веб апликација

За изградњу веб апликације углавном се користе скрипт језици. Основна разлика између скрипта језика и програмских језика (*C, C++, Java, Visual Basic* …) лежи у томе што програмски код који је написан у програмским језицима мора прво бити компајлиран (тј. преведен у машински код) да би се могао извршавати.

Код скриптних језика програмски код се извршава у тренутку његовог покретања и то помоћу посебног програма (интерпретера) који истовремено интерпретира и извршава написани програмски код. Скриптни језици који служе за изградњу веб апликација подијељени су у двије велике групе.

* **Скриптни језици који се извршавају на клијент рачунарима** – (нпр. *JavaScript*,*VB script*) њихов угнијежђени код у *HTML* документу се извршава у оквиру претраживача на клијент рачунарима. Претежно се користи за структурирање приказа садржаја и уређивање интерфејса веб апликације, интеракцију са корисником и друго.
* **Скриптни и програмски језици који се извршавају на сервер рачунарима** – (нпр. *PHP*) група језика који углавном служе изградњи главних функционалности веб апликација. Дакле, помоћу њих се најчешће реализује пословна логика саме апликације.

Сама архитектура веб апликација је вишеслојне природе због чега имплементација оваквих рјешења захтијева познавање значајног броја веб и интернет технологија, скриптних и програмских језика, мрежне архитектуре, база података и друго.

С обзиром да се позивају и користе у окиру веб *browser*а (а знамо да их има више) могу да постоје међусобне разлике у приказу и функционисању исте веб апликације у различитим веб *browser-*има.

Ријеч је о међусобној некомпатибилности веб *browser-*а, јер се обично старије верзије одређених веб *browser-*а не узимају у обзир при имплементацији неког веб решења иако се има на уму што већа покривеност веб *browser-*а и њихове верзије.

Како се архитектура веб апликација заснива на архитектури клијент-сервер тиме се веб апликације најчешће опслужују са једног (или више) удаљеног сервера рачунара. Уколико дође до било какве опструкције у раду сервера, веб апликација постаје неупотребљива за вријеме док је сервер из било ког разлога недоступан.

Такође, угодност коришћења веб апликација зависи и од перформанси мрежне архитектуре и самих серверских рачунара и њихових могућности услуживања клијената који захтјевају одређене функционалности веб апликација.

### Типови вeб апликације

Најчешћа подјела веб апликација се врши по критеријуму намјене због којих се користе. Тако на примјер имају неке од сљедећих типова веб апликација као што је приказано на Слици 1.2:

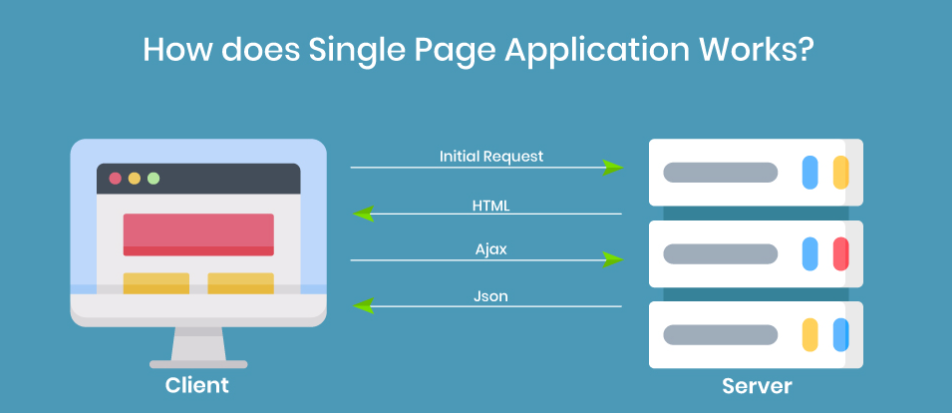
* друштвене мреже (*Facebook, MySpace, Twitter*)
* апликације за уређивање и дијељење слика (*Snapfish, Flickr, Photobucket, Shutterfly*)
* календари (*Google* календар, *О2* календар, *Yahoo* календар)
* апликације за дописивање (*chat-avenue, omgpop, czateria, Meebo*)
* апликације за куповину путем интернета (*Amazon*)
* адресари (*Yellow.com, White Pages, AnyWho*)
* апликације за уређивање блогова (*Blogger, Word Press, Tumbir*)
* *webmail* сервиси (*Gmail, Yahoo Mail, HotMail*)
* апликације географских мапа (*Google Maps, MapQuest*)
* интернет претраживачи (*Google, Bing, Yahoo*)
* веб апликације других имена…



Слика 1. -Познате вeб апликације

## Oбјашњење *SPA* (*Single page application*)

*SPA* (апликација на једној страници) је имплементација веб апликације која учитава само један веб документ, а затим ажурира садржај тijeла тог појединачног документа преко *JavaScript APIs (енгл.**Application Programming Interface)* као што су *XMLHTTPRequest* и *Fetch* када треба да се прикаже различит садржај. Ово стога омогућава корисницима да користе веб-сајтове без учитавања читавих нових страница са сервера, што може довести до побољшања перформанси и динамичнијег искуства, са неким компромисним недостацима као што је оптимизација веб-сајта за претраживаче *SEO (енгл.* *Search Еngine Оptimization*), потребно је више труда да се одржи стање, имплементира навигација и да се ураде значајне перформансе праћење. *SPA*-ови могу брже да се учитавају, преузимају податке у позадини, *SPA* могу подржати инкрементална ажурирања, чување дјелимично попуњених образаца или докумената без потребе да корисник кликне на дугме да би подњео образац. *SPA* могу подржати богата понашања на страни клијента, као што је превлачење и испуштање, много лакше од традиционалних апликација. *SPA* -ови могу бити дизајнирани да раде у искљученом режиму, правећи ажурирања модела на страни клијента која се на крају синхронизују назад са сервером када се веза поново успостави. На Слици 1.3 је дат приказ како ради *SPA.*



Слика 1. -Како ради SPA

Архитектура апликације на једној страници је једноставна. Садржи технологије на страни сервера и клијента и постоје три избора:

**Рендеровање на страни сервера (*SSR*)**

* Претраживач упућује захтјев за *HTML* датотеку са сервера.
* Сервер набавља све жељене податке, издваја апликацију и производи *HTML* датотеку апликације у ходу.
* Корисници виде приступачан садржај.
* Оквир апликације са једном страницом је оно што спаја догађаје, креира виртуелни *DOM* (*eнгл.* *Document Object Model*) и извршава даље радње.
* Апликација је спремна за употребу.

Рендеровање на страни сервера се може изабрати јер повезује и брзину веб апликације на једној страници и не преоптерећује прегледач корисника, чинећи апликацију брзом.

**Рендеровање на страни клијента (*CSR*)**

* Прегледач води захтјев за *HTML* датотеку са сервера.
* Сервер брзо одговара једноставном *HTML* датотеком, заједно са повезаним скриптама и стиловима.
* Корисници виде празну страницу или одређену слику учитавача док *JavaScript* ради.
* Апликација набавља податке, генерише приказе и убацује их у *DOМ*.
* Апликација је спремна за употребу.

Ово може бити алтернатива за веб локације једноставног изгледа. Tумачење података на страни клијента укључује многе ресурсе са уређаја и то може преоптеретити претраживач корисника. Као посљедица тога, овај избор може бити спор у поређењу са све три алтернативе. Такође, за веб локацију са великим прометом, *CSR* би био пожељнији, јер би испоручио информације корисницима без пуно разговора са сервером.

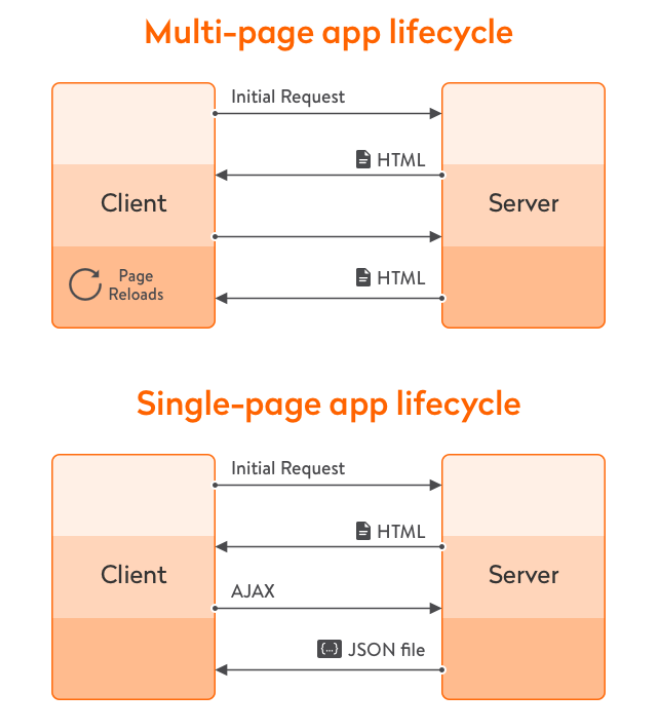
**Генератори статичког сајта (*SSG*)**

* Претраживач упућује захтјев за *HТМL* датотеку са сервера.
* Сервер затим брзо одговара са претходно организованом *HTML* датотеком.
* Корисници виде страницу.
* Апликација црта податке, производи приказе и убацује их у *DOM*.
* Апликација је спремна за употребу.

Сматра се да је то брза и добра опција, али генератори статичких сајтова (*SSG*) су кориснији за статичне странице.

### Разлике између *SPA* и *MPA (Multi-Page Apps)*

Апликације са више страница (*енгл.Multi Page Application)* или *МPА* су веб апликације које се састоје од више од једне странице и морају поново да учитају цijeлу страницу, чак и за учитавање мале иконе на веб страници. Ове апликације су велике и компликоване за развоj у поређењу са *SPА,* основна поређења су дата на Слици 1.4. Неки уобичајени примјери *МPА* су веб локације за е-трговину, попут Амазона.



Слика 1. -Разлика између SPA и MPA

SPA или MPA разлике:

* **Брзина**: је овдје важан фактор – временско трајање пажње људи је све краће. *SPA* се брже оптерећује. Пошто већину ресурса апликације учитава само једном. Страница се не учитава у потпуности кад год корисник затражи нови податак. *МPА* је спорији јер прегледач мора поново да учита цијелу страницу од нуле кад год корисник жели да приступи новим подацима или се пресели на други дио веб локације. Оптимално вријеме учитавања за веб локацију је 0,4 секунде. Ако је веб локација или апликација препуна слика, онда је одабир *SPА* сигурнија опција.
* **Квачило**: *SPA* је снажно одвојен, што значи да су фронтенд (*енгл. frontend*) и бекенда (*енгл. backend*) одвојени. Апликације на једној страници користе *АPI*-је које су развили програмери на страни сервера за читање и приказивање података. У *МPA* *frontend* и *backend* су више међусобно зависни. Сво кодирање се обично налази у оквиру једног пројекта.
* **Оптимизација претраживача**: jедна од слабости *SPA* је *SЕО (енгл.* *Search engine optimization)*. Нажалост, они нису тако прилагођени за *SЕО* као *МPА*. То је првенствено зато што се већина апликација на једној страници покреће на *JavaScript*-у, што већина претраживача не подржава. Веб странице се индексирају путем „пузања“ или „паука“. Покретачи претраживача преузимају *HTML* датотеке странице што олакшава рангирање статичких *HTML* веб страница. *МPA* омогућава боље позиционирање веб странице, јер свака страница може бити оптимизована за другу кључну ријеч. Такође мета ознаке могу бити укључене на сваку страницу – ово позитивно утиче на *Google* рангирање.
* **Корисничко искуство**: добро корисничко искуство више није опција већ услов. *SPA* центри су прилагођенији мобилним уређајима и то је нешто што вриједи запамтити јер велики промет долази са мобилних уређаја. Чак је и *Google* почео да даје предност мобилном искуству у односу на десктоп. Оквири примјењени у развоју *SPA* омогућавају вам да развијате мобилне апликације. *МPA*, с друге стране, омогућавају бољу архитектуру информација. Може се креирати онолико страница колико је потребно и може се укључити онолико информација на страници колико вам је потребно без икаквих ограничења. Навигација је јасна, тако да корисник може лако да се снађе на веб локацији, што позитивно утиче на њихово искуство.
* **Безбједност**: што је веб локација већа, потребно је више труда да се она обезбједи. Ако се ради са апликацијама на више страница са *МPА*, онда се мора обезбједити сваку веб страницу. У *SPA*-у, све што треба урадити да би се очувала страница безбједном је да се држе безбједне крајње тачке података, али не нужно и безбједније. *SPA*-ови су склонији хакерским нападима, јер раде на *JavaScript*-у, који не врши компилацију кода, што га чини рањивијим на малвер.
* **Процес развоја**: jедна од највећих предности *SPA*-а је позадински код за вишекратну употребу. Може се примјенити исти код који се користи у веб апликацији на матичну мобилну апликацију. То је важна информација, јер се апликације и веб локације често користе на мобилним. Захваљујући јасној подјели између *frontend* -а и *backend* -а, оба дјела се могу развијати истовремено, што убрзава цијели процес развоја. Развијање *MPA*-а траје дуже, јер у већини случајева серверска страна мора бити кодирана од почетка.
* ***JavaScript* зависност**: *SPA* живи и дише *JavaScript*, што може бити проблематично. Више претраживача је почело да подржава *JavaScript*, али са различитим резултатима. Ниво подршке у великој мјери зависи од *ЈS* оквира који се користи. Ако се апликација покреће у прегледачу са онемогућеним *JavaScript*-ом, то може да изазове проблеме са функционисањем апликације, што може довести до виших стопа посјете само једне странице и ниже конверзије. Ослањање на *JavaScript* такође доприноси проблемима са *SEO* оптимизацијом и безбједносним проблемима. *МPA* се могу изградити без икакве зависности од *JavaScript*-а.

# СПЕЦИФИКАЦИЈА И МОДЕЛОВАЊЕ СИСТЕМА

Под појмом систем подразумјева се организован скуп чинилаца, са јасно дефинисаним међусобним везама, који дјелују ради испуњења неког постављеног циља. Без обзира на начин на који је настао, сваки појединачни систем посједује:

* МИСИЈУ (циљ или скуп циљева због којих постоји)
* ОРГАНИЗАЦИОНУ СТРУКТУРУ (скуп чинилаца и везе међу њима које формирају топологију система)
* ФУНКЦИОНАЛНУ СТРУКТУРУ (активности које спроводи)
* ИНФОРМАЦИОНУ СТРУКТУРУ (свеукупност семантичких представа са којима и над којима обавља функционалне трансформације)
* УПРАВЉАЧКУ СТРУКТУРУ (принуду, унутрашњу или спољашњу, која усклађује укупно понашање система)
* ВЕЗЕ СA ОКРУЖЕЊЕМ – КОНТЕКСТОМ

Софтвер је скуп инструкција, података или програма који се користе за рад рачунара и извршавање одређених задатака. Софтвер, у свом најопштијем смислу, је скуп инструкција или програма који упућују рачунару да обавља одређене задатке. Софтвер је генерички термин који се користи за описивање рачунарских програма.

Сам ток развоја софтвера има више етапа подјеле, а једна подјела била би на:

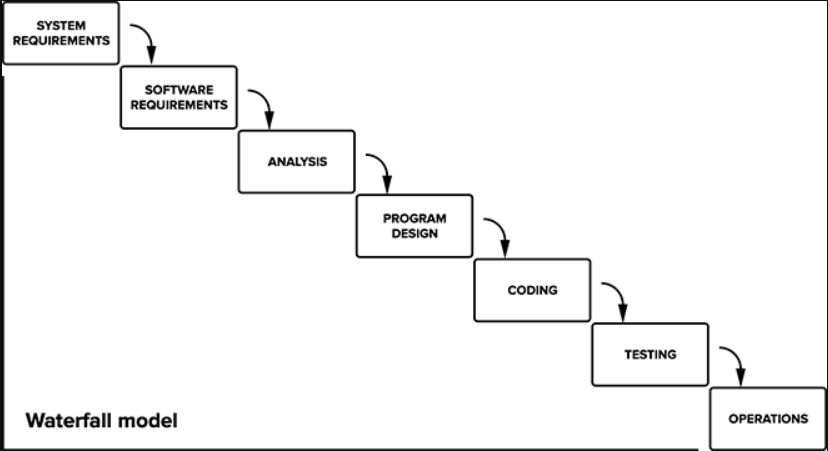
* планирање
* анализа захтјева
* дизајн
* програмирање
* тестирање
* коришћење у реалним условима

Прве три етапе представљају припремне фазе, којима се одређују план и методологија рада и обликује концепт апликације. Слиједи програмирање које резултује радном верзијом програма.

Програм се тада тестира и исправљају му се грешке, да би потом почело његово коришћење у пробном периоду, током кога се процијени колико програм одговара почетним захтјевима. Чак и кад програм потпуно одговара захтјевима, ангажман програмера се наставља на одржавању и унапређењу софтвера.

Суштински, ове етапе представљају 4 фазе класичног развоја – дефинисање шта програм ради, дефинисање како програм треба да ради, програмирање и коришћење

**Метод водопада** представља методологију према којој етапе у развоју софтвера дословно прате једна другу, почетак анализе захтјева, преко дизајна, програмирања, тестирања, па до инсталације и одржавања. Овај модел је представљен на Слици 2.1 . Свака фаза се комплетира и представља основ за сљедећу. При томе се свака етапа реализује веома пажљиво како се не би десиле грешке које би биле скупо плаћене у каснијим фазама. Метод водопада представља класичан, дисциплиновани начин развоја софтвера, који се у великој мјери ослања на детаљну документацију у свакој етапи.



Слика 2.1 -Модел водопада

## Функционални захтјеви система

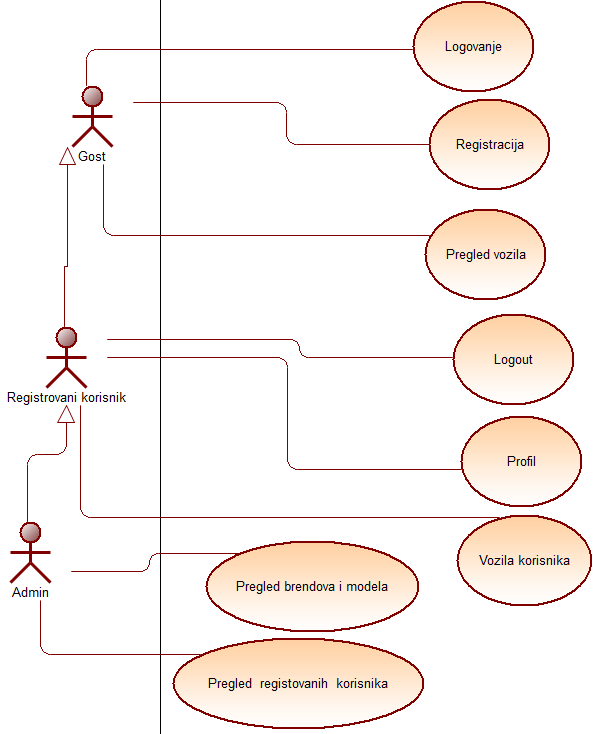
Представљају карактеристике производа или функције које програмери морају имплементирати да би омогућили корисницима да остваре своје задатке. Дакле, важно је да буду јасни и за развојни тим и за заинтересоване стране. Генерално, функционални захтјеви описују понашање система у специфичним условима. Приказ функционалних захтјева дат је у Табели 2.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број | Захтјев | Опис | Ризик |
| 1.0 | **Гост** |  |  |
| 1.1 | Приступ систему | Корисници система имају могућност самосталног приступа систему | висок |
| 1.2 | Прегед свих возила | Подразумијева могућност прегледа свих креираних возила на нивоу система. Приказ возила је сортиран према датуму и времену у опадајућем реду. | висок |
| 1.3 | Претрага возила | Подразумијева могућност претраге возила по бренду возила и по моделу возила | средњи |
| 1.4 | Логовање | Систем треба да обезбједи аутентификацију  корисника који му приступа. | висок |
| 1.5 | Регистрација | Систем треба да омогући регистрацију нових профила система | висок |
| 2.0 | **Регистровани корисник** |  |  |
| 2.1 | Одјава | Омогућена одјава из система | висок |
| 2.2 | Преглед профила | Кориснику је омогућено да има увид својих личних података | висок |
| 2.3 | Измјена профила | Кориснику је омогућено да измјени своје личне податке | висок |
| 2.4 | Измјена шифре | Омогућена промјена постојеће шифре са новом шифром | висок |
| 2.5 | Преглед возила | Омогућен преглед возила која је креирао улоговани корсиник | висок |
| 2.6 | Измјена возила | Омогућена измјена података одређеног возила које је креирао корисник | висок |
| 2.7 | Брисање возила | Омогућено брисање одређеног креираног возила | висок |
| 2.8 | Креирање возила | Омогућено креирање новог возила | висок |
| 3.0 | **Админ** |  |  |
| 3.1 | Преглед профила | Омогућен преглед свих креираних профила на систему | висок |
| 3.2 | Брисање профила | Брисање одређемог профила | висок |
| 3.3 | Креирање бренда | Додавање нових брендова аутомобила у систем | висок |
| 3.4 | Креирање модела | Додавање нових модела брендова у систем | висок |
| 3.5 | Брисање бренда | Брисање одређеног бренда | висок |
| 3.6 | Брисање модела | Брисање одређеног модела | висок |

Табела 2.1 -Приказ функционалних захтјева

## Use-case model

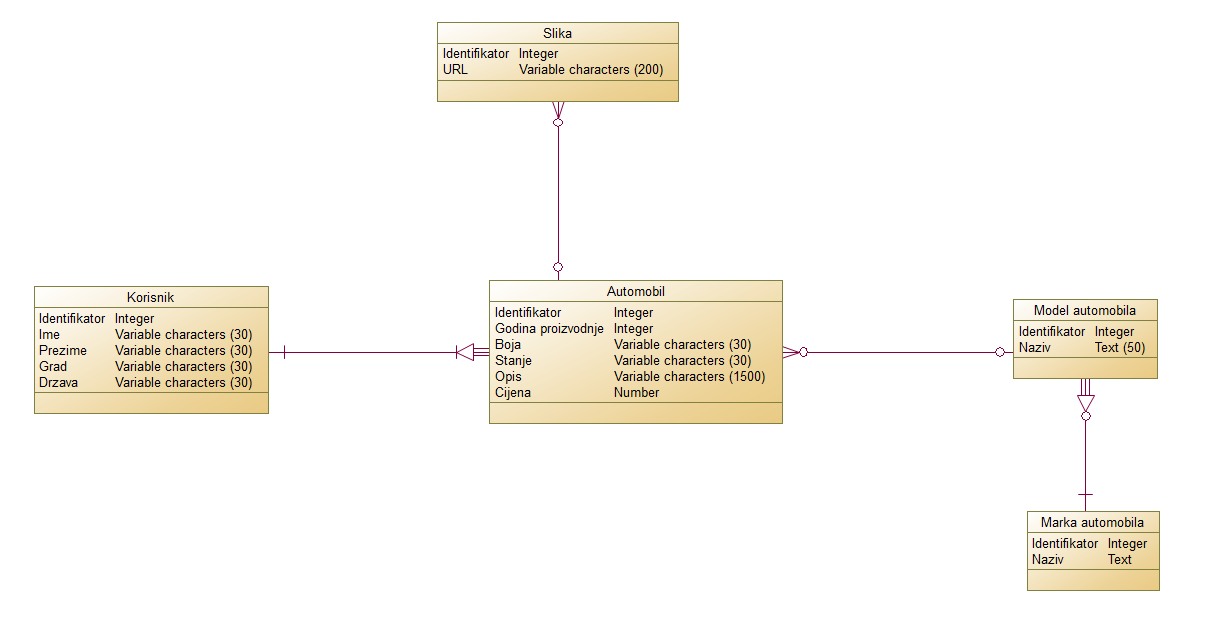
Описује предложену функционалност система. *Use case* представља дискретну јединицу интеракције између корисника (људског или машине) и система. *Use case* модел је приказан на Слици 2.2.



Слика 2.2 -Use Case модел

## Koнцептуални модел

Концептуална шема је опис информационих потреба на високом нивоу који је заснован на дизајну базе података. Обично укључује само главне концепте и главне односе међу њима. Обично се ради о моделу који је прво исечен, без довољно детаља за изградњу стварне базе података. Концептуални модел је приказан на Слици 2.3.



Слика 2.3 -Концептуални модел

# ПРЕГЛЕД КОРИШТЕНИХ ТЕХНОЛОГИЈА

За имплементацију пројекта су кориштене различите технологије за развој *frontend*-а и *backend-*а. Бекенд рађен у *ASP.NET Core* технологији уз употребу *Entity Framework Core* – a и *MSSQL* базе података, док је фронтенд рађен у *React*-у уз употребу *Redux-*а*.*

Поред наведених софтвера и софтверских алата, при реализацији пројекта коришћени су и технологије на серверској страни *Swagger,Microsoft Azure,* технологије на клијентској страни *HTML ,CSS, TypeScript*

## *Microsoft SQL Server*

*Microsoft SQL Server* је систем за управљање релационим базама података *RDBМS* (*енгл. relational database management system*) који је развио *Microsoft*. Овај производ је направљен за основну функцију складиштења преузимања података и преузимањем како то захтијевају друге апликације. Може се покренути или на истом рачунару или на другом преко мреже. *Microsoft* пласира најмање десетак различитих верзија *Microsoft* *SQL* *Servera*, дизајнираних за различиту публику и радна оптерећења у распону од малих апликација за једну машину до великих апликација окренутих према интернету са много истовремених корисника.

Основни циљ релационе базе података је да складишти податке у једну или више табела које се састоје од редова и колона, гдје сваки ред у табели посједује јединствени кључ који га једнозначно описује. Колоне се другачије називају и атрибутима (својствима). Свака табела представља један ентитет, гдје ред представља једну инстанцу (примјерак) тог ентитета, а колоне представљају атрибуте те инстанце. Између ентитета могу да постоје везе којима се дефинише међузависност између података ентитета који се налазе на крајевима везе.

Историја *Microsoft* *SQL* *Servera* почиње са првим *Microsoft* *SQL* *Servera* произвео *DOM* — *SQL* *Server 1.0*, 16-битним сервером за *OS* / 2 оперативни систем 1989. године — и протеже се до данашњих дана. Данас, *Microsoft* подржава сљедеће верзије:

* *SQL* *Server* *2012*
* *SQL* *Server* *2014*
* *SQL* *Server* *2016*
* *SQL* *Server* *2017*
* *SQL* *Server* *2019*

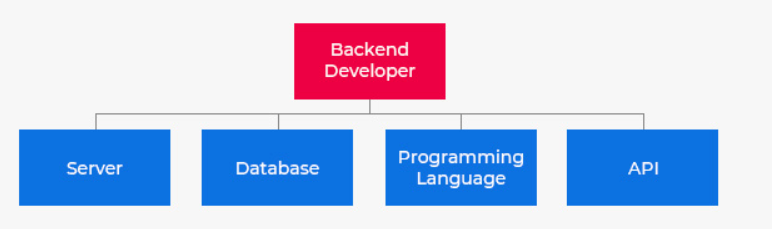
Oд *SQL* *Servera* *2016* па надаље, производ је подржан само на *x64* процесорима.

## Технологије на серверској страни

*Backend* развој укључује активности које се изводе на полеђини програма. Позадина веб странице састоји се од три главне компоненте – сервера, базе података и апликације за имплементацију логике као што је приказано на Слици 3.1. *Backend* развој функционише као невидљиви процес који шаље и прима информације, комуницира са *frontendom* и приказује податке у облику веб странице.

*Backend* ради са системским компонентама, пословним процесима и архитектуром података, као и писањем *API*-ја и библиотека. Различите врсте алата, језика и технологија користе се да би се постигли одређени захтјеви. Поред тога, користе се алати као што су *MongoDB, Oracle, MySQL* и *SQL* сервери за креирање, читање, ажурирање и брисање података, које затим сервирају назад на предњи крај апликације (на страни клијента).

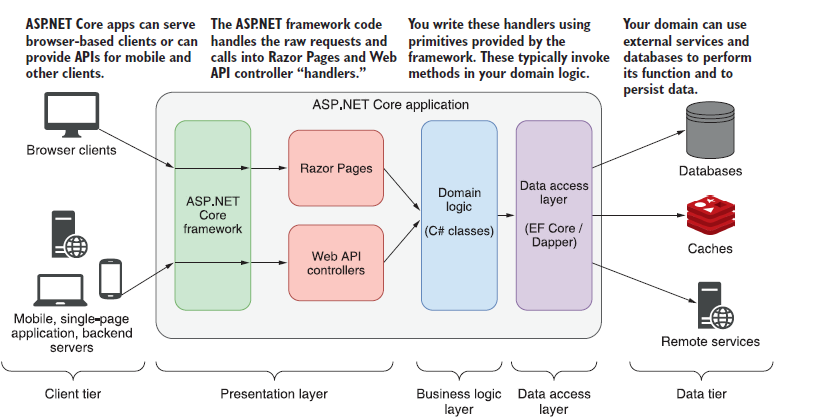
Обично се позадинске технологије односе на библиотеке језика на страни сервера који се користе за креирање серверске конфигурације веб локације. Позадинске технологије чине основу развоја софтвера. Да би се успоставила функционална веза корисничке апликације, серверске апликације и базе података могуће је користити велики број различитих технологија као што су *PHP*, *Ruby*, *Python*, *Java*, .*NET* за развој апликације на серверској страни.



Слика 3.1 -Backend развој

## *ASP.NET Core*

*ASP.NET Core* је вишеплатформски оквир за веб апликације отвореног кода које могу да се користи за брзу изградњу динамичких апликација које се приказују на страни сервера. Такође може се користити *ASP.NET Core* за креирање *HTTP* *API*s које могу да користе мобилне апликације, од апликације за једну страницу засноване на претраживачу, као што су *Angular* и *React*, или друге позадинске апликације. *ASP.NET Core* обезбјеђује структуру, помоћне функције и оквир за изградњу апликације. На Слици 3.2 је приказ *ASP.NET Core.*



Слика 3.2 -Приказ ASP.NET Core

Типична *ASP.NET Core* апликација се састоји од неколико слојева. Код *framework* –a *ASP.NET* *Core* рукује захтјевима клијента, бавећи се сложеним мрежним кодом. *Framework* код рукује сировим захтјевима и позива уводне *Razor Pages* и контролере *Web API*-ја. Оне позивају у логику *DOM-*ена апликације, а то су обично *С#* класе и објекти без икаквих зависности специфичне за *ASP.NET Core*.

Иако већ постоји велики број различитих радних оквира особине које *ASP.NET CORE* радни оквир издвајају од осталих су:

* Брзина рада

За разлику од интерпретерских језика, као што су нпр. *JavaScript* или *Ruby*, код .*NET*-а се компајлира, што знатно утиче на брзину рада. Асинхроно извршавање задатака такође битно утиче на брзину рада (Може бити 5 до 10 пута брже него код *Node.js*).

* Екосистем

Иако је *CORE* радни оквир прилично нов, .*NET* постоји већ дуже вријеме, тако да постоји велики број *NuGet* пакета као што су пакети за *JSON* десеријализацију, генерисање *PDF* докумената, конектори за базу података и многи други.

* Сигурност

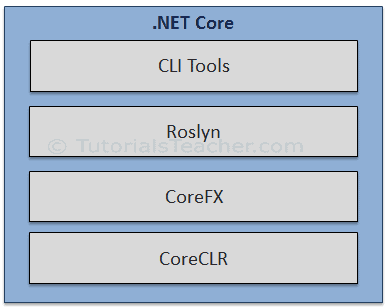
На сигурности *ASP.NET CORE* је рађено од самог почетка. Посједује велики број алата и библиотека за побољшање сигурности апликација. На једноставан начин рукује „тајнама“ унутар апликације, чува осјетљиве податке без излагања коду.

Историја *ASP.NET CORE* је приказана у Табели 3.1:

| Version | Latest Version | Visual Studio | Release Date | End of Support |
| --- | --- | --- | --- | --- |
| .NET 5 | Preview 1 | VS 2019 | 16th March, 2020 |  |
| .NET Core 3.x - latest | 3.1.3 | VS 209 | 24th March, 2020 | 12th March, 2022 |
| .NET Core 2.x | 2.1.17 | VS 2017, 2019 | 24th March, 2020 | 21st August, 2021 |
| .NET Core 1.x | 1.1.13 | VS 2017 | 14th May, 2019 | 27th May, 2019 |

Табела 3.1 -Историја .*NET* CORE радног оквира

*.NET Core* Састоји се од четири основне компоненте које су приказане на Слици 3.3:



Слика 3.3 -Архитектура .NET CORE радног оквира

***CLI Tools*** представља скуп алата који служе за развој софтвера.

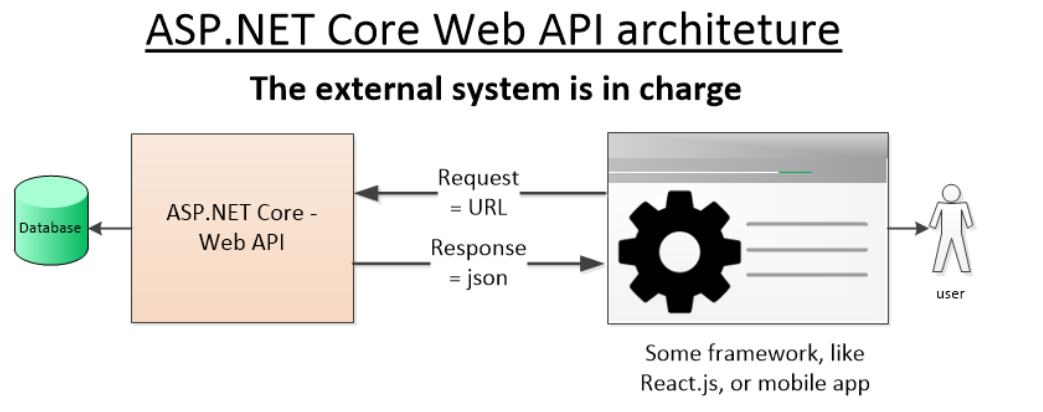
***Roslyn***је програмски преводилац за *C#* и *Visual Basic* програмске језике.

***CoreFX*** представља скуп билбиотека радног оквира.

***CoreCLR*** је извршни механизам (*енгл.* *execution engine*).

### Web Api

*ASP.NET Web API* је прошириви оквир за прављење *HTTP* базираних услуга којима се може приступити у различитим апликацијама на различитим платформама као што су веб, *Windows*, мобилни итд. Ради мање-више на исти начин као *ASP.NET MVC* (*енгл.model view controler)* веб апликација осим да шаље податке као одговор уместо *HTML* приказа. То је као веб сервис , али изузетак је што подржава само *HTTP* протокол. *ASP.NET Web API* подржава *Restful* апликације и користи глаголе *GET, PUT, POST, DELETE* за комуникацију са клијентима. На Слици 3.4 је приказана *Web API* архитектура.



Слика 3.4 -Приказ Web API

**Разлике између *ASP.NET Web API* и *ASP.NET MVC су приказане у Табели 3.2*:**

|  |  |
| --- | --- |
| *Web API* | *MVC* |
| користИ за генерисање *HTTP* услуга које одговарају само као подаци. | МОЖЕ КОРИСТИТИ *MVC* ЗА РАЗВОЈ ВЕБ АПЛИКАЦИЈЕ КОЈА ОДГОВАРА НА ПРИКАЗ И ПОДАТКЕ. |
| *Web API* враћа податке у различитим форматима, као што су *JSON, XML* и други формати на основу прихватања заглавља захтјева. | АЛИ *MVC* ВРАЋА ПОДАТКЕ У *ЈЅОN* ФОРМАТУ КОРИСТЕЋИ *JSONResult*. |
| подржавапреговарање садржају и само-хостовање | НЕ ПОДРЖАВА |
| *Web API* помаже у креирању *RESTful services*  преко *.Net Framework* | НЕ ПОДРЖАВА |

Табела 3.2 -Приказ разлика измеђуе *Web API* и *ASP.NET* MVC

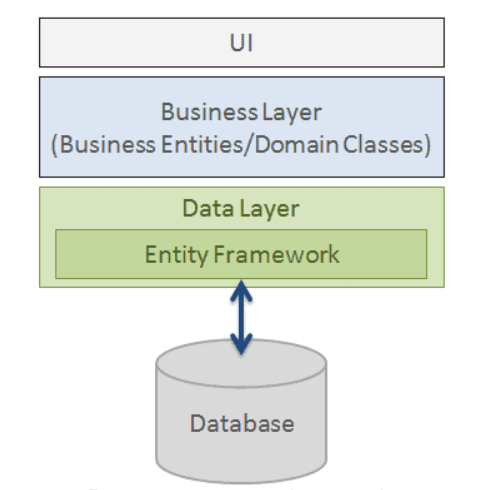
## *Entity Framework*

*Entity Framework* (Слика 3.5)је објектно-релациони мапер (*O/RM*) који омогућава *.NET* програмерима да раде са базом података користећи *.NET* објекте. То елиминише потребу за већином кода за приступ подацима који би програмери обично морали да напишу.

*Microsoft* је обезбједио оквир под називом *Entity Framework* за аутоматизацију свих активности у вези са базом података за одређену апликацију.

Објектно-релационо мапирање у рачунарству је техника програмирања за претварање података између система типова коришћењем објектно оријентисаних програмских језика. Ово ствара, у ствари, "базу података виртуелних објеката" која се може користити из програмског језика.

Омогућава програмерима да раде са подацима користећи објекте класе специфичне за домен без фокусирања на основне табеле и колоне базе података у којима се ови подаци чувају. Са Entity Framework, програмери могу да раде на вишем нивоу апстракције када раде са подацима и могу да креирају и одржавају апликације оријентисане на податке са мање кода у поређењу са традиционалним апликацијама.



Слика 3.5 -Entity Framework

Према горњој слици, *Entity Framework* се уклапа између пословних субјеката (класадомена) и базе података. Он чува податке ускладиштене у својствима пословних субјеката и такође преузима податке из базе података и аутоматски их конвертује у објекте пословних субјеката.

Kарактеристике *Entity* радног оквира су:

* Моделовање

Креира *EDM* (*Entity Data Model*), *POCO* (*Plain Old CLR Object*) базиране, ентитете са *get/set* методама за различите типове података. Користи овај модел у комуникацији са базом података.

* Упити

Користи *LINQ* упите код прибављања података из базе података, а провајдер базе података преводи *LINQ* упите у одговарајуће *SQL* упите.

Оставља могућност директног писања *SQL* упита за рад са базом података.

* Праћење измјена

Води рачуна о свим промјенама, унутар ентитета, које се касније морају сачувати у бази података.

* Штедња

*INSERT, UPDATE,* и *DELETE* команде се извршавају тек након позива *SaveChanges()* методе.

Посједује и асинхрону *SaveChangesAsync()* методу.

* Конкурентност

Користи „оптимистичну“ конкурентност, спрјечава пребрисавање измјењених података све док корисних не заврши са радом над тим подацима.

* Трансакције
  + Води рачуна о трансакцијама приликом рада са подацима.
  + Нуди и могућност измјене трансакционог процеса уколико је то потребно.
* Кеширање
  + Упити који се понављају више пута враћају резултате из кеш меморије умјесто из базе података директно.
* Уграђене конвенције

Прати уграђене конвенције програмирања и аутоматски подешава *EF* модел у складу са тим конвенцијама.

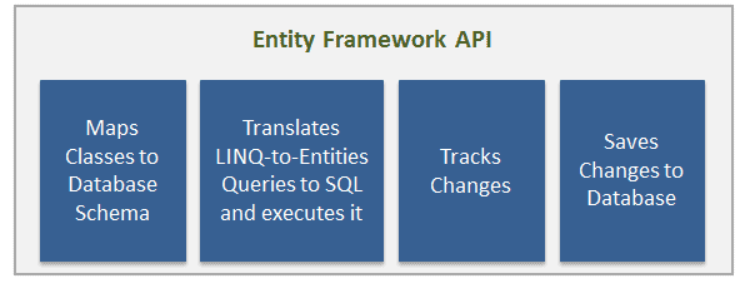
* Подешавања

Омогућава измјену *EF* модела (Модела по уграђеним конвенцијама) кориштењем анотационих атрибута или *Fluent API-а*.

* Миграције

Нуди скуп команди за креирање и или измјену базе података.

***Entity Framework* API** (*EF6 & EF Core*)) укључује могућност мапирања класа домена (ентитета) у шему базе података, превођења и извршавања *LINK* упита у *SQL* , праћења промјена извршених на ентитетима током њиховог животног вијека и чувања промјена у бази података као што је приказано на Слици 3.6.

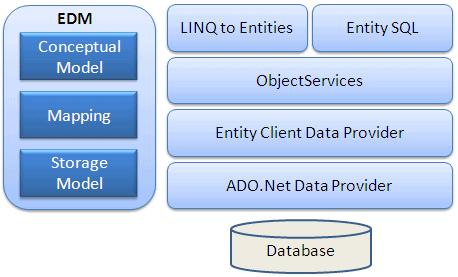


Слика 3.6 -Entity Framework API

### *Entity Framework Architecture*

Компоненте *Entity* радног оквира су (Слика 3.7):

* *Entity* модел података (*енгл.* *Entity Data Model*) гради помоћи EF API и који садржи:
  + Концептуални модел
    - Садржи класе модела и везе између њих независно од дизајна базе података.
  + Модел за складиштење је дизајн базе података и садржи:
    - Табеле
    - Погледе
    - Ускладиштене процедуре
    - Везе и кључеве
  + Пресликавања
    - Садрже инфромације на који начин концептуални модел, треба да се преслика у модел за складиштење.
* *LINQ to Entities*
  + Користи се за писање упита према објектном моделу. Враћа ентитете дефинисане у концептуалном моделу.
* *Entity SQL*
  + Користи се за писање упита према објектном моделу али само за *EF 6*.
* Објект сервис
  + Главна тачка размјене података између базе података и апликације.
* *Entity Client Data Provider*
  + Главни задатак ове компоненте јесте превођење *LINQ-to-Entities* или *Entity SQL* упита у упите *SQL* језика. Комуницира са *ADO.Net data provider-ом* који шаље или прикупља податке из базе података.
* *ADO.Net Data Provider*
  + Слој за комуникацију са базом података.

**

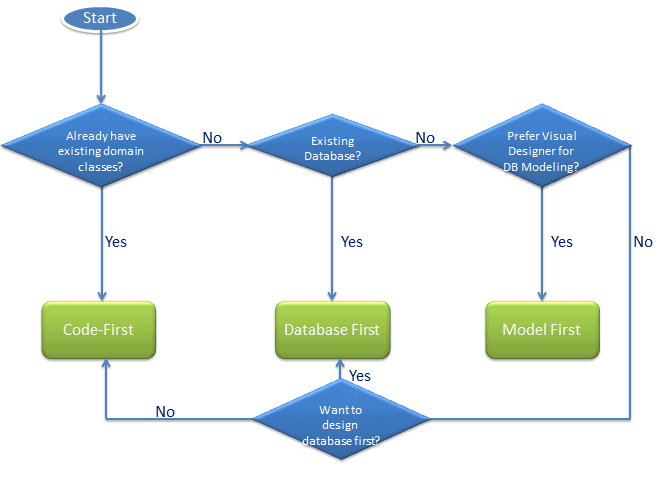
Слика 3.7 -Архитектура Entity радног оквира

### Приступи развоју

Постоје три различита приступа развоју:

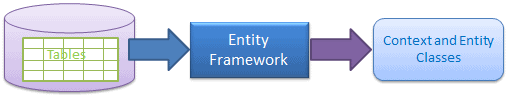
* База прво (*енгл*. *Database-First*)
* Код прво (*енгл*. *Code-First*)
* Модел прво (*енгл*. *Model-First*)

За који од приступа се одлучити зависи од архитектуре читаве апликације, приоритета и од ресурса. Дијаграм који може помоћи при одлучивању је дат на Слици 3.8.



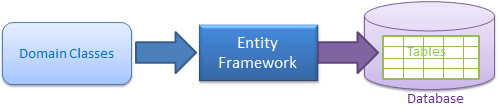
Слика 3.8 -Дијаграм одлучивања код развоја базе података (EF)

„База прво“: контекст и ентитети се генеришу на основу већ постојеће базе података (Слика 3.9).



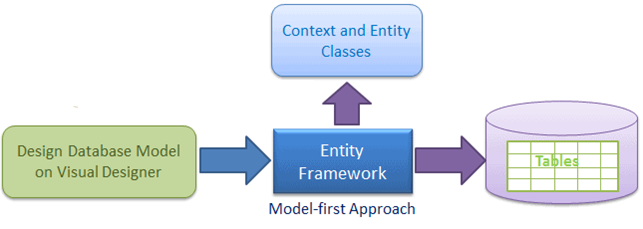
Слика 3.9 -Приступ „База прво“

„Код прво“: база података се генерише на основу већ постојећих доменских класа и контекстне класе (Слика 3.10).



Слика 3.10 -Приступ „Код прво“

„Модел прво“: контекст, ентитети и база података се генеришу на основу модела базе података (Слика 3.11).

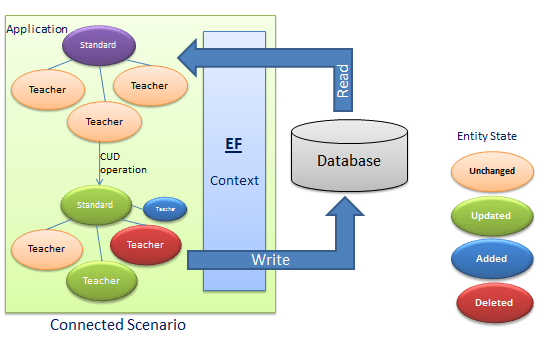


Слика 3.11 -Приступ „Модел прво“

### Сценарио развоја

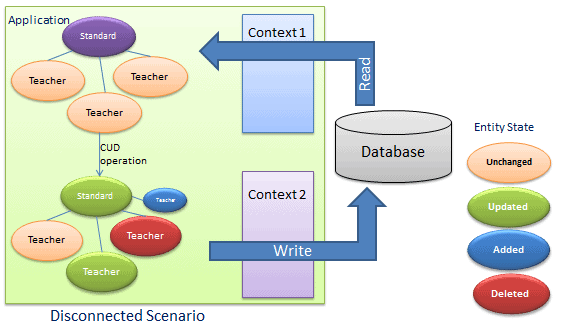
У зависности од надлежности контекстне класе постоје повезани сценарио (*енгл*. *Connected Scenario*) и неповезани сценарио (*енгл*. *Disconnected Scenario*).

Код повезаног сценарија (Слика 3.12) контекстна класа служи и за читање и за уписивање (чување) података. Прати измјене свих ентитета током читавог њиховог животног циклуса.Овај сценарио је користан ако се користи локална база података или база података која је на истој мрежи.



Слика 3.12 -Повезани сценарио

Неповезани сценарио (Слика 3.13) користи двије различите истанце контекстне класе за читање и за уписивање података. Након читања података, контекстна класа се уништава, а креира се нова истанца контекстне класе за уписивање података .Пошто истанца контекстне класе не води рачуна о ентитетима потребно је поставити одговарајуће стање ентитета, кориштењем методе *SaveChanges()*, прије уписа у базу података.

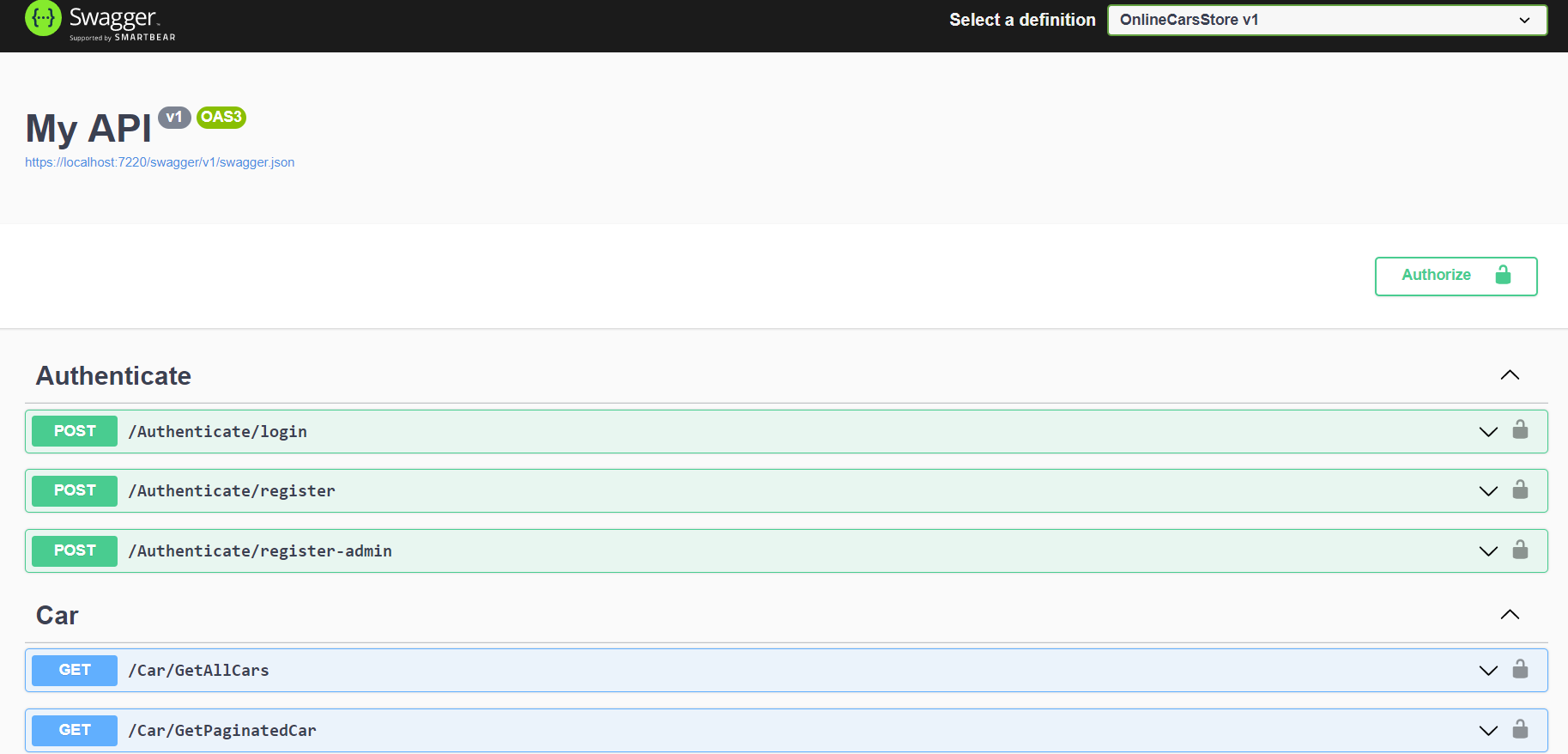


Слика 3.13 -Неповезани сценарио

## *Swagger UI*

*Swagger UI* омогућава развојном тиму или крајњим потрошачима — да визуализује и комуницира са ресурсима *API-*а без икакве логике имплементације. Аутоматски се генерише из *OpenAPI* (раније познате као *Swagger*) спецификације, са визуелном документацијом која олакшава позадинску имплементацију и потрошњу на страни клијента.

Swagger је језик који се користи за описивање *RESTful API* интерфејса. У апликацији је конфигурисан тако да ће приказати све методе на контролерима. *Swagger* *UI* који ће генерисати кориснички интерфејс за нас на основу података преузетих током покретања апликације. На Слици 3.14 је приказан примјер изгледа *Swagger UI* из апликације.



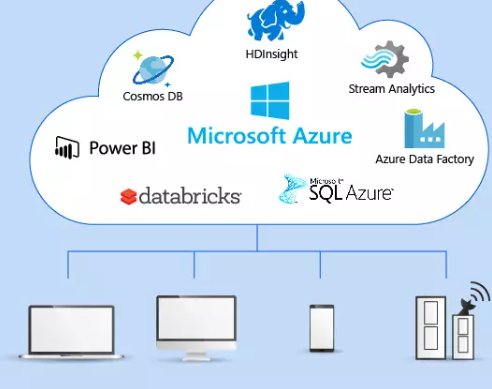
Слика 3.14 -Изглед Swagger UI у апликацији

## *Microsoft* Azure

Данас, апликације и платформе рачунарства које користе ***cloud*** брзо расту у свим индустријама, служећи као ИТ инфраструктура која покреће нова дигитална предузећа. Рачунарство које ради са *cloud* је технологија која омогућава приступ различитим рачунарским ресурсима путем интернета. Приликом повезивања добије се приступ рачунарским ресурсима, који могу укључивати рачунарство сервера, виртуелне машине, складиште и разне друге ствари. У основи, добављачи услуга у *cloud* имају огромне центре података који садрже стотине сервера, система за складиштење података и компоненти које су критичне за многе врсте организација. Ови центри података су на безбједним локацијама и чувају велику количину података. Корисници се повезују са овим центрима података како би прикупили податке или их користили када је то потребно. Корисници могу користити различите услуге: на примјер, ако желе обавјештење сваки пут када неко пошаље поруку или е-пошту, услуге у *cloud* могу помоћи. Најбољи дио у вези са платформама у *cloud-*у је то што се плаћају само услуге које се користе и нема унапријед наплата. Рачунарство које користи *сloud* се може користити у различите сврхе: машинско учење, анализа података, складиштење и резервне копије, стримовање медија и још много тога.

***Azure*** је платформа за рачунарство у cloud и портал на мрежи који омогућава приступ и управљање услугама на *cloud*-у и ресурсима које пружа *Microsoft*. Ове услуге и ресурси укључују чување ваших података и њихову трансформацију, у зависности од захтјева. Да би добили приступ овим ресурсима и услугама, све што треба је активна интернет веза и могућност повезивања на *Azure,* *portal.Azure* подржава више програмских језика, укључујући *Java, Node Js*, и *C#*

Још једна предност *Azure*-а је број центара података које има широм свијета. Постоје 42 *Azure* дата центра распрострањена широм свијета, што је највећи број центара података за било коју платформу у облаку. Примјер изгледа *Microsoft Azure-*а је приказан на Слици 3.15



Слика 3.15 -Microsoft Azure

Различите употребе *Azure*:

* Развој апликација: Може креирати било коју веб апликацију у *Azure*.
* Тестирање: Након успјешног развоја апликације на платформи, може се тестирати.
* Хостовање апликација: Када се тестирање заврши, може помоћи да хостује апликацију.
* Креира виртуелне машине: Може да креира виртуелне машине у било којој конфигурацији уз помоћ *Azure*.
* Интегришите и синхронизујте функције: *Azure* омогућава да интегрише и синхронизује виртуелне уређаје и директоријуме.
* Прикупљање и складиштење метрике: *Azure* омогућава да прикупља и чува метрике, што може помоћи да се пронађе оно што функционише.
* Виртуелни чврсти дискови: Ово су екстензије виртуелних машина; обезбјеђују огромну количину складиштења података.

## Технологије на клијентској страни

Кориснички интерфејс (*UI*) који дефинише шта сваки дио сајта или апликације ради и како ће изгледати.

***HTML*, *CSS*, и *JavaScript***: Ова три језика су неопходна за свакога ко жели да ради у *front-end* развоју. *HTML*, *CSS*, и *JavaScript*: раде заједно како би одредили изглед и функционалност странице.

***Frameworks***: Су алати потребни да би *JavaScript* и *CSS* радили онако како је планирано. Њихово добро разумjевање је кључно за креирање структура страница.

**Алати и софтвер за програмере**: Софтвер као што је контрола верзија, који прати и контролише промjене у изворном коду је од кључног значаја да би омогућио унос измјене без почетка. Разумијевање како да се користи много различитих алата за развој софтвера је блок за успjешну каријеру.

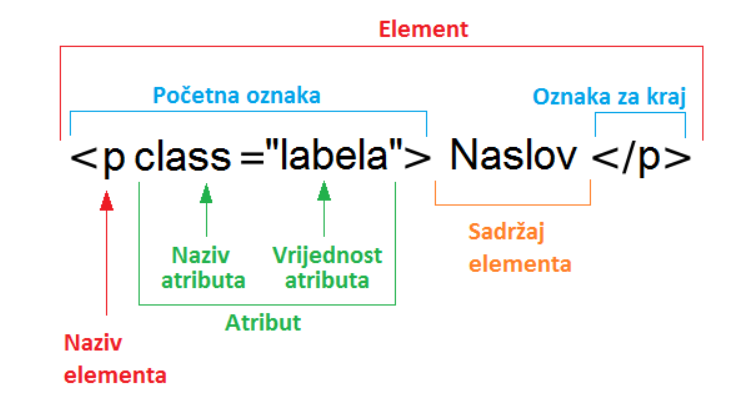
## *HTML*

*HTML* (*енгл*. *HyperText* *Markup* *Language*) је скуп симбола за означавање и кодова смјештених у датотеку која је намијењена за приказивање на веб страници. Ови симболи за означавање и кодови идентификују структурне елементе као што су пасуси, наслови и листе. *HTML* се такође може користити за постављање медија (као што су графика, видео и аудио записи) на веб страницу и описивање образаца за попуњавање подацима. Веб *browser* тумачи код за означавање и приказује страницу. *HTML* дозвољава приказивање информација независно од платформе у мрежи. Без обзира на ком типу рачунара је направљена веб страница, било који веб *browser* покренут у било ком оперативном систему може је приказати.

### *HTML* елементи

Дефинишу структуру и садржај *HTML* документа,представљају контејнере у којима могу да се налазедруги *HTML* елементи или текстуални садржај. Свака страница на вебу има бар неколико *HTML* елемената. Стога је саставни дио сваке веб странице.

Основни градивни елементи *HTML* језика:почетни таг нпр. <div> ,завршни таг </div>, само почетни нпр. <hr>, самозатварајући </br>, ознаке без садржаја <div></div>, *case insensitive*, по конвенцији се користе мала слова. На Слици 3.16 је приказ *HTML* елемента.



Слика 3.16 -HTML element

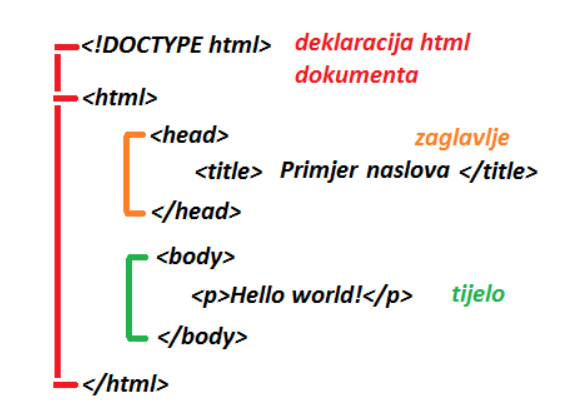
**Релације између елемената:**

* *Рarent*: Елемент који садржи други елемент је његов родитељ
* *Children*: Елементи који су садржани у другом елементу су његова дјеца
* *Descendants*: Елементи садржани у подређеним елементима, угнијежђени елеменати
* *Brothers and sisters*: Елементи који имају исти "родитељ"

*DOM* модел документа у виду стабла са видљивим односима између елемената

### Начин употребе

Сваки *HTML* документ мора да почиње са <*HTML*> и такође се завршава са </*HTML*>. Унутар тих ознака су друге групе, тј. <head> и <body> слиједе одмах иза њих. <head> се такође назива заглављем, а унутар ових ознака се уноси наслов <title> документа. Такође је важно да се повежу спољне датотеке, јер је добра пракса позивати *CSS* или *JavaScript* датотеке у заглављу. <body>, тј. тијело садржи све остале информације као што су текст, слике и други елементи. На Слици 3.17 је приказана декларација *HTML* документа.



Слика 3.17 -Декларација HTML документа

## *CSS*

Користећи *CSS*-a, дефинишемо правила у листи стилова која одређују како да изгледа садржај описан одређеним *HTML* кодом. Коришћењем *CSS* обрасца можемо да контролишемо било који дио сегмента на интернет страници: подешавање позадине (боја позадине, слике на позадини), уређивање маргина, контрола величине елемената, уређивање табела, уређивање текста.

*HTML* омогућава да се *CSS*-a повеже на три начина:

* унутар једног *HTML* елемента, преко атрибута "style" (линеарни *CSS*),
* унутар <head> дијела међу ознакама <style>, (интерни *CSS*),
* у посебној *CSS* датотеци, (екстерни *CSS*).

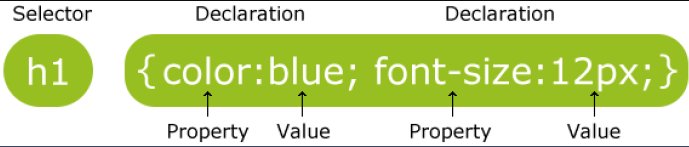
Такође је могуће референцирати више *CSS* докумената унутар једног *HTML* документа. Ако су атрибути дефинисани за исти селектор насљеђивање се примењује на више мјеста према следећем приоритету:

1. Линијски стилови
2. Уграђени стилови
3. Повезани стилови
4. Кориснички дефинисани стилови у веб претраживачу
5. Подразумијевани стилови веб претраживача

Атрибути који су дефинисани на више мјеста биће унапријед дефинисани са атрибутима вишег приоритета, остали задржавају своје вриједности.

### Синтакса

Дефинисана је листа правила која гласи: на који дио/дијелове документа важе правила и на које се примјењују правила форматирања изабрани дио/дијелови (декларације). Сваки стил се састоји из два дијела: селектора и декларације стила. Селектор означава елемент *HTML* документа на који ће бити примјењен стил. Он може бити конкретан *HTML* елемент, назив класе или *id* атрибут елемента. Декларација стила се састоји од *CSS* својства које се подешава (боја, фонт, оквир итд.) и његове вриједности.На Слици 3.18 је приказана *CSS* синтакса.



Слика 3.18 -CSS синтакса

## *JavaScript*

*JavaScript* интерпретациони програмски језик који се у потпуности извршава на клијентској страни, унутар веб прегледника. У комбинацији са *HTML* -ом језиком и *CSS* -ом, *JavaScript* чини *DHTML* (Динамички *HTML*). Он је скриптни језик који је развијен у фирми *Netscape* *JavaScript* је језик који подржава неколико парадигми, укључујући објектно оријентисану и функционалну парадигму програмирања.

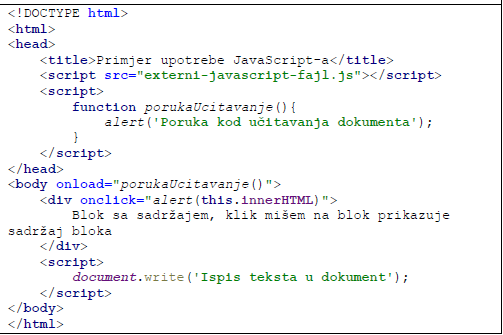
У новије вријеме се *JavaScript* у значајној мјери користи и за програмирање на серверској страни, првенствено у оквиру платформе *Node.js*, али се и даље може сматрати програмским језиком чија је главна примјена на клијентској страни.

Помоћу *JavaScript*-а веб странице су постале много динамичније него што је то случај код употребе само *HTML* и *CSS.* Омогућено је да се програмски приступа дијеловима документа, да се манипулише садржајем документа, додају или уклањају садржаје, детектују и обрађују догађаје, као и да корисник на бољи начин комуницира са документом.

Структуру података *JavaScript* програмског језика чине примитивни типови и објекти. Поред тога, одређена програмска логика може се да се извршава на клијентској страни, у оквиру веб *browser*а. Основне предности које пружа *JavaScript* су:

* *ЈаvaScript* је једноставан за употребу на веб страници. Потребно је имплементирати *ЈаvaScript* код у *HTML* документу и обавијестити веб *browser*.
* *ЈаvaScript* функционише на клијентском рачунару чак и када не постоји конекција са интернетом.
* Омогућава креирање интерактивних интерфејса који побољшавају корисничко искуство и пружају динамичке функционалности, без потребе за чекањем сервера да реагује и прикаже нову страницу.
* Учитавање садржаја у *HTML* документ је могуће онда када корисник то жели, без потребе за поновним учитавањем странице. Ово се углавном постиже употребом *AJAX* – a.
* *ЈаvaScript* може тестирати могућности веб *browser*а и извршавати се у складу са тим. Ова појава се назива принцип ненаметљивог програмирања.
* Омогућава кориговање грешки у подржаним веб *browser*има, као што је кориговање *CSS* стилова итд.

*JavaScript* код се са *HTML* документом може повезати на више начина, слично могућностима које има *CSS*. Дијелови *JavaScript* кода могу да се налазе у *HTML* елементима и њиховим атрибутима, затим у оквиру посебних *HTML* скрипта елемената, као и у екстерним фајловима који су са документом повезани помоћу елемента script . Сви дијелови кода заједно чине један комплетан *JavaScript* програм који се учитава заједно са документом и извршава по потреби. Одређени дијелови кода могу да се изврше одмах при читању документа, а неки други тек након позива функције или као реакција на неки догађај. Приказ *ЈЅ* у *HTML* документу је дат на Слици 3.19.

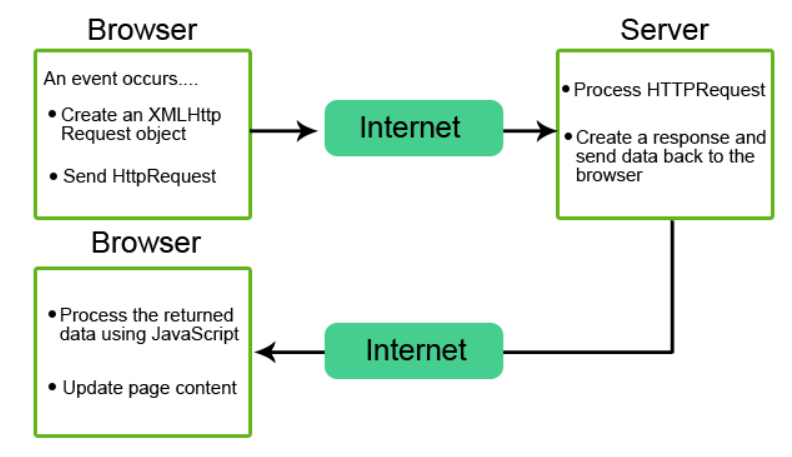


Слика 3.19 -Приказ ЈЅ у HTML документу

### Ajax

Асинхрони *JavaScript* у комбинацији са *XML*-ом односно *JSON*-ом чини *AJAX*, групу за динамичко креирање страница. Кориштењем *AJAX* -а побољшава се корисничко искуство, његова главна особина јесте динамичко освјежавање (динамичка промјена) одређеног дијела веб странице без потребе за поновним учитавањем исте.

Асинхроно понашање подразумијева да након интеракције корисника са интерфејсом, *JavaScript* генерише *XMLHTTPRequest* који се прослијеђује серверу приказујући резултате када они буду расположиви (Након одговора сервера), а у међувремену корисник може несметано да настави кориштење веб апликације. Рад *AJAX* –а је приказан на Слици 3.20.



Слика 3.20 -Рад Аjax-а

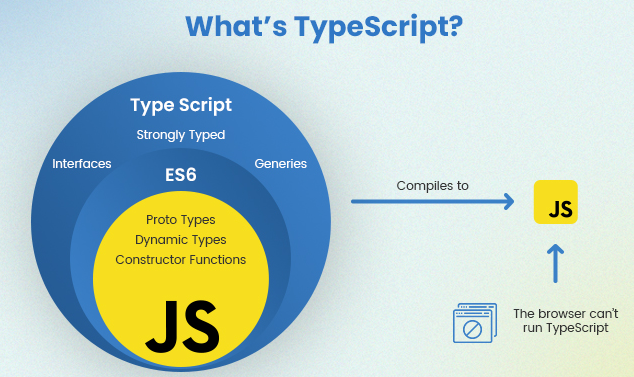
## *TypeScript*

*TypeScript* је програмски језик отвореног кода који омогућава примјену својстава објектно оријентисаних и типизираних програмских језика коришћењем *JavaScript* синтаксе (Слика 3.21). Као такав, брзо је постао популаран међу многим програмима који су навикли на могућности које нуде класично објектно оријентисани језици, као што је статичка провјера типова података, те једноставнији рад са класама и интерфејсима.

*TypeScript* код се не може извршавати директно у веб *browser-*у, он мора да се компајлира у стандардним *JavaScript* помоћу *TypeScript* компајлера. *TypeScript* се може користити самостално, могуће је у оквиру њега користити постојећи *JavaScript* код, као и позивати *TypeScript* код из *JavaScript* кода. *ТypeScript* је надскуп *JavaScript*, који садржи опционо куцање.

Поред тога, библиотеке *JavaScript*-а су такође компатибилне са *TypeScript*-ом. Испуњава исту сврху као *JavaScript* и може се користити и за апликације на страни клијента и на страни сервера. *TypeScript* је програмски језик који подржава и динамичко и статичко куцање. Обезбјеђује класе, опсеге видљивости, просторе имена, наслијеђе, уније, интерфејсе и многе друге карактеристике. Такође, нуди коментаре, промјенљиве, изјаве, изразе, модуле и функције. Може се извршити на *Node.Js* или било ком другом претраживачу који подржава *ECMAScript 3* или његове најновије верзије.

*TypeScript* обезбјеђује датотеке које могу да садрже податке о типу из тренутних објектних датотека. Дакле, друге апликације могу да користе вриједности дефинисане у датотекама само као *TypeScript* ентитете са статички откуцаним вриједностима.



Слика 3.21 -Структура TypeScript-а

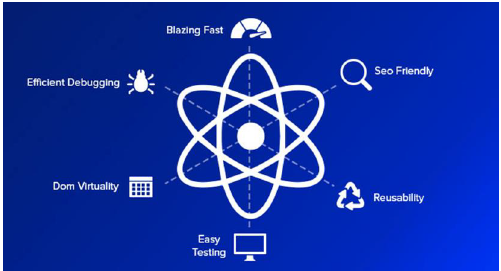
*JavaScript* језик је уведен као програмски језик на страни клијента. Али са употребом *JavaScript*-а за веб развој, програмери су научили да се он такође може сматрати програмским језиком на страни сервера. Међутим, *JavaScript* код је такође постао прилично сложен и тежак. Стога *JavaScript* није могао да испуни очекивања објектно оријентисаног програмског језика. Као резултат тога, *JavaScript* никада неће цвјетати као технологија на страни сервера у индустрији. Због тога је рођен *TypeScript* језик да премости овај јаз. *TypeScript* је језик отвореног кода за прављење великих веб апликација, док је *JavaScript* програмски језик на страни сервера који помаже у развоју интерактивних веб страница.

## *React*

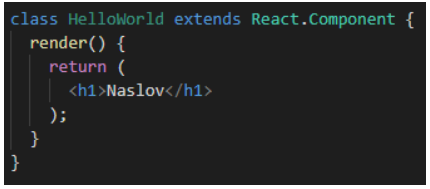
*React* је библиотека отвореног кода заснована на једноставном дизајну интерфејса која се базира на *JavaScript*.Одговоран је само за приказивање слојева у апликацији. Направио га је *Facebook*, првобитно направљени за сопствене потребе да би олакшали одржавање *Facebook-а*, али данас се почео користити за скоро сваку веб апликацију због своје лакоће коришћења. *React* користи виртуелни *DOM*, због чега има предности у односу на друге библиотеке и језике, јер страница не мора да се поново учитава када се одређени објекти промјене, али *React* мора знати који елемент треба замјенити, док остатак странице остаје непромијењен. У *React* библиотеци је могуће писати компоненте користећи одређени језик који се зове *JSX.* Рад *React-*а је приказан на Слици 3.22.

*JSX* омогућава писање кода попут *HTML*-a али се може комбиновати са *JavaScript*-ом примјер је дат на Слици 3.23. (*JSX*) je XML синтакса која конструише ознаке у *React*-у компоненте. *React* ради без *JSX*-а, али са *JSX*-ом је лакше читати и писати компоненте, као и структурирање било ког другог *HTML* елемента. *JSX* се компајлира током времена извршавања *JavaScript*-а.

*React* ће конвертовати код у виртуелни *DOM* и на крају испоручити *HTML*-a за корисникa. *React* брзо реагује на промјене стања у компонентама и поново рендерује компоненте унутар *HTML* *DOM*-а користећи виртуелни *DOM*. *React* апликације користе погледе (*енгл.*view) који се мијењају само када корисник кликне на другу страницу. Реаговати креира апликације које садрже само једну страницу *SPA*. Са *React*-ом, претраживач не мора поново да учитава цијелу страницу, већ се само мијења изглед те замијењује старим. *React* *Native* је популарно развојно окружење које програмерима омогућава прављење робусних мобилних апликација.

**

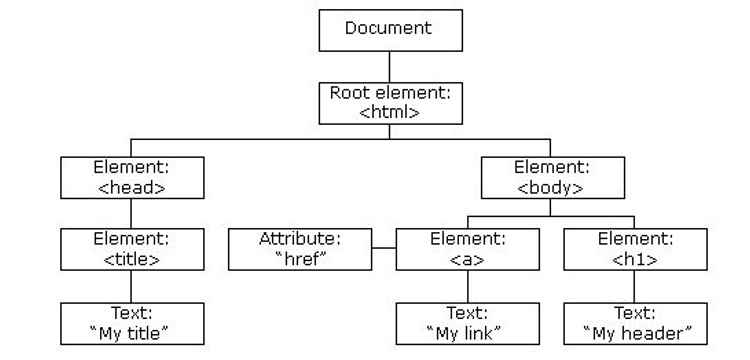
Слика 3.22 -Рад React-а



Слика 3.23 -Примјер JSX

### Virtuelni *DOM*

У апликацијама са једном страницом, претраживач у почетку учитава један *HTML* документ и док корисник користи апликацију, он заправо остаје на њеној страници, мијењају се само објекти унутар странице. *JavaScript* брише и креира нови кориснички интерфејс. Корисник ће имати осјећај да мијења странице, али у ствари је увек на истој *HTML* страници, а за то је одговоран виртуелни *DOM* (Слика 3.24). *React* је библиотека која је дизајниран за ажурирање *DOM* претраживача. Са *React*-ом страница нe комуницира директно са *DOM* програмским интерфејсом апликације. Умјесто тога комуницира са виртуелним *DOM*-ом или скупом инструкција које ће *React* користити за изградњу корисниковог интерфејса и интеракцију са претраживачем. Када се стање једног објекта промијени *VDOM* мијења само тај објекат унутар „стварног“ *DOM*-a умјесто да ажурира све објекте. Виртуелни *DOM* се састоји од *React* елемената који концептуално изгледају слично *HTML* елементима, али су заправо *JavaScript* објекти.



Слика 3.24 -DOM веб апликације

*DOM* посматра *HTML* или *XML* документ као стабло, а чворови стабла су елементи документа. Када се стање неког објекта у *React* апликацији промијени, *VDOM* се ажурира, а затим врши поређење тренутног стања са претходним и у „стварном“ *DOM-*у ажурира само измјењене елементе. Ова техника знатно убрзава приказ веб апликације јер ажурира само објекте чије се стање промијенило.

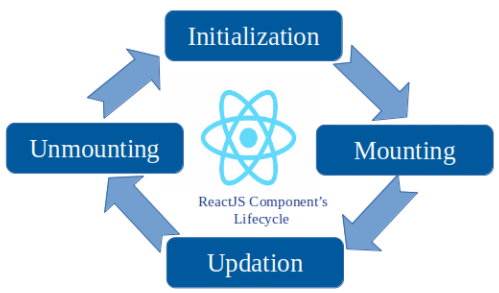
*React* посједује мноштво проширења. Омогућава приказивање на страни сервера (*енгл*. *Server-side rendering* ), страница се обрађује на серверској страни и шаље се кориснику као потпуно приказана страница.

### Животни циклуси компоненти

Постоје различите методе животног циклуса које *React* пружа у различитим фазама живота компоненте. *React* аутоматски позива одговарајућу методу на основу фазе у којој се налази компонента. Ове методе пружају бољу контролу компоненте.

Животни циклуси компоненте су (Слика 3.25):

* Иницијализација (*енгл*. *Initialization*)
* Уградња (*енгл*. *Mounting*)
* Ажурирање (*енгл*. *Updating*)
* Уклањање (*енгл*. *Unmounting*)



Слика 5 -Животни циклуси компоненте

Иницијализација компоненте подразумјева постављање стања и својстава компоненте (Обично путем конструктора).

У фази уградње компонента се уграђује у *DOM* и приказује по први пут. У овој фази су доступне двије методе:

* *componentWillMount()* након позива ове методе компонента се уграђује у *DOM.*
* *componentDidMount()* метода се позива након што је компонента уграђена у *DOM.*

Након што је компонента уграђена у *DOM* почиње трећа фаза, фаза ажурирања. У овој фази се компоненте, као одговор на интеракцију корисника, ажурирају и поновно приказују у складу са промјеном стања њиховог стања.

Методе које су доступне у овој фази су:

* *shouldComponentUpdate()* метода одлучујуе да ли се нека компонента треба ажурирати или не.
* *componentWillUpdate()* метода се позива након *shouldComponentUpdate()* методе и извршава се одмах прије поновног приказа компоненте.
* *ComponentDidUpdate()* метода се позива након што је завршен поновни приказ компоненте.

Посљедња фаза у животном циклусу је уклањање компоненте из *DOM*-a и посједује само једну методу *componentWillUnmount()* која се позива прије уклањања компоненте и тиме се завршава животни циклус једне компоненте.

### *React* *Hooks*

*Hooks* су функција која нам омогућава да користимо стање и друге *React* функције без потребе за писањем класе. Можемо рећи да су то функције које "закачу" *React* стање и карактеристике животног циклуса са компоненти функције.

*Hook* су такође карактеристичне јер имају два важна правила:

1. *Hook* се позивају само на највишем нивоу. Не позивајте их унутар петљи, угнијежђени функција или услова. Ово правило осигурава да се *Hook* позивају истим редоследу *DOM* –а сваки пут када се компонента поново рендерује.

2. Позив закачивања само из компоненти функције *React*. Не могу се позивати из редовних *JavaScript* функција. Умјесто тога, можете позвати *Hook* из компоненти функције *React*. Могу се звати и из ручно рађених кука.

Једна од главних предности *React*-а је начин на који синхронизује стање апликације са корисничким интерфејсом. Како се стање мијења, усљед интеракције корисника или ажурирања података из система или мреже. Реаговање интелигентно и ефикасно одређује које промјене треба да изврши у *DOM*-у у претраживачу или у корисничком интерфејсу уопште у другим окружењима. Ако упоредимо *Hook* са класама и њиховим функционалним компонентама, доћи ћемо до закључка да оне омогућавају чистије и транспарентније кодирање и код који се лако тестира, одржава и самим тим поново користи и „рециклира“. Функционалне компоненте са кукицама више не требају све методе животног циклуса јер се ефекти могу инкапсулирати у *Hook* -ове. Двије најпознатије и најчешће коришћене *Hook*-а су *useState* i *useEffect*.

### *Material UI*

*Material-UI* је једноставно библиотека која омогућава да се увезе и користи различите компоненте за креирање корисничког интерфејса у *React* апликацијама. Ово штеди значајну количину времена јер програмери не морају да пишу све од нуле.

Material-UI је *frontend* оквир отвореног кода за *React* компоненте који има 60.500 плус звјездица на *github*-у. Изграђен је користећи *Less* (*енгл.* *Leaner Style Sheets*) је језичка екстензија за *CSS* компатибилна са назад. Материјални кориснички интерфејс је заснован на *Google's Material Design* како би се обезбjедило висококвалитетно дигитално искуство уз развој *frontend* графике. *Material Design* се фокусира на пружање смjелих и оштрих дизајна – гради текстуре фокусирајући се на то како компоненте бацају сјенке и рефлектују свјетлост.

## *Redux*

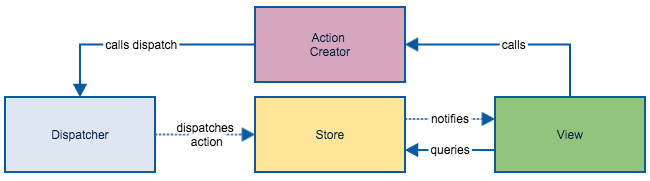
*Flux*  је архитектонски дизајн шаблон који је *Facebook* направио за развој *React*  апликација. Представља малу измјену *observer* дизајн шаблона, није ни библиотека ни радни оквир, и заснива се на концепту једносмјерног тока података.

*Redux* је библиотека заснована на *Flux* шаблону, може се сматрати примјеном *Flux* шаблона, олакшава контролу стања апликације и рукује приказом података при интеракцији корисника са апликацијом.

Сам по себи *React* није веома скалабилан, те приликом развоја великих апликација тешко је водити рачуна о стањима свих компоненти. Главни проблем је уска повезаност корисничких акција и промјена стања компоненти. Код сложених апликација често једна корисничка акција може утицати на низ промјена стања. Без *Flux*-a или *Redux*-а ово постаје веома сложен процес.

*Flux* дизајн шаблон се састоји од четири сегмента (Слика 3.26):

* *Action*
* *Dispatcher*
* *Store*
* *View*

****

Cлика 3.26 -Сегменти Flux дизајн шаблона

Треба да постоји само један *Dispatcher* сегмент, а може постојати више *Store* и *Action* сегмената. Акције се заснивају на функционалностима апликације, а на основу акција се мијења приказ апликације.

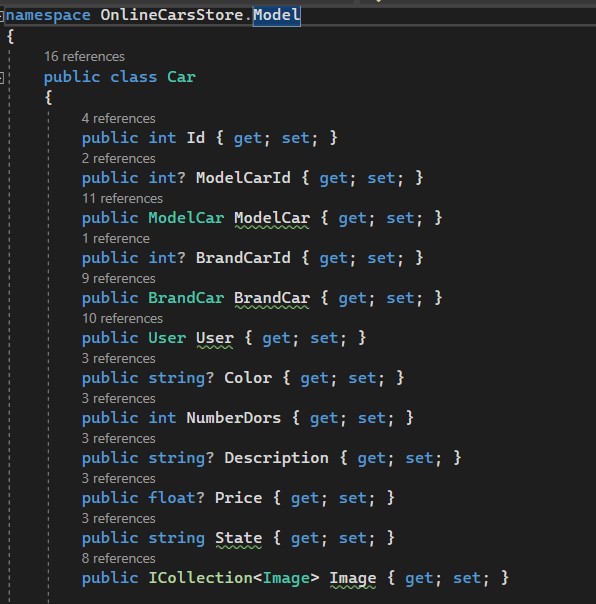
Код *Redux*-а *Dispatcher* је замијењен са редукторима (*енгл*. *Reducers*), а такође не постоји више *Store* сегмента, већ након сваке акције старо складиште се копира, примјењују се измјене и поставља се ново складиште. Сегменти Redux шаблона су приказани на Слици 3.27.



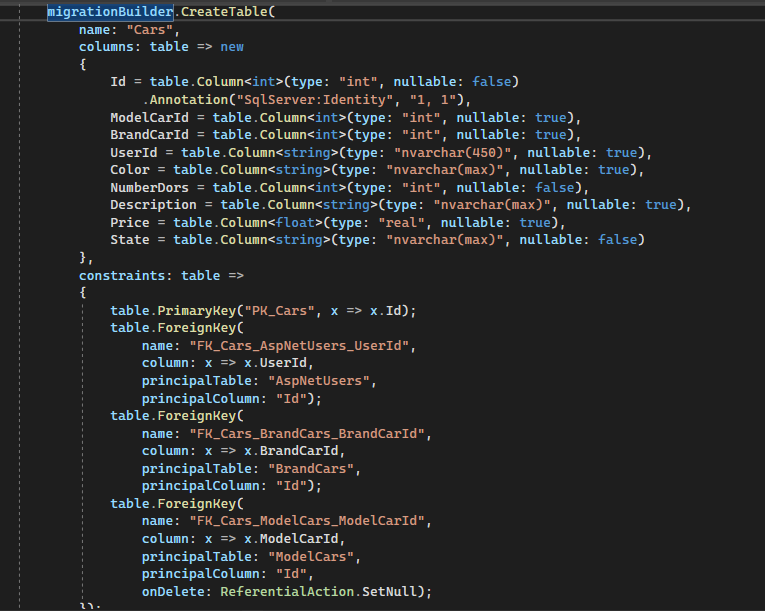
Cлика 3.27 -Сегменти Redux шаблона

# ИМПЛЕМЕНТАЦИЈА БАЗЕ ПОДАТАКА И БЕКЕНДА

Користећи технологије из претходног поглавља, систем за продају аутомобила је имплементиран тако што је кориштен *Code-First* приступ, односно прво су креирани модели (Слика 4.1) за ову апликацију, а затим уз помоћ миграција (Слика 4.2) креирана је и попуњена база података. Функција миграције у *EF Core*-у пружа начин за постепено ажурирање шеме базе података како би била синхронизована са моделом података апликације уз очување постојећих података у бази података. Све табеле носе генрализоване називе као што су *Сarѕ, BrandCars, ModelCars* и слично, те на тај начин општост система није нарушена ни на клијентској а ни на серверској страни. Овакав приступ обезбјеђује да систем буде прилагођен за било који сценарио.



Слика 4.1 -Примјер модела за табелу Car



Слика 4.2 -Примјер миграције за креирање табеле Car

## *Repository pattern*

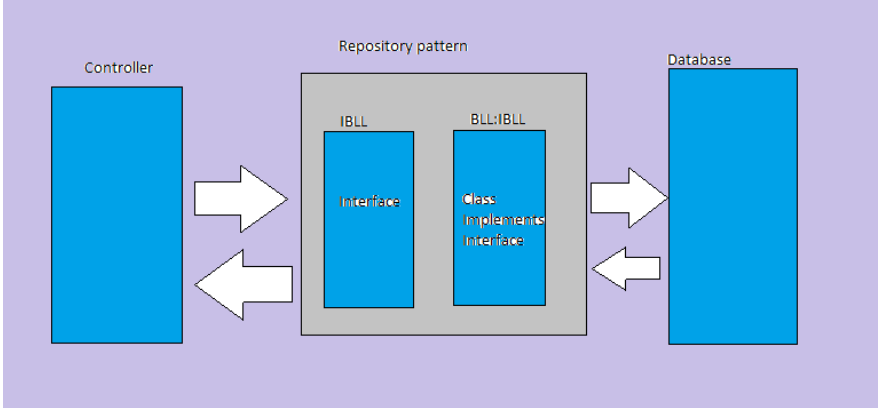
Шаблон спремишта (*енгл.repository pattern*) је образац дизајна који посредује податке од и до слојева приступа домену и подацима (као што је *Entity Framework Core / Dapper*). *Repositories* су класе које скривају логику потребну за складиштење или преузимање података. Тако да апликацији неће бити важно какву врсту објектно-релационог мапирања (*енгл.Object-Relational Mapping*) (*ОРМ*) користи, пошто се све што је у вези са *ОРМ*-ом рукује унутар слоја *repository*. *Repository pattern* је један од најчешће коришћених образаца дизајна за прављење чистијих рјешења. Намјера коришћења *Repository* *pattern* (Слика 4.3) је да се одвоји логика домена од *CRUD* операција извора података, тако да је могуће користити исту имплементацију домена са различитим складиштима података.

**Предности *repository pattern:***

* Логика приступа бази података и логика домена могу се тестирати одвојено са овим шаблоном.
* Развој вођен доменом је лакши.
* Чист, одржаван и вишекратни код
* Смањује редундантност кода, генеричко спремиште је боље рјешење од уобичајеног шаблона спремишта за смањење дуплицирања кода.
* Са овим шаблоном је лако одржавати логику централизованог приступа подацима.
* Коришћењем шаблона дизајна спремишта, може сакрити детаље о томе како се подаци на крају чувају или преузимају из складишта података (база података)
* Пружа флексибилну архитектуру.

**Недостаци** ***repository pattern:***

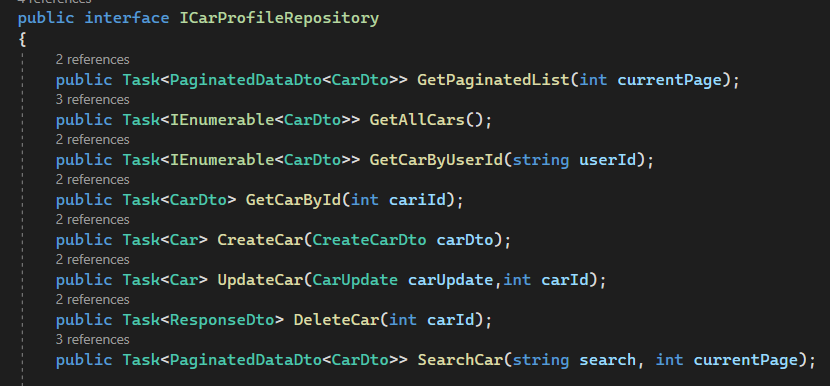
* Додатни слој апстракције - Због другог слоја апстракције одређени ниво сложености што га чини претјераним за мале апликације.
* Са шаблоном спремишта захтјева се креирање новог спремишта за сваки ентитет.
* Образац спремишта не одваја приступ подацима од складишта података и овдје крши приступ чистог кодирања.



Слика 4.3 -Repository pattern

***Repository Pattern Interface***:

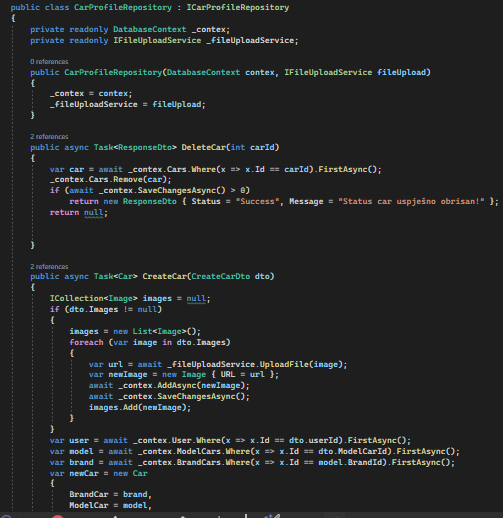
* Које операције (тј. методе) подржава repository
* Подаци потребни за сваку од операција, тј. параметри које треба прослиједити методи и подаци које метода враћа
* Интерфејс спремишта садржи шта може да уради, али не, како ради.
* Детаљи имплементације су у одговарајућој *repository class* која имплементира *repository interface* (Слика 4.4).



Cлика 4.4 -Repository Pattern Interface

***Repository pattern* - *SQL Server* имплементација:**

Сљедећа класа *CarProfileRepository* пружа имплементацију за *ICarProfileRepository*. Ова специфична имплементација ради са подацима из базе података *SQL Server*-а (Слика 4.5) .



Cлика 4.5 -Repository pattern - SQL Server

У класама фолдера контролера, програмер имплементира REST сервисе унутар контролера.

*REST* сервиси најчешће користе *HTTP* протокол и *HTTP* операције (методе) за манипулацију веб ресурсима. За сваку од ових метода, сервер генерише одговарајући одговор који корисничкој клијент апликацији преноси информацију о томе да ли је захтјевана метода извршена успјешно или је дошло до грешке приликом извршења или обраде. Неке од основних *HTTP* операција су:

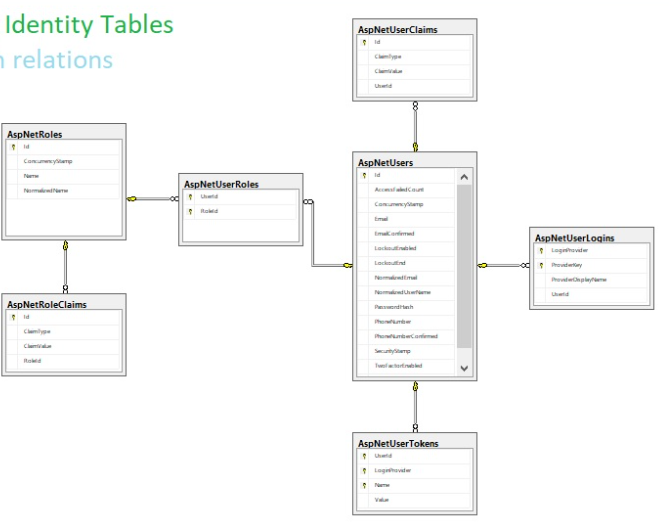
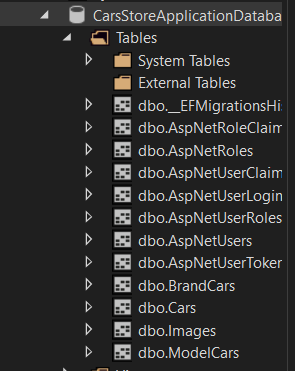
* *GET* – клијент захтјева читање ресурса,
* *POST* – клијент захтјева креирање ресурса,
* *PUT* – клијент захтјева ажурирање ресурса,
* *PATCH* – клијент захтјева ажурирање конкретних својстава ресурса,
* *DELETE* – клијент захтјева брисање ресурса.

Свака од ових операција се пресликава на неку од *CRUD* операција:

* *CREATE* – операција која креира нови запис у бази података,
* *READ* – операција чита запис из базе података,
* *UPDATE* – операција ажурира запис у бази података,
* *DELETE* – операција која брише запис из базе података.

## Модел базе

Ова веб апликација користи *MSSQL* релациони систем управљања базама података који заједно са *EF* *Core*-ом нуди једноставно и поуздано управљање подацима и корисницима. Обзиром да при изради апликације није прво креиран модел базе, већ је било потребно у коду креирати ентитете (доменске класе) и на основу кода генерисати базу података користећи *EF* *Core*, модел базе приказујемо уз помоћ *SQL* *Server* *Management Studio (SSMS).* Табела *Identity* и база података су приказани на Слици 4.6.



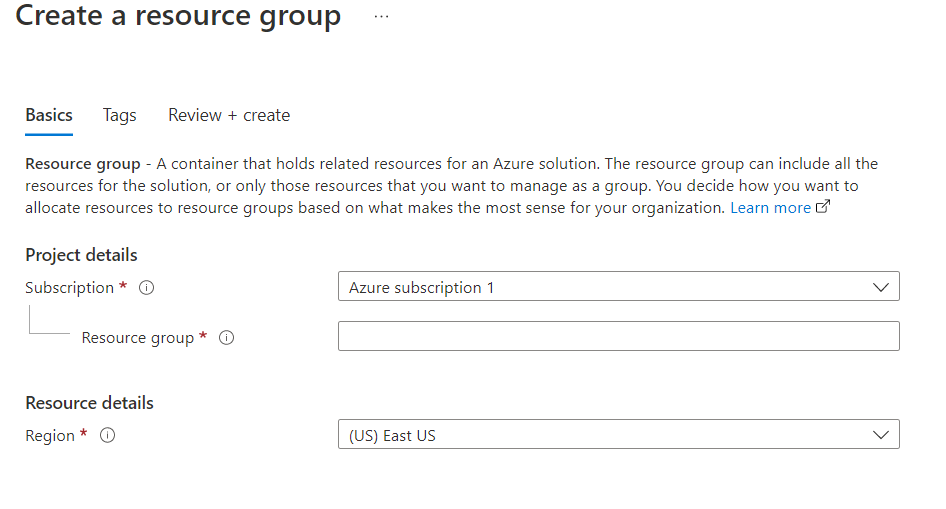
Cлика 4.6 -Табела Identity и база података

## Складиштење фајлова

За постављање и преузимање слика кориштен је *Microsoft* *Azure*.

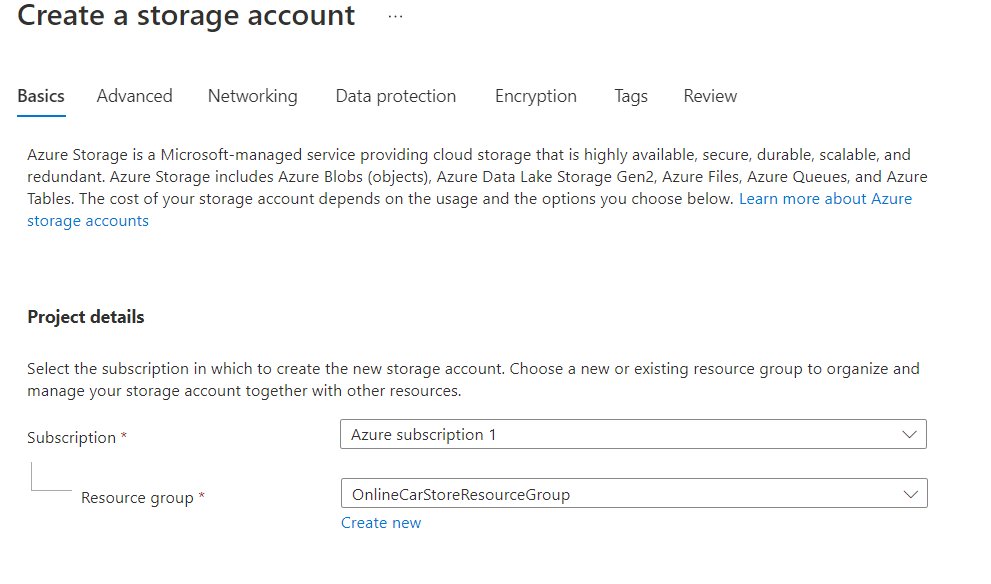
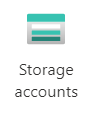
Кораци који су потребни да би успјешно искористили ресурс за складиштења слика су:

* Креирати групу ресурса (Слика 4.7) који представљају контејнер који садржи повезане ресурсе за *Azure* рјешење. Група ресурса може укључивати све ресурсе за рјешење или само оне ресурсе којима желите да управљате као група. У *Azure* -у, ресурс је ентитет којим управља Азуре. Виртуелне машине, виртуелне мреже и налози за складиштење су све примјери Азуре ресурса. *Subscription* у *Azure*-у је контејнер који садржи колекцију повезаних пословних или техничких ресурса. Ресурси се користе и наплаћују као група. Један или више центара података који се налазе у одређеном географском подручју и повезани су мрежом ниске латенције називају се *Regions*.



Cлика 4.7 -Resource group

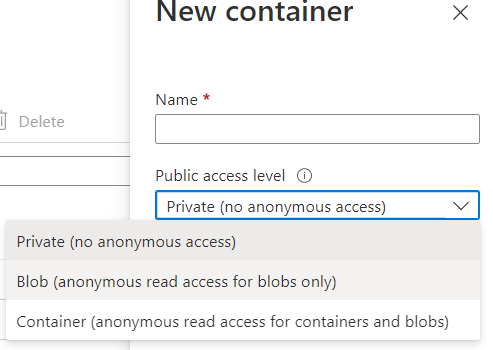
* Налог за складиштење(*енгл. storage account*) је ресурс *Azure Resource Manager*-а. Сваки ресурс менаџера ресурса, укључујући *Azure storage account*, мора припадати *Azure resource group*. Група ресурса је логички контејнер за груписање *Azure services*. Када се креира налог за складиштење (Слика 4.8), има опција да се креира нова *resource group* или да користите постојећу *resource group.*



Слика 4.8 -Storage account

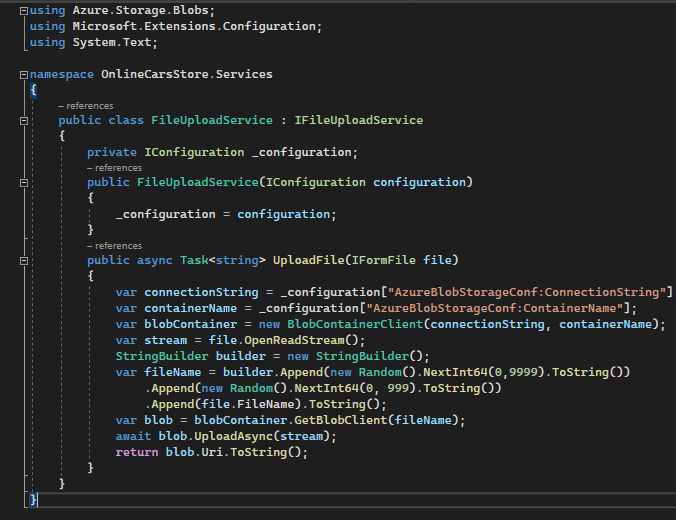
* Корисници разматраног апликативног рјешења имају могућност додавања фајлова било ког типа у оквиру креираног задатка. Дате датотеке се могу касније преузети. *Azure Blob storage* је коришћено за имплементацију поменуте функционалности у оквиру апликације. Апликациони програм делегира цијели процес складиштења датотека на услугу *Azure Blob storage*. Успјешно сачуваним датотекама се додјељује адреса преко које им се може приступити. Адреса датотеке, заједно са примарним идентификатором задатка за који је креирана, чува се у оквиру одговарајуће табеле у *Azure SQL relational database*.

* *Azure Container Instances* (Слика 4.9) је услуга која омогућава програмеру да примјени контејнере у *Microsoft Azure public cloud* без потребе да обезбјеђује или управља било којом основном инфраструктуром. Бира се *Blob* да буде свима доступно.На *Container-*у се чувају слике.



Слика 4.9 -Azure Container Instances

* Конекциони стринг за приступ сервису за складиштење података доступан је у секцији *Access keys*. Конекциони стринг и назив контејнера чувају се у оквиру фајла *appsettings.json*.
* Програмски код за рад са сервисом Azure Blob storage service је приказан на Слици 4.10



Cлика 4.10 -Програмски код за рад са сервисом Azure Blob storage service

# ИМПЛЕМЕНТАЦИЈА ФРОНТЕНДА

Идеја и циљ ове апликације јесте да процес е-трговине односно продају аутомобила учини што једноставнијом. Приступ овој локацији је омогућен са било које локације.

У функционалном смислу, ова апликација је подијељена на два дијела. Први дио ове апликације доступан је гостима (посјетиоцима веб странице) и корисницима, док је други дио апликације доступан само администраторима. Приступи корисничком дијелу апликације, односно администраторском, као и дозвољене акције у апликацији су одређене ауторизационим правилима дефинисаним у контролерима.

## Корисничка хијерархија

Посматрајући корисничку хијерархију ове апликације, можемо да издвојимо три различите улоге корисника:

* **Администратор** – Има приступ администраторском панелу, могућност прегледа регистрованих корисника, као и њиховим управљањем и брисањем, могућност креирања и брисања одређених брендова и модела аутомобила, одлоговање са система.
* **Регистровани корисник** – Преглед измјене података профила, могућност замјене своје шифре на систему, преглед, измјена и брисање аутомобила које је тај регистовани корисник креирао, одлоговање са система.
* **Гост** – Нерегистровани корисник који може да прегледа садржај веб странице и да се информише о аутомобилима, да врши претрагу одређених аутомобила по бренду или моделу. Има право регистрације на сајт и могућност логовања на веб страницу

## Структура веб апликације

Структурно ову апликацију можемо подијелити на три цјелине, а то су:

* Заглавље
* Главни садржај странице
* Подножје

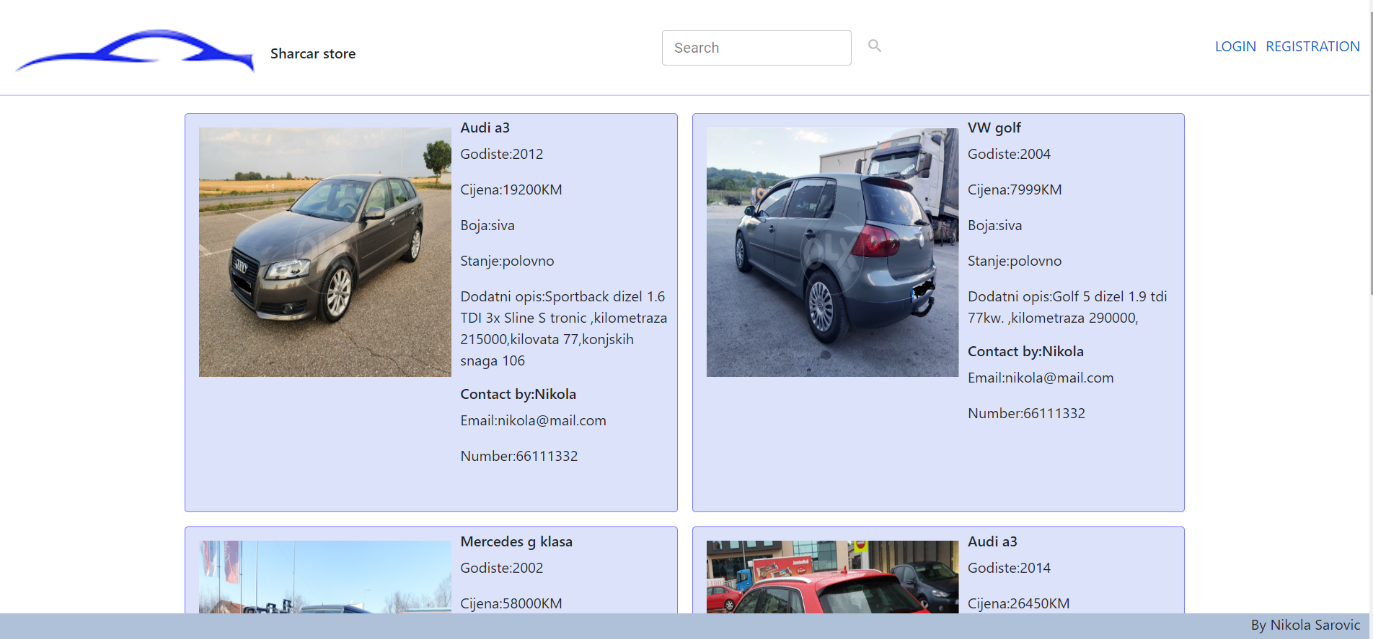
Навигациони мени који се налази у заглављу служи за навигацију између страница и његов садржај зависи од типа корисника (администратор и логовани корисник додатне опције). У десном дијелу навигацијског садржаја су линкови који омогућавају кориснику пријаву на систем или регистровање,а у средини налази се могућност претраге.

Главни садржај странице мијења се при сваком позиву неке од могућих акција. Заузима највећи простор ове веб апликације. И њен фокус је у већини случајева рад са аутомобилима

У футеру су приказане информације о веб страници.

## Приступ систему

Приликом приступа веб страници, кориснику ће се приказати страница са Слике 5.1 и Слике 5.2. Страница се састоји од навигационог менија у заглавља, радног простора и подножја са информацијама. Пагинација обиљежава стране аутомобила за приказ. На радном простору приказани су аутомобили који су на продаји чији су власници регистровани корисници.



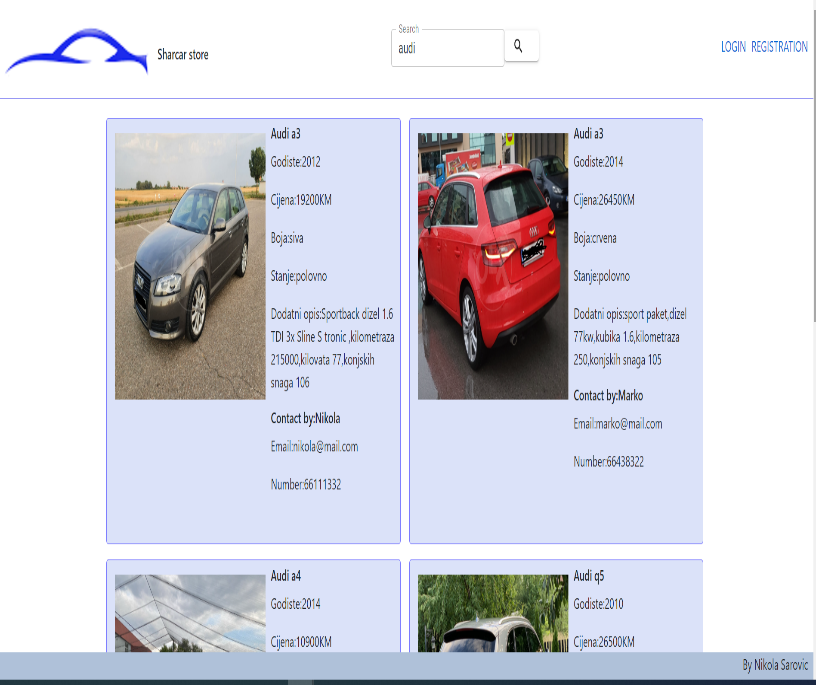
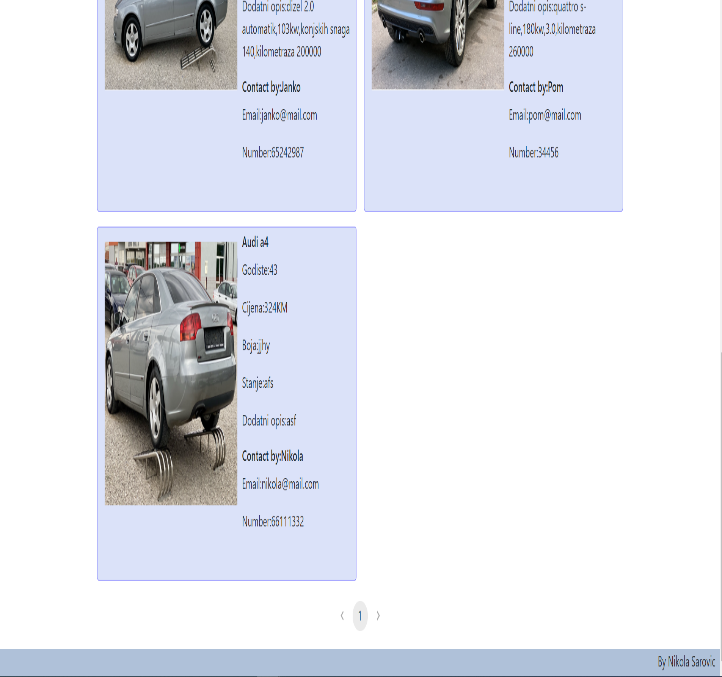
Слика 5.1 -Почетна страница са заглављем и главним дијелом

****

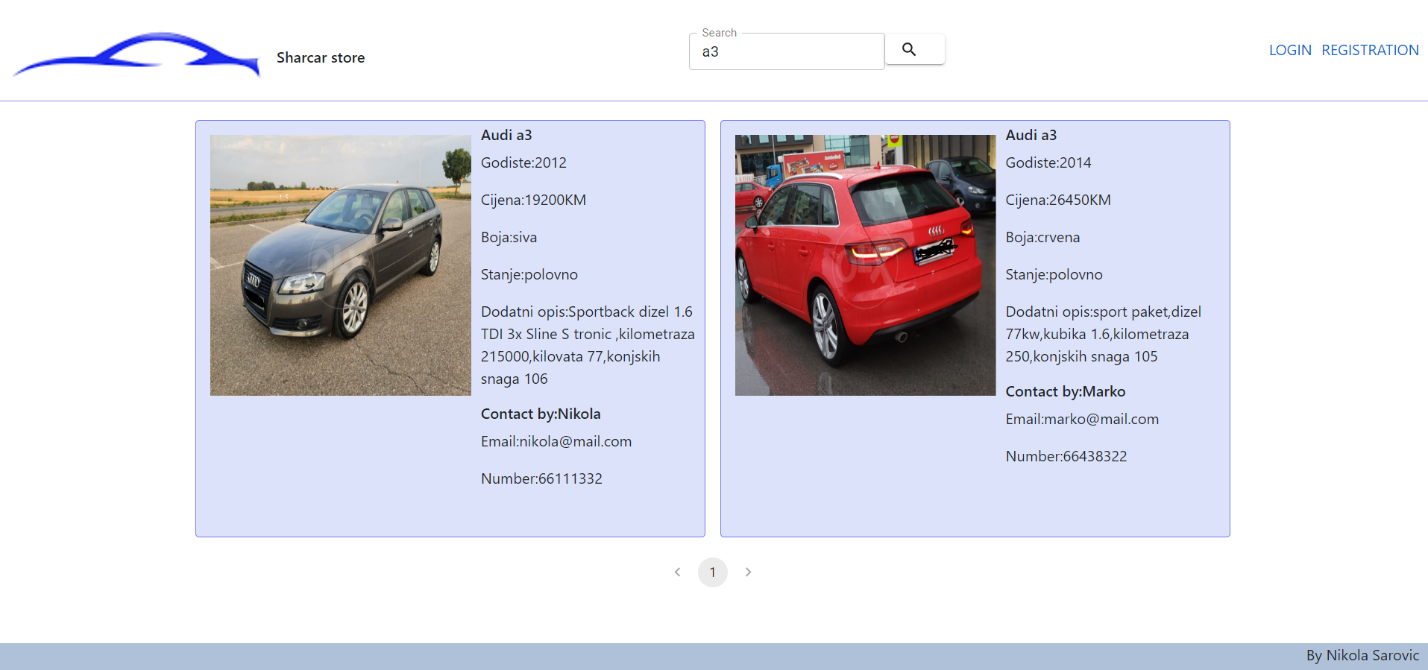
Слика 5.2 -Крај странице

## Претрага аутомобила

Омугућена је претрага аутомобила да корисник лако дође до одређеног аутомобила који га интересује. Претрага је омугућена по бренду (Слика 5.3) или моделу (Слика 5.4) аутомобила.



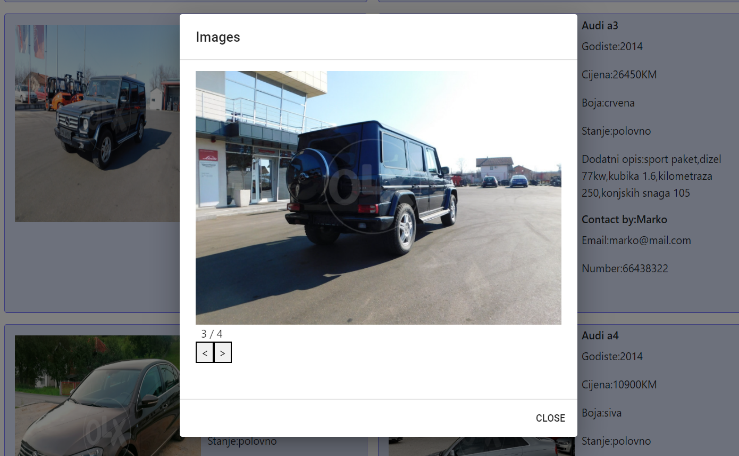
Cлика 5.3 -Претрага по бренду аутомобила



Слика 5.4 -Претрага по моделу аутомобила

## Приказ аутомобила

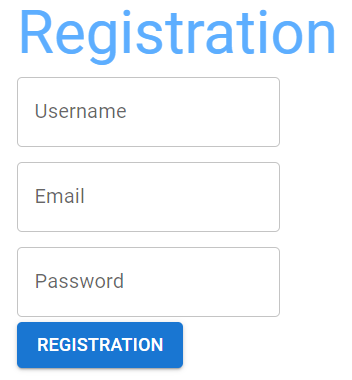
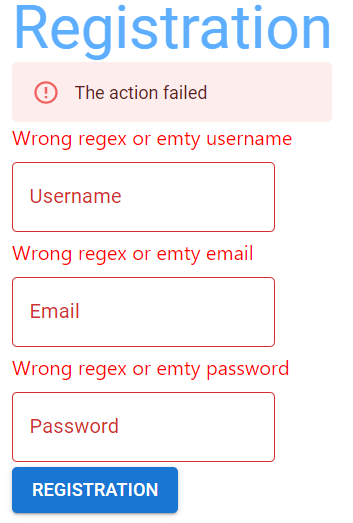
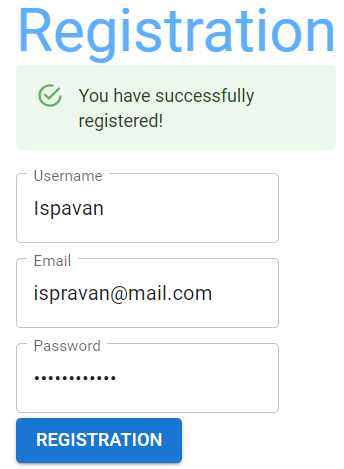
Корисницима је омогућено да виде информације о одређеном аутомобилу који је на продаји, такође и листање слика тог аутомобила за који је корисник заинтересован (Слика 5.5).



Слика 5.5 -Приказ аутомобила

## Регистрација корисника

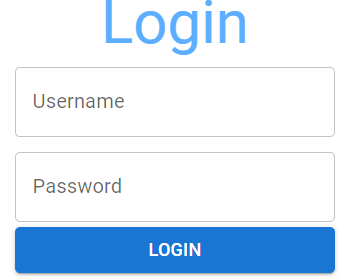
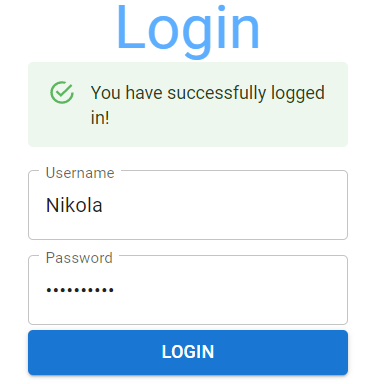
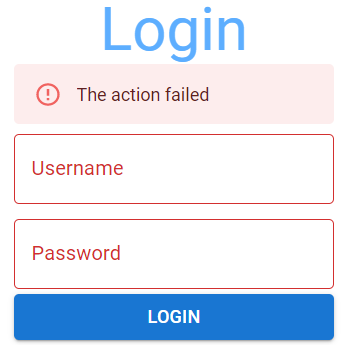
Да би посјетилац веб странице имао могућност креирања својих аутомобиа потребно је да се региструје. Кликом на дугме *Register* у навигационом менију (Слика 5.6), отвара се страница са формом за регистровање (Слика 50) коју гост треба да попуни. Потребно је повести рачуна да подаци који се уносе буду валидни. Уколико је унијето име или адреса заузета, гост ће бити обавијештен о томе. Процес регистрације неће бити могуће извршити све док се не унесу валидни подаци.



Слика 5.6 -Форма за регистрацију

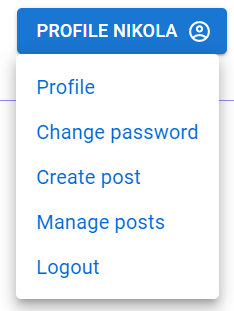
## Пријава на систем

Потребно је у навигационом менију кликнути на дугме *Login*, након чега нам се приказује страница приказана на Слици 5.7. Уколико су унијети подаци неисправни, корисник се обавјештава о томе. Али уколико су унијети подаци исправни корисник је успјешно пријављен на систем. Умјесто дугмади за пријаву и регистровање има могућност отварања навигацијонг менија, гдје има могућност одјављивања на дугме *Logout* Слика 5.8.



Слика 5.7 -Форма за пријаву на систем

На Слици 5.8 је приказ навигационог менија:



Слика 5.8 -Навигациони мени

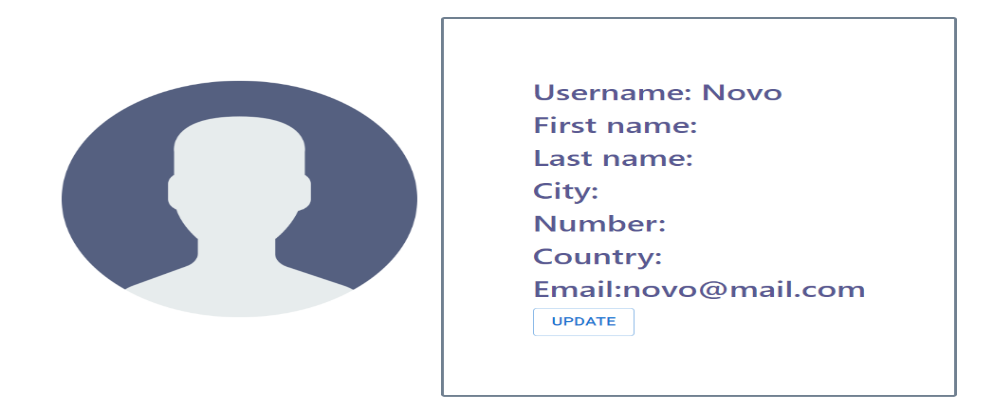
### Кориснички профил

Регистровани корисник има **прелгед** свог профила (Слика 5.9), гдје су дати основни подаци о том кориснику.



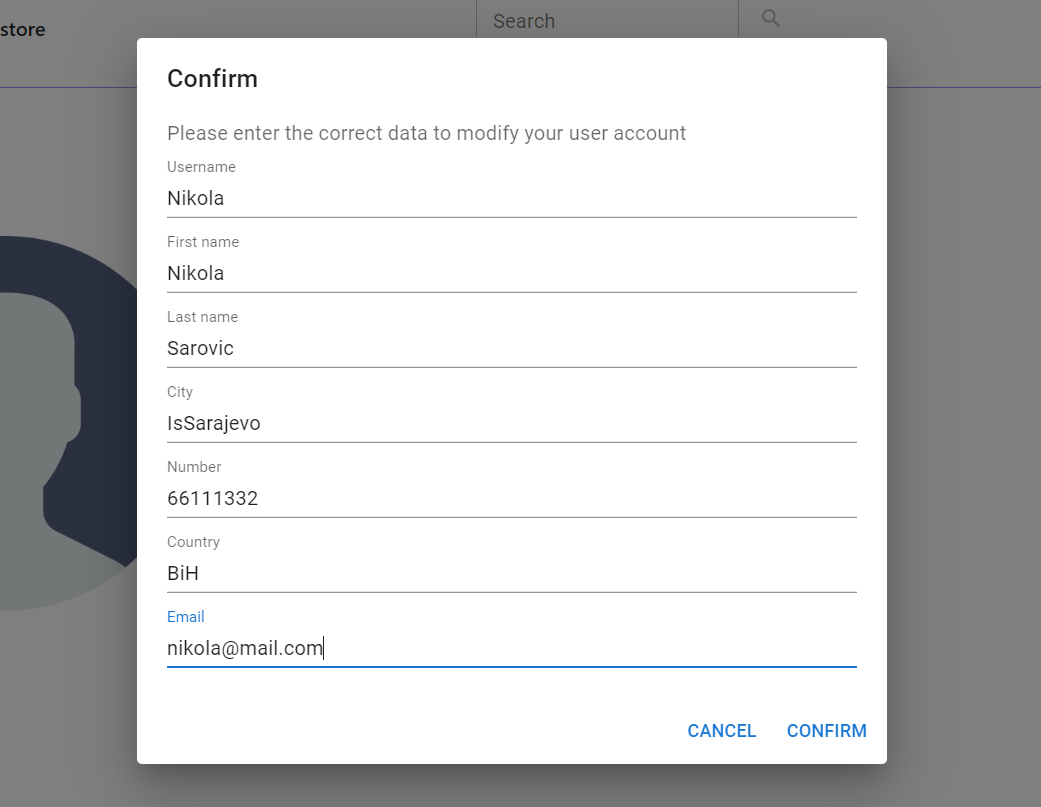
Слика 5.9 -Преглед података о профилу

Корисник који се тек региструје на систем и приликом првог приступа систему и уласка на свој кориснички профил, сви подаци осим *Username*-a i *Email-*a ће бити празни (Слика 5.10). Тако да остаје могућност на регистрованом кориснику да **дода** још неке податке о себи.



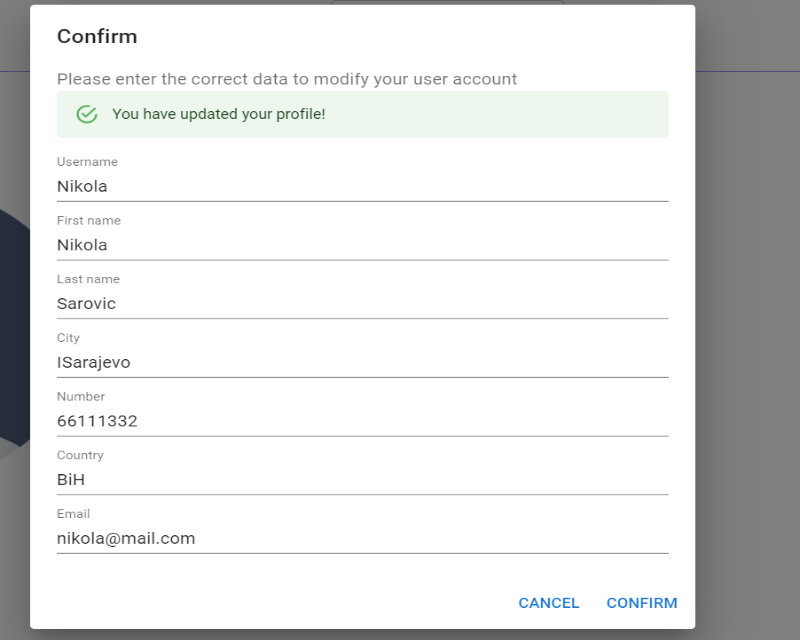
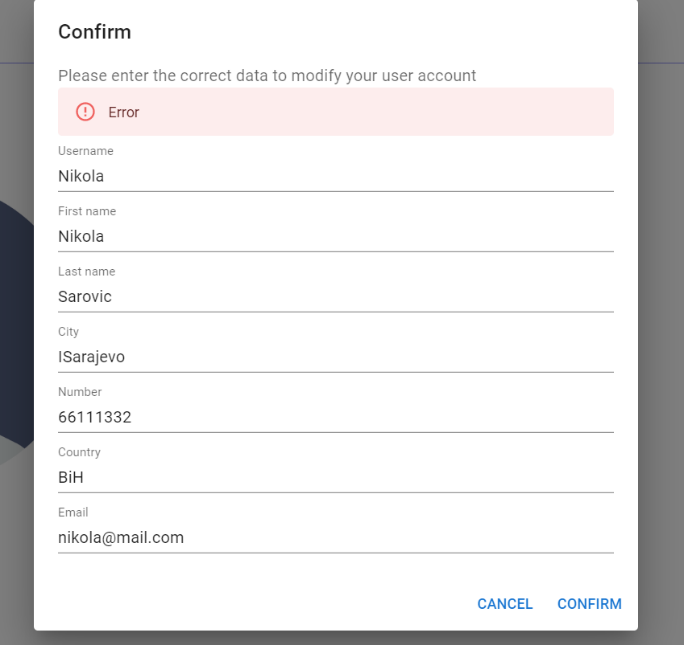
Слика 5.10 -Преглед података о креираном профилу

Такође има увијек могућност **измјене** одређених података о себи (Слика 5.11).



Слика 5.11 -Додавање или измјена података о профилу

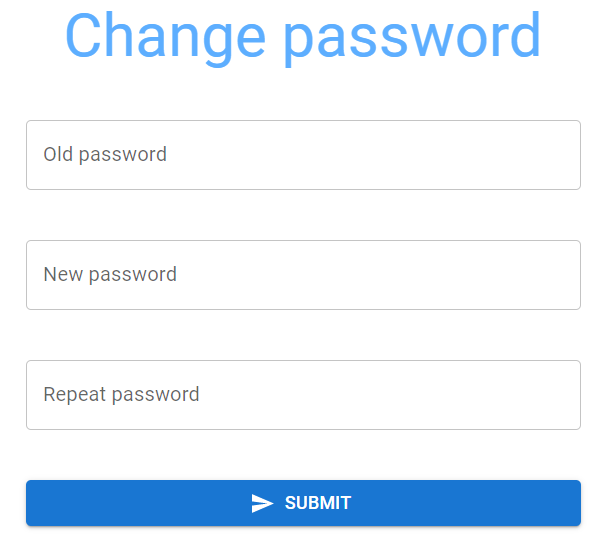
Ако су подаци исправно унесени, тада ће се ажурирати профил са одређеним измјенама или допунама.У супротном биће порука да је грешка као што је приказано на Слици 5.12.



Слика 5.12 -Приказ поруке о измјенама на профилу

### Измјена шифре

Регистровани корисник има могућност да промјени своју шифру (Слика 5.13). Потребно је повести рачуна да подаци који се уносе буду валидни, односно да је стара лозинка исправна и да се нове лозинке поклапају. Процес промјене шифре неће бити могуће извршити све док се не унесу валидни подаци.

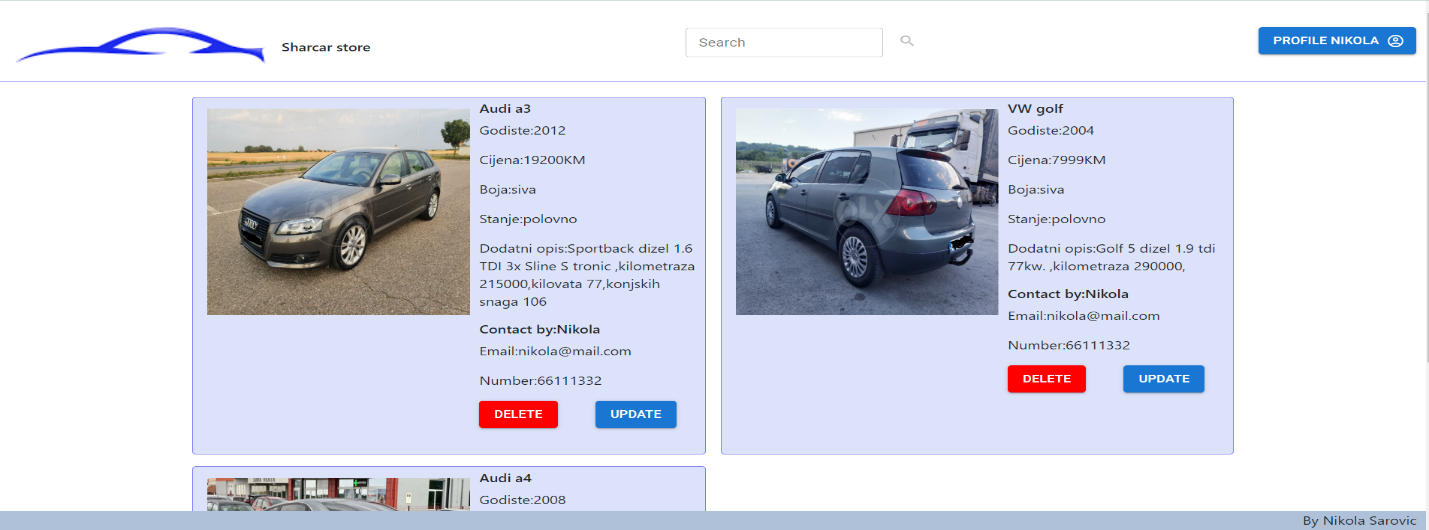


Слика 5.13 -Форма за измјену шифре

### Управљање објавама

Кликом на *manage posts* на навигационом менију регистровани корисник има могућност **прегледа** свих својих објава, то јесте аутомобила које је тај корисник креирао и који су на продају изложени осталим корисницима .

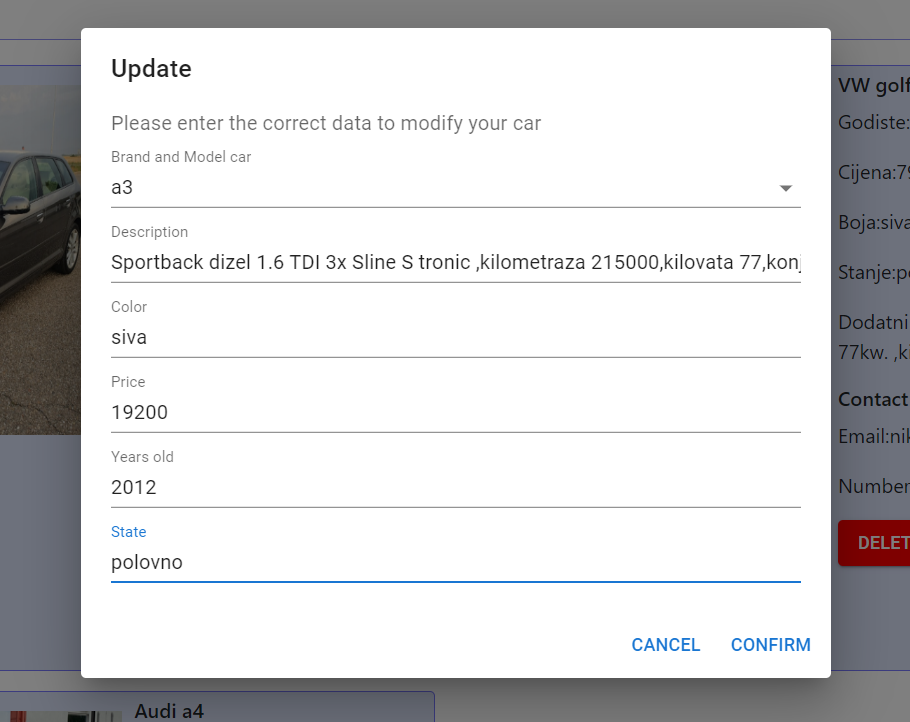
Преглед аутомобила је дат на Слици 5.14:



Слика 5.14 -Преглед креираних аутомобила од корисника

Регистровани корисник осим прегледа има могућност да **измјени** одређене податке о креираном аутомобилу, у случају да их је погрешно уносио током креирања аутомобила.

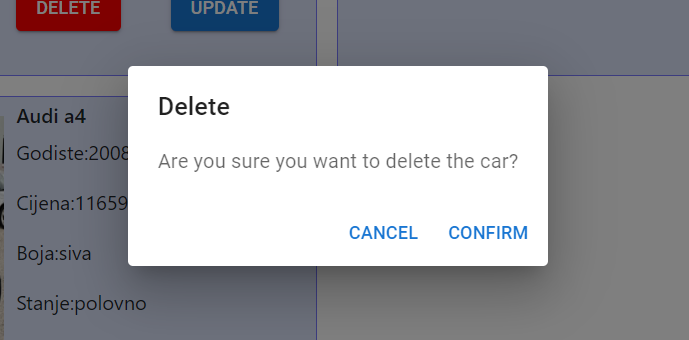
Диалог за измјену података одређеног креираног аутомобила дат је на Слици 5.15:



Слика 5.15 -Измјена података креираног аутомобила

И на крају регистровани корисник има могућност **брисања** одређеног креираног аутомобила, у случају да је договорио продају са неким корисником или да је одустао од продаје тог аутомобила.

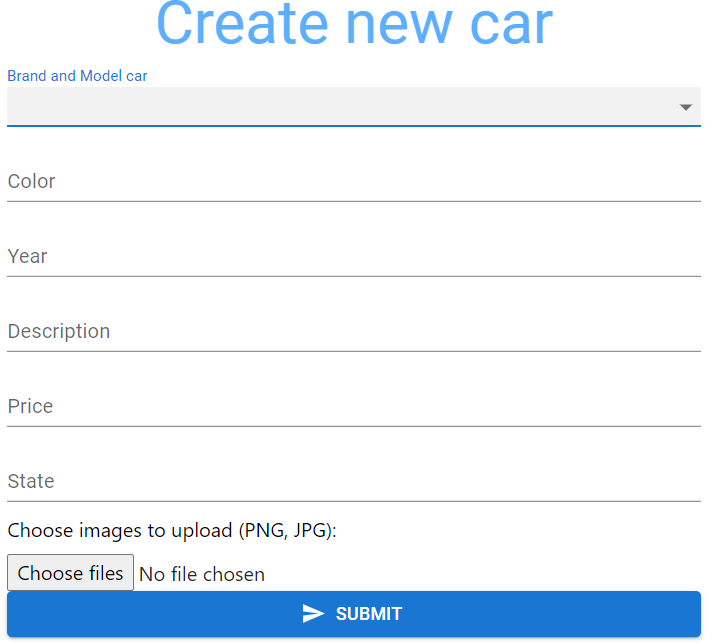
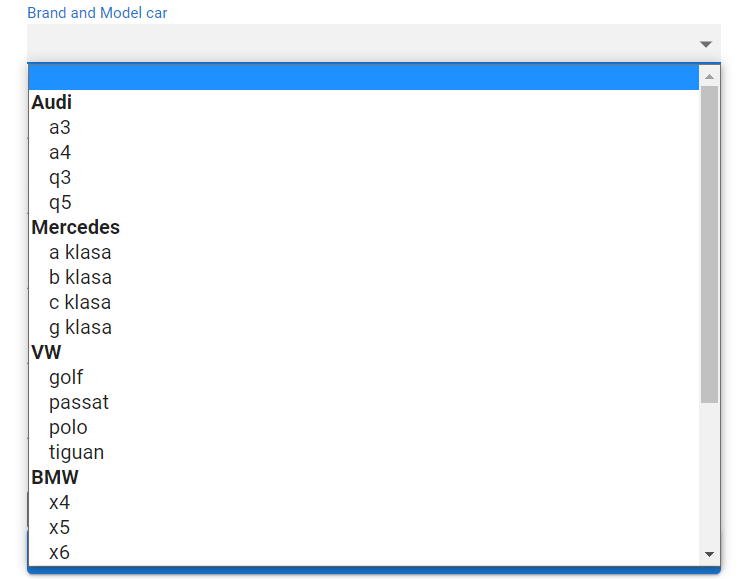
Диалог за брисање одређеног креираног аутомобила дат је на Слици 5.16:



Слика 5.16 -Брисање одређеног креираног аутомобила

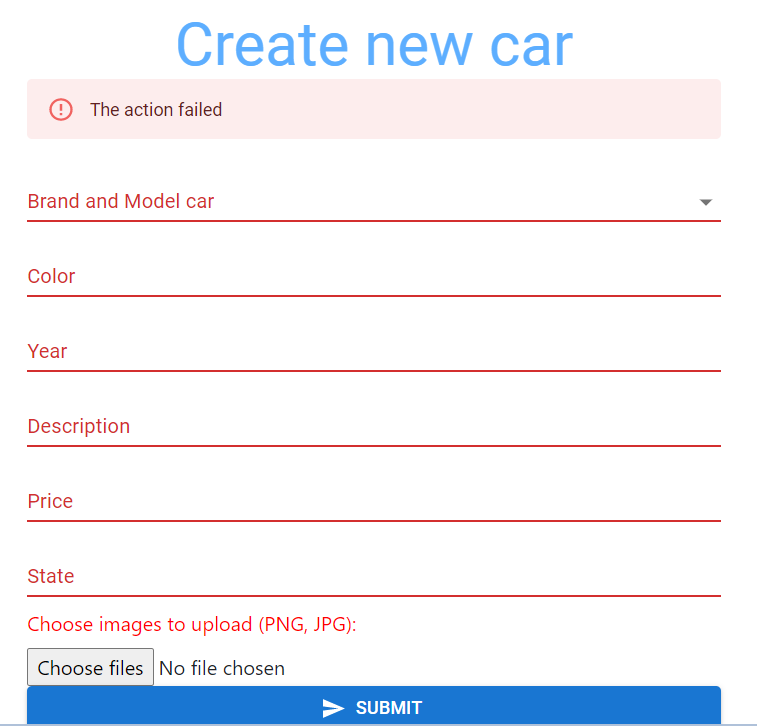
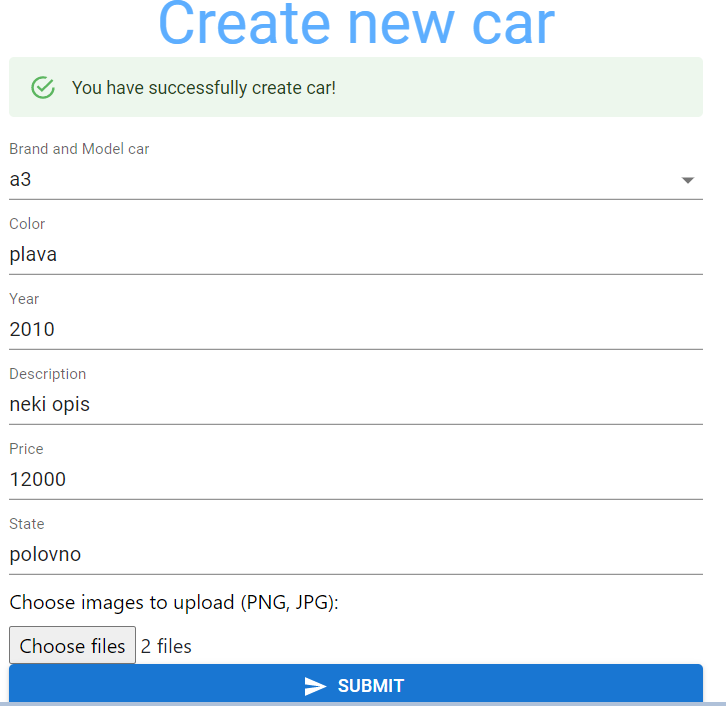
### Креирање аутомобила

Сваки регистровани корисник креирањем аутомобила објављује податаке и слике о том аутомобилу којег жели да прода (Слика 5.17). Регистованом кориснику је омогућено да преко овог сајта нађе одређеног купца који је заинтересован за тај одређени аутомобил.



Слика 5.17 -Креирање аутомобила

Ако регистровани корисник не унесе све податке и слике, креирање аутомобила ће бити неуспјешно и приказаће се порука о несупјешној акцији креирања аутомобила, у супротном аутомобил ће се успјешно креирати и приказаће се порука о успјешној акцији креирања аутомобила као што је приказано на Слици 5.18.



Слика 5.18 -Поруке о креирању аутомобила

# ЗАКЉУЧАК

Циљ ове апликације је да омогући корисницима да на оптималан начин могу да купе или продају свој аутомобил. Предност ове веб странице је што јој можемо приступити са било које локације и на тај начин можемо претражити већ понуђени каталог аутомобила који су на продаји, а исто тако можемо објавити свој аутомобил, један или више њих које желимо да продамо .У поређењу са другим страницама, сличног типа садржаја, предност се огледа у томе што на страници нема велики број реклама, али и не захтјева од корисника велики број кликова да би добио жељене информације. Такође, од корисника се не тражи плаћање претплате на страницу.

Наведени резултати су остварени искључиво уз помоћ технологија које смо користили при изради ове апликације. Дате технологије су омогућиле брзу и релативно оптималну реализацију завршног рада. Посебно се издваја *EF Core* чије окружење је довело до много простије реализације валидације унешених података, једноставно извршавање миграција, веома једноставан начин за формирање комплексних упита при комуникацији са базом података. Док на клијентској страни издаваја се *Redux*  помоћу *Redux*-а стање апликације се чува у *store-*у , а *store-*у можемо приступити из свих компоненти у апликацији. ASP.NET Core, React и Тypescript су технологије које су такође подједнако допринијеле приликом реализације ове апликације. Уз помоћ интернета на коме је доступан велики број страница са едукативним садржајем о претходно наведеним технологијама израда је била знатно олакшана и убрзана.

Приказано апликативно рјешење могуће је надоградити тако да се омогући извршавање на *Android* и *iOS* оперативним системима. Поменуто унапријеђење подразумијева кориштење постојећег *Web Api-*ја и креирање корисничког интерфејса. Уколило би се апликација показала корисном, садржај би се могао проширити и на остала моторна возила.

# ЛИТЕРАТУРА

1 Ивана Стојановић, МР - Електронска трговина и куповина путем Интернета у Србији, 2010.Београд

2 Веб апликације

<https://programirajmozajedno.wordpress.com/2015/01/09/kratak-uvod-u-world-wide-web-i-web-aplikacije>, Приступљено 12.05.2022. године.

3 Single page application

<https://www.monocubed.com/blog/what-is-single-page-application>,Приступљено 25.05.2022. године

4 Документација о *Microsoft SQL Server*

[*https*://en.wikipedia.org/wiki/*Microsoft*\_SQL\_Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server),Приступљено 02.06.2022. године

5 *ASP.NET* Core in Action [Andrew Lock](https://sr.b-ok.xyz/author/Andrew%20Lock)

6 Web API

<https://www.tutorialsteacher.com/webapi/what-is-web-api>,Приступљено 05.06.2022. године

7 Entity Framework

<https://www.entityframeworktutorial.net>, Приступљено 13.06.2022. године

8 *Microsoft* Azure

<https://www.simplilearn.com/tutorials/azure-tutorial/what-is-azure>,Приступљено 17.06.2022. године

9 Увод у веб програмирање *HTML*, *CSS* и *JavaScript* Данијел Мијић

10 *TypeScript*

[https://radixweb.com/blog/*TypeScript*-vs-*JavaScript*](https://radixweb.com/blog/typescript-vs-javascript)*,* Приступљено 22.06.2022. године

11 *React Redux*

<https://react-redux.js.org/>, Приступљено 27.06.2022. године