Универзитет у Београду

Електротехнички факултет

Ангулар 5

Дипломски рад

Ментор:

проф. др Бошко Николић

Студент:

Јелена Живковић

Београд, Септембар 2018.

Садржај

1 Увод

Повећана употреба динамичког садржаја у веб апликацијама довела је до потребе за развојем нових софтверских решења како би се омогућиле боље перформансе. Како би задовољили све веће потребе тржишта, тим програмера из америчке компаније Гоогле (енг. Google), долази на идеју да развије оквир који би омогућио лако тестирање и одржавање динамичких фронт-енд апликација. Тако настаје Ангулар JC оквир, чија прва верзија бива издата 2009. године. Суфикс “JC” указује на то да се ради о ЈавaСкрипт (енг. JavaScript) оквиру за развој клијентских веб апликација.

АнгуларJC оквир ради тако што прво прочита ХТМЛ страну која има уграђене нестандардне таг атрибуте. Ангулар те атрибуте интерпретира као директиве како би повезао улазне и излазне делове странице за модел који је представљен стандардним ЈавaСкрипт променљивама. Вредности тих ЈавaСцрипт променљивих се могу ручно подесити у коду или могу бити преузете од статичких или динамичких JCOH извора. Овакав приступ је у многоме задовољавао потребе које су постојале на тадашњим веб апликацијама. Међутим, с развојем Андроид и Иос оперативних система ситуација почиње да се мења и недостаци у погледу перформанси и скалабилности постојећег АнгуларJC решења постају све видљивији.

2016. Године објављена је друга верзија Ангулар оквира, верзија Ангулар 2. АнгуларJC је у потпуности редизајниран на основу савета из праксе, али тако да буде у кораку са технологијама које су се појавиле након прве верзије. Нова верзија је након издавања бета верзије названа само “Ангулар”. Избацивши “JC” из имена, Гоогле указује на то да језик на ком је базирана нова Ангулар платформа више није ЈавaСкрипт већ ТајпСкрипт (енг. TypeScript). Базирање платформе на реактивном језику као што је ТајпСкрипт омогућило је подршку реактивном програмирању, као и динамичко учитавање компоненти. Велика предност је и то што веб апликација бива у потпуности састављена од директива и компоненти, као и могућност коришћења свих карактеристика других ЈавaСкрипт библиотека. Поред тога, унапређена је и комуникација са

базом и ВебСокет-има (енг. WebSockets). Нова верзија уз развој веб апликација, омогућава и развој мобилних апликација и тиме Ангулар постаје универзална платформа.

Основна намена Ангулар платформе је развој једностраничних веб апликација. Уз развој једностраничних веб апликација, могућ је и развој вишестраничних апликација коришћењем Ангулар Универсал библиотеке. Како би Ангулар омогућио развој мобилних апликација уведен је Ионик оквир који је такође развијен од стране Гоогле-а. Уз Ионик долазе елементи који се смештају на стек и чине мобилну апликацију, али и низ интерфејса за комуникацију са компонентама унутар мобилног уређаја, као што су камера и ГПС сензор.

Добра својства прве верзије АнгуларЈС оквира, попут директива и убризгавања зависности (енг. Dependency injection), наслеђена су и побољшана унутар Ангулар платформе. Поред тога, побољшане су перформансе апликације и избачени су неки старији механизми ради лакшег сналажења при раду са платформом. Унутар Ангулар платформе, сваки ентитет се дефининише помоћу ТупеСкрипт класе чиме је омогућено једноставније писање и лакше сналажење унутар самог кода. Уз то, Ангулар платформа долази са командним алатом Ангулар-ЦЛИ који у многоме олакшава рад са самом платформом што убрзава развој апликације.

2 Једностраничне апликације

Једностранична апликација (енгл *Single-page application (SPA)*), такође позната као једностранични интерфејс (енгл. *Single-page interface (SPI)*), је веб-апликација на којој се интеракција са корисником обавља тако што се динамички учитавају делови страна како би се избегло поновно учитавање целе стране са сервера. Овакав приступ омогућује лакшу интеракцију са корисником и чини да се апликација понаша више као обична стона(desktop) апликација.

Сав код у једностраничним апликацијама се преузима у оквиру учитавања једне странице, или се одговарајући ресурси учитавају динамички и по потреби, обично као последица неке корисничке акције (клик мишом, клик на тастатури итд.). Страница се не учитава поновно ни у једној тачки процеса, нити се пребацује на неку другу страницу. Међутим, новије веб-технологије (које су укључене у HTML5) пружају могућност перцепције и навигације кроз више логичких страница у оквиру исте апликације. Интеракција са једностраном апликацијом често укључује динамичку комуникацију са веб-сервером у позадини.

JavaScript језик може да се користи у веб претраживачу за приказивање корисничког интерфејса (енг. User Interface), покретање апликационе логике и за комуникацију са веб сервером. Међутим, потреба програмера за писањем JavaScript кода није тако велика с обзитром да постоји велики број бесплатних библиотека за развој једностраничних апликација.

Технике које омогућавају веб прегледачу да задржи једну страну чак и када апликација захтева комуникацију са веб сервером су:

1. JavaScript оквири (енг. JavaScript frameworks)
    a. Ангулар (Angular)
    b. Реакт (React)
    c. Метеор (Meteor.js)

d. Ембер (Ember.js)
e. Аурелиа (Aurellia)
f. Ву (Vue.js)

2. Ајах
3. Веб прикључнице (енг. Websockets)
4. Догађаји послати од сервера (енг. Server Sent Events)
5. Додаци претраживача (енг. Browser plugins)
6. Транспорт података (енг. Data transport)
7. Архитектура сервера (енг. Server architecture)
    g. Архитектура лаких сервера (Thin server architecture)
    h. Архитектура тешких сервера без меморије (Thick stateless server architecture)
    i. Архитектура тешких сервера са меморијом (Thick stateful server architecture)

2.1 Једностраничне апликације у Ангулар-у

Ангулар је у потпуности клијентски оријентисан оквир. Његови шаблони засновани су на двосмерном UI data binding-у (повезивање података на корисничком интерфејсу). Повезивање података је аутоматски начин освежавања прегледа (енг. view) кад год се модел промени, као и обрнуто. Хтмл шаблон се компајлира (преводи) у претраживачу. Превођењем се креира чист ХТМЛ, који претраживач касније интерпретира у преглед који види корисник. Овај корак се понавља за узајамне прегледе странице. Традиционално, серверски оријентисано ХТМЛ програмирање захтевало је међусобну интеракцију контролера и модела са сервером. Међутим, Ангулар оквир омогућује да се стања контролера и модела одржавају и освежавају унутар клијентског претраживача. Стога, нове странице могу да се генеришу без икакве интеракције са сервером.

3 Тајпскрипт (енг. TypeScript)

Тајпскрипт (енг. Typescript) је бесплатан програмски језик отвореног кода (енг. open source language), који развија и одржава Мајкрософт. Строг је надскуп Јаваскрипта, и додаје језику опциону статичку типизацију и објектну оријентисаност. Тајпскрипт се може користити за развој Јаваскрипт апликација за извршавање на клијенту или серверу (Node.js). Дизајниран је за развој великих апликација и компајлира се у Јаваскрипт.[5] Како је надскуп Јаваскрипта, сви постојећи Јаваскрипт програми су такође и валидни Тајпскрипт програми. Подржава хедер фајлове који могу да садрже типовске информације за постојеће Јаваскрипт библиотеке, омогућавајући тиме другим програмима да користе објекте дефинисане у хедер фајловима као да су снажно типизирани Тајпскрипт објекти. Тајпскрипт компајлер је и сам написан у Јаваскрипту и лиценциран под Apache 2 лиценцом.

Тајпскрипт је укључен као језик прве класе у Microsoft Visual Studio 2013 (Update 2) развојном окружењу и новијим, уз C# и остале Мајкрософтове језике.[6] Официјална екстензија омогућава рад у Тајпскрипту и у Visual Studio 2012.

Тајпскрипт је настао због перципираних недостатака Јаваскрипта за развој великих апликација од стране Мајкрософта и њихових клијената.[8] Изазови са комплексним Јаваскрипт кодом довели су до потражње за прилагођеним алатима за би се олакшао развој компоненти у језику.[9]

Дизајнери Тајпскрипта тражили су решење које неће изгубити компатибилност са стандардом и његовом вишеплатформском подршком. Знајући за тренутни предлог за ECMAScript стандард који је обећавао подршку за класе у будућности, Тајпскрипт су засновали по том предлогу. То је довело до Јаваскрипт компајлера са скупом синтаксичких језичких проширења, надскупом заснованим на предлогу, који претвара проширења у обичан Јаваскрипт

Тајпскрипт је језичко проширење које додаје могућности ECMAScript 5. Додатне могућности су:

1. Типовски потписи и провера типова за време компилације

2. Дедукција типа података
3. Класе
4. Интерфејси
5. Бројачки тип
6. Mixin
7. Генерици
8. Модуларно програмирање[10]
9. Скраћена "стрелица" синтакса за анонимне функције
10. Опциони параметри и подразумевани параметри
11. Tuple

4 Бутстреп (фронт-енд фрејмворк)

Бутстреп (енгл. Bootstrap) представља бесплатни веб фрејмворк отвореног кода, за креирање веб сајтова и веб апликација. Базиран је на HTML и CSSшаблонима за типографију, креирању формулара, дугмади, навигационим и осталим компонентама интерфејса, као и опционим ЈавaСкрипт додацима. Циљ Бутстреп фрејмворка (енгл. framework) је олакшавање програмирања за веб.

Бутстреп је модуларан и састоји се од низа Less стилова који дефинишу различите компоненте скупа алата. Ови стилови су најчешће садржани у пакету и убачени у веб-страницу, док се појединачне компоненте могу убацити или не. Бутстреп обезбеђује велики број конфигурационих променљивих помоћу којих се контролишу ствари као што су боја или растојање садржаја од ивица код различитих компоненти.

Од верзије 2.0, Бутстреп документација садржи посебан чаробњак за подешавање. Програмери могу да изаберу компоненте које желе да користе и, ако је потребно, подешавају вредности одређених опција према својим потребама. Од верзије Бутстреп 4, користи се Sass стилски језик уместо Less-а.

Свака Бутстреп компонента се састоји од HTML структуре, CSS декларација и у неким случајевима ЈавaСкрипт кода. Бутстреп обезбеђује скуп стилова који пружају основне дефиниције за све кључне HTML компоненте. Тиме се добија униформан, модеран приказ текста, табела и формулара.

Поред основних HTML елемената, Бутстреп садржи и неке често коришћене елементе за формирање интерфејса. Они укључују дугмад са посебним функцијама (на пример груписање дугмади или дугмад са drop-down опцијом, навигационе листе, хоризонтални и вертикални табови, итд.), ознаке, напредне типографске могућности, поруке са упозорењима, прогресивне траке, и др. Компоненте су имплементиране као CSS класе, које морају бити везане за одређене елементе на веб страни.

Бутстреп у основи садржи неколико ЈавaСкрипт компоненти у облику ЏејКвери додатака. Они пружају додатне елементе за кориснички интерфејс као што су дијалог боксови, описи алатки (енгл.tooltips) и каруселi (енгл. carousel).

Они такође проширују функционалност неколико постојећих елемената, укључујући, на пример, функцију за аутоматско попуњавање поља за унос. У верзији 2.0, подржани су следећи ЈавaСкрипт додаци: Modal, Dropdown, Scrollspy, Tab, Tooltip, Popover, Alert, Button, Collapse, Carousel и Typeahead.Бутстреп 3 је подржан у најновијим верзијама Гугл Хрома, Фајерфокса, Интернет Експлорера, Опере и Сафарија (осим на Виндоусу). Додатно је подржан и у Интернет Експлореру 8 и најновијем Фајерфокс издању са додатном подршком (енгл. Firefox Extended Support Release - ESR).[11]

Од верзије 2.0 Бутстреп подржава прилагодљив веб дизајн (енгл. resposnive web design). Ово значи да се изглед веб стране динамички прилагођава, узимајући у обзир тип уређаја који се користи (десктоп рачунар, таблет, мобилни телефон).

Почевши од верзије 3.0, Бутстреп је усвојио принцип дизајна "мобилни уређаји пре свега", наглашавајући прилагодљив веб дизајн као подразумевану опцију.

Алфа верзија Бутстрепа 4 додала је подршку за Sass и CSS flexbox.

Сав кôд везан за Бутстреп доступан је и бесплатан на ГитХабу. Програмери се подстичу да учествују у овом пројекту и да сами допринесу његовом развоју.

5 Ангулар

5.1 Историјат и раније верзије

Ангулар JC:

Овај оквир представља зачетак Ангулар платформе али уједно и прву верзију Ангулара. Касније верзије Ангулара засноване су на овој, али су разлике ипак превелике па се ова верзија посматра одвојено од осталих. За разлику од осталих Ангулар верзија, ова верзија нема свој језик и апликациону структуру па због тога она представља оквир, а не платформу. Развој је започет 2009. године али је прва званична верзија оквира издата 2010. Једна од битнијих верзија овог оквира је верзија 1.6 јер је у тој верзији представљен концепт архитектуре

Оригинално, преписка Ангулар JC оквира, названа је Ангулар 2 од стране развојног тима. Међутим, ово је довело до забуне, па је тим најавио промену у имену оквира. Наиме, они су донели одлуку да се назив “Ангулар JC” односи на 1.X верзије, док се “Ангулар”, без суфикса “JC”, односи на верзије почев од верзије 2.

Ангулар 2.0

Ангулар 2.0 најављен је на нг-Европа() конференцији 22-23. Октобра 2014. Драстичне промене које је ова верзија унела изазвале су велике контраверзије између програмера. Први кандидат за издавање је представљен у Мају 2016. године. Званична верзија издата је 14.Септембра 2016. Почев од ове верзије, Ангулар постаје платформа и почиње да се заснива на Тајпскрипт(TypeScript) језику због чега се и избацује суфикс “JC” из имена.

Ангулар 4.0

Децембра 2016. Године најављена је нова верзија Ангулар платформе, верзија 4. Верзија 3 је прескочена као би се избегла конфузија због неусаглашености верзије пакета рутера која је већ дистрибуирана као в3.3.0. Финална верзија Ангулар 4 објављена је у 23. Марта 2017. године. Ангулар 4 верзија је компатибилна уназад (енг. Backward compatible[1]) са Ангулар 2 верзијом.

[1] Backward compatible - може да се користи са старијим верзијама хардвера и софтвера без икаквог прилагођавања и без икаквих измена

Својства (енг. features) у верзији 4:

1. Хттп клијент (енг. HttpClient), мања, лакша за коришћење и доста јача библиотека за прављење Хттп захтева [2](енг. HttpRequest)
2. Нови за догађаји везани за животни циклус рутера (енг. Router life cycle events) за чуваре (енг. Guards) и ресолвере (енг. Resolvers). Додата су четири нова догађаја и то су : GuardsCheckStart, GuardsCheckEnd, ResolveStart и ResolveEnd.
3. Условно искључивање анимација

Ангулар 5.0

Ова верзија је објављена 1.Новембра 2017.године. Кључна побољшања су подршка за прогресивне веб апликације, оптимизатор процеса генерисања кода и побољшања везана за материјални дизајн (енг.Material Design)

Ангулар 6.0

Ова верзија је објављена 4.Маја 2018.године. Ово је велико издање фокусирано мање на унутрашњи оквир, а више на унапређење алатки и на олакшавање самог рада са Ангуларом у будућности.

5.2 Интерфејс командне линије (Angular CLI)

Ангулар ЦЛИ[3] (енг. Angular CLI) налази се у склопу инсталације Node.js сервера. Овај интерфејс омогућава лакши рад са Ангулар апликацијом, поседује команде за аутоматско креирање апликације по најбољим стандардима, као и команде за интеракцију са другим деловима Ангулар платформе.

Инсталација интерфејса врши се извршавањем команде *npm install -g @angular /cli* било где на диску. Након овога, интерфејс се може користити било где на диску како би се креирао нови Ангулар пројекат, али и како би се омогућила каснија интеракција са тим пројектом.

[2] Хттп захтев - захтев који се шаље неком удаљеном серверу како би се добила одређена информација

[3] CLI (Command Line Interface) - интерфејс командне линије који омогућава лакши рад са апликацијом путем командне линије.

5.2.1 Креирање и покретање Ангулар пројекта

У овом поглављу биће речи о креирању и покретању Ангулар пројекта уз коришћење Ангулар ЦЛИ интерфејса. Подразумевано је да се пре овог корака, одради инсталација Ангулар ЦЛИ, као и Node.js сервера, што је објашњено у претходном поглављу.

Команде за креирање пројекта:

1. ng new [име пројекта] - команда за креирање и иницијализацију пројекта. Након извршења ове команде пројекат садпжи основну AppComponent компоненту и може да се покрене. Пројекат се смешта у истоимени директоријум у оквиру радног простора.
2. cd [име пројекта] - команда за постављање тренутног радног директоријума на директоријум пројекта
3. ng serve - команда за покретање извршавања апликације која представља пројекат. Ова апликација се иницијално покреће тако да служи на порту 4200, али се та подешавања могу променити по потреби.

Након извршења ових корака, апликацији се може приступити на адреси http://localhost:4200.

Поред овога може се додати и Бутстреп фрејморк у Ангулар апликацију користећи команду npm install --save bootstrap@[број верзије за инсталацију]. Након овога, Бутстреп фрејмворк ће бити локално инсталиран и спреман за коришћење. Како бисмо конфигурисали пројекат да користи Бутстреп фрејмворк, треба само да га увеземо тако што ћемо, унутар секције styles, у документу angular-cli.json додати путању до bootstram.min.css документа.

5.3 Структура апликације

5.3.1 Основна структура Ангулар апликације

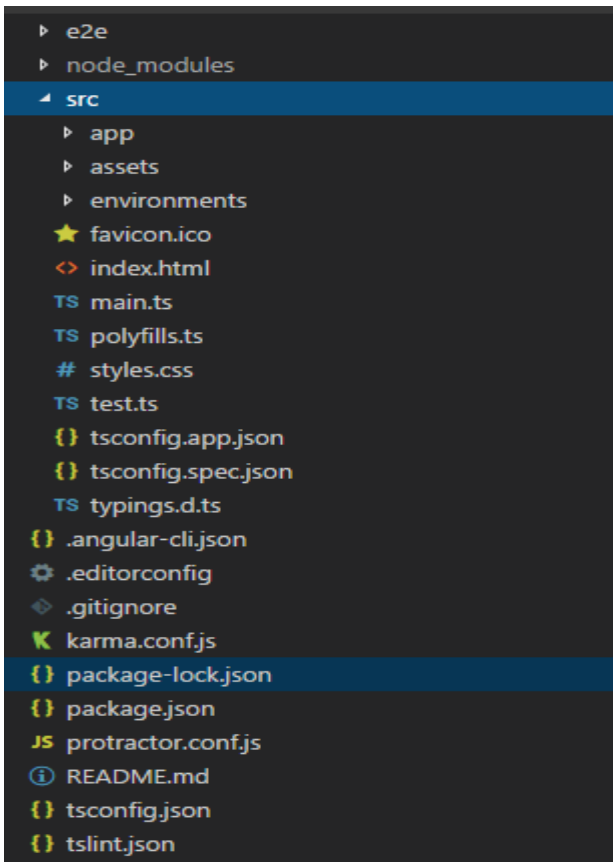
Ангулар апликација састоји се од мноштва компоненти које креирају стабло (поглавље 2).С обзиром да скуп компоненти представља логичку целину, тај скуп се групише у модул (енг. module). Свака апликација креирана Ангулар оквиром мора да се састоји од барем једног модула, а тај модул треба

да садржи корену компоненту. Остале компоненте се угњежђавају унутар те корене компоненте.

Модул је ТајпСкрипт класа која групише скуп функционалности у једну целину. Функционалности које Ангулар оквир пружа су: компоненте, директиве (поглавље 3), филтери (поглавље 5), сервиси (поглавље 6) и модули. Како би се назначило да нека класа представља модул, пре њене декларације ставља се декоратор @Module из @angular/core библиотеке. Декоратору се прослеђује објекат који садржи информације о самом модулу. Неке од тих информација су: како су компоненте, директиве и филтери декларисани унутар модула, који су сервиси пружени унутар модула, који су додатни модули увежени у модул, која је корена компонента модула.

Иако не постоји правило о називу кореног модула, углавном се тај модул назива AppModule. Тај корени модул увози BrowserModule модул који декларише све уграђене функционалности, попут директива и филтера у свој радни оквир (енг. workspace). Тада се те функционалности, нпр. NgIf директива или async филтер, могу користити уз компоненте које су декларисане унутар модула.

Наредбом platformBrowserDynamic().bootstrapModule(AppModule) врши се самопокретање модула(енг. bootstraping) и тиме се назначава да ће се Ангулар апликација извршавати унутар претраживача. Уз апликацију долази и интерни Ангулар преводилац који води рачуна о генерисању делова апликације. Тај преводилац преводи апликацију у тренутку када је она отворена у прегледачу. Делови апликације се динамички освежавају тако да прате измене у коду. Такав поступак превођења назива се јуст-ин-тиме (енг. just in time) превођење. Уз јуст-ин-тиме, постоји и ахеад-оф-тиме (енг. Ahead of time)превођење где се апликација код ког се апликација генерише унапред, без посредства Ангулар преводиоца.



За изградњу апликације обично се користи Вебпек(енг. Webpack)- алат који мноштво мањих датотека повезује у неколико већих пакета, а између осталог, води се рачуна и о томе да се ТајпСкрипт код преведе у ЈаваСкрипт код помоћу ТајпСкрипт преводиоца. Поред тога, Вебпек води рачуна о уношењу страних (енг. Third party) библиотека од којих зависи апликација, али и о томе да смањи количину кода уз поступке умањивања (енг .minify) и поружњивања (енг. uglify). АнгуларЦЛИ интерно користи Вебпек. Коришћењем наредбе ng build можемо да изградимо апликацију уколико је претходно иницијализована коришћењем тог алата. Након што је код преведен и пакети изграђени, они се увозе унутар главног ХТМЛ документа и тиме креирају веб апликацију. Битно је нагласити да се унутар ознаке body ХТМЛ документа мора налазити ознака која одговара селектору корене компоненте кореног модула како би се сав приказ који приказује апликација могао повезати на ту ознаку.

5.3.2 Структура апликације након иницијализације путем Ангулар ЦЛИ

На слици је приказана структура Ангулар апликације након креирања путем команде ng new. У тој структури се налазе неки конфигурациони фајлови, директоријуми са компонентама и тестовима и други аутоматски генерисани фајлови.

Опис структуре:

- e2e[4] је директоријум у ком се смештају интеграциони тестови
- Node-modules је директоријум у ком се смештају предефинисани Ангулар модули, као и они који ће тек бити креирани

[4] e2e (End to End) - ознака за тестирање с краја на краја, односно за интеграционо тестирање

- Src је директоријум у ком се налази изворни код апликације, компоненте, асети, као и нека подешавања окружења..
- App је директоријум који представља основну Ангулар компоненту . Новокреиране компоненте се приказују као поддиректоријуми овог директоријума.
- Поред ових директоријума креирају се и фајлови који се налазе у директоријуму саме апликације и углавном се користе за додатно конфигурисање апликације.

5.4 Компоненте

Основна јединица грађе Ангулар апликације је компонента (енг. component). Компонента у софтверском инжењерству представља независну целину кода која самостално обавља неку функционалност, нудећи и захтевајући притом различите интерфејсе. Конкретно,у случају Ангулар апликације, интерфејс неке компоненте састоји се од улазних својстава (енг. properties) и излазних догађаја (енг. Output events).

Компоненте се користе због неколико практичних и очигледних разлога. Наиме, сама компонента унутар Ангулар апликације појављује се у облику ТајпСкрипт класе. Унутар те класе енкапсулирана је апликативна логика компоненте тако да се лако уочи концепт који та компонента треба да представља. Такође, унутар класе се декларишу и дефинишу чланови који би требало да се налазе унутар компоненте. Будући да је компонента дефинисана класом, више инстанци те компоненте могу се појавити у различитим деловима апликације. Овиме се постиже поновна употребљивост компоненте (енг. reusability). Унутар компоненте могуће је дефинисати и друге компоненте. Компоненте састављене од других компонената креирају стабло[5], опонашајући саму структуру ХТМЛ ДОМ-а (енг. Document Object Model).[6]

[5] свака компонента представља један чвор стабла и приказује се унутар претраживача

[6] **DOM (Document Object Model) -** интерфејс који омогућава да унутар неког програмског језик дохватако садржај неког структурираног документа и манипулишемо њиме. У случају веб развоја ради се о управљању ХТМЛ документом коришћењем Јава скрипт програмског језика. ХТМЛ документ дефинише хијерархијски структуриране елементе који се приказују унутар

5.4.1 Декларисање компоненте

Представљање класе као компоненте:

1. Како би се апликацији показало да нека класа представља компоненту користи се @Component декоратор. Овај декоратор се увози из @angular/core библиотеке на следећи начин:

```
import { AppComponent } from './app.component';
```

2. Декоратор се поставља изнад декларације класе и прослеђују му се одговарајућа својства (поглавље)

```
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

Испред кључне речи *class* додаје се кључна реч *export* како би се омогућило да класа (компонента) буде видљива остатку апликације, тј. да би други делови апликације имали могућност да увозе и користе ту компоненту. Након ове иницијализације, рад са компонентом своди се на рад са класом, односно да декларисање променљивих и имплементације одговарајућих метода.

5.4.2 Својства компоненте

Како би се дефинисало понашање, изглед и улога неке компоненте, на декоратор @Component се додају одређена својства. Објекат који се прослеђује декоратору говори на који ће се начин вршити приказ ХТМЛ кода везаног уз компоненту.

претраживача. Таква структура може се представити путем н-арног стабла - сваки елемент је чвор унутар стабла, а ХТМЛ елемент је корен стабла чија су деца body и head елементи.

Атрибути који се прослеђују објекту:

1. Selector - овај атрибут садржи стринг којим се одређује ознака компонте у апликацији, односно ознака на основу које компонента може да се позове из других компоненти. Све компоненте, изузев корене компоненте неког модула, позивају се из других компоненти. Другим речима, компоненте се угњежђују.
2. Обично се назива као app-[име компоненте]
3. templateUrl - овај атрибут се користи како би се дефинисала путања до спољног ХТМЛ фајла који одређује изглед компоненте.
4. Обично се тај фајл налази у директоријуму компоненте и користи се релативна путања за приступ. Template - уколико је компонента мала и не садржи велики број ХТМЛ тагова, ХТМЛ код који описује компоненту смешта се у виду текста у овај атрибут stylesUrl - овај атрибут се користи како би се дефинисала путања до спољног .css фајла који дефинише стилове који се примењују на компоненту.
   Као и код атрибута templateUrl, тај фајл се налази у директоријуму компоненте и користи се релативна путања за приступ.
5. Styles - овај атрибут садржи стилове који се примењују на компоненту. Углавном се користи као алтернатива атрибуту stylesUrl.
6. Encapsulation[7] - овај атрибут може имати вредности: ViewEncapsulation.Emulated, ViewEncapsulation.None ili ViewEncapsulation.Native. Омогућава да се дефинише која ће се енкапсулација користити унутар компоненте.

Поред ових атрибута, декоратор @Component садржи низ других атрибута чији се описи могу наћи у званичној Ангулар документацији.

[7] Енкапсулација одређује како ће се CSS стил примењивати на HTMLкод компоненте.

- ViewEncapsulation.Emulated - CSS код се примењује само на део кода који припада компоненти и то тако да се HTML елементу и CSS селектору придружују јединствени атрибути
- Prilikom ViewEncapsulation.None - стил дефинисан унутар компоненте примењује се глобално
- ViewEncapsulation.Native - сав стил који би се иначе приказивао постаје недоступан унутар приказа компоненте и примењује се само стил који је дефинисан уз саму компоненту

5.4.3 Увожење компоненте у модул апликације

Ангулар користи модуле како би се компоненте или неки други делови апликације сјединили у неку целину, односно пакет. App Module је главни и основни модул апликације.

Као и компонента, модул се креира тако што се на класу додаје декоратор.

Код кореног модула налази се у фајлу app.module.ts[8]

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

Увожење компоненте:

1. Унутар атрибута declarations треба навести и име новокреиране компоненте, како би се она увезла у корени модул.
2. Како би тај модул знао да препозна компоненту, треба је увести путем команде:

   Import {име компоненте} from ‘./server/[име компоненте].component’

Након ових корака компонента је спремна за коришћење унутар апликације.

[8] App.module.ts - Уколико име корене компоненте није промењено ово је назив фајла. У супротном назив фајла је [име компоненте].module.ts

5.4.4 Креирање компоненте помоћу Ангулар ЦЛИ

Како би се избегле грешке, али и како би се смањило време потребно за креирање компоненте, Ангулар ЦЛИ је представио могућност аутоматског креирања и иницијализације компоненте.

Команда ng generate component [име компоненте] аутоматски креира компоненту, поставља декоратор изнад класе са предефинисаним атрибутима и уписује информације о тој компоненти у главни модул.Након овакве иницијализације компонента је спремна за моделовање. ХТМЛ код се налази у фајлу [име компоненте].component.html, стилови се налазе у фајлу [име компоненте].component.css, док се изворни код компоненте налази у фајлу [име компоненте].component.ts. Модификацијом ових докумената дефинише се изглед и понашање компоненте.

5.5 Повезивање података

Када је реч о компонентама, један од основних принципа који се везује за њих је повезивање података (eng. data binding). Будући да се компонента декларише коришћењем Тајпскрипт класе, на њу се примењују сви основни прицпи објектно-оријентисаног програмирања који важе за класе[9]. С обзиром да је поновна употребљивост компоненте једно од њених својстава, можемо да имамо више инстанци те компонте унутар Ангулар апликације.

```
<app-message></app-message>
<app-message></app-message>
<app-message></app-message>
```

На овом примеру је приказано вишеструко коришћење компоненте унутар шаблона корене компоненте. Креирано је три инстанце message компоненте и свака од њих представља дете корене компоненте, док је корена компонента њихов родитељ. На основу овог примера се јасно види да корена компонента са осталим компонентама креира структуру стабла.

[9] Класе - У општој теорији објектно-оријентисаног програмирања, класа је логичка целина која дефинише неку физичку целину - објекат. Класа дефинише својства објекта, а та својства могу бити подаци и метод.

У претходном примеру креиране су три инстанце исте компоненте без додатних података. У претраживачу би се приказале три потпуно идентичне поруке, односно компоненте. С обзиром да компонента представља неку врсту шаблона за приказ података она мора однекле да добије те податке. Како би се подаци са фронтенда повезали са подацима унутар саме компоненте уводи се принцип повезивања података.

Постоје две врсте повезивања података:

1. Повезивање улазних својстава
2. Повезивање излазних догађаја

У наредном делу биће речи о оба приступа, али пре свега је потребно декларисати податке унутар компоненте како бисмо имали чиме да манипулишемо. Стога, унутар корене компоненте у Тајпскрипт[10] фајлу декларисати променљиве које ће представљати податке. На пример:

```
   message = " Hello world";
```

Овај атрибут је видљив унутар опсега компоненте. Манипулација се може вршити унутар класе или унутар приказа. Уколико желимо да приступ овом атрибуту буде омогућен и остатку апликације, испред имена атрибута додајемо @Input(“[име атрибута]”) декоратор.

```
@Input() message = " Hello world";
```

Користећи интерполацију стринга[11] можемо да приступимо овом својству из ХТМЛ кода компоненте. Такође, уместо назива својства, може се користити и метод чија је повратна вредност стринг или нешто што може да се конвертује у стринг.

```
<app-message> {{ message }} </app-message>
```

5.5.1 Повезивање улазних својстава (енг. Property binding)

Овај вид повезивања података представља могућност да се вредност неког својства компоненте, ХТМЛ елемента, директиве итд. повеже са

[10] Тајпскрипт фајл садржи екстензију .ts

[11] Интерполација стринга (string interpolation) је принцип који нам омогућава приступ својствима Ангулар компоненте коришћењем синтаксе {{ [име својства] }}. Унутар витичастих заграда се наводи име својства компоненте. Оваква синтакса садржи Јаваскрипт код који покреће Ангулар а вредност својства се уграђује у ХТМЛ приказ.

вредношћу атрибута Ангулар класе. На пример, уколико у коду имамо <button disabled>Button</button> у приказу ћемо увек имати дугме које је онемогућено и то се не може променити у време извршавања. Уколико бисмо ипак хтели да омогућимо клик на дугме након одређене акције, морали бисмо да повежемо то дугме са неком вредношћу. У том случају бисмо користили повезивање података методом улазних својстава. Наиме, својство disabled треба повезати са вредношћу неког од атрибута компоненте.

Процес повезивања својства са атрибутом:

1. Декларисати променљиву или метод чију вредност треба повезати са својством,
   Након тога, дефинисати како ће се мењати та променљива.

```
buttonDisabled = false;
```

2. На својство додати [] заграде и доделити му име променљиве компоненте као вредност

```
<button [disabled] = "!buttonDisabled" >
```

Након што се повеже са атрибутом, елемент чије је својство коришћено за повезивање ће динамички мењати своје стање у зависности од вредности тог атрибута.

Овај начин повезивања се у неким случајевима може заменити методом интерполације стринга. Наиме, уколико је циљ повезати и приказати текстуалну вредност треба користити интерполацију стринга. Међутим, уколико је циљ повезати неко својство и везати га за праву вредност променљиве, односно њену нумеричку, логичку или текстуалну вредност, онда треба користити метод повезивања својстава.

5.5.2 Повезивање догађаја (Event Binding)

Метод интерполације стринга и метод повезивања својстава користе се за приказивање података унутар шаблона. Како би апликација, одностно њен приказ, могла да реагује на догађаје уведен је нови тип повезивања података - повезивање излазних догађаја.

На пример, уколико бисмо желели да нешто деси уколико се кликне на дугме из претходног примера, требало би да click догађај тог дугмета повежемо са неким извршним кодом.

Повезивање догађаја се ради на следећи начин:

1. Дефинисати метод који желимо да извршимо. Метод се дефинише у компоненти у чијем се опсегу налази ХТМЛ атрибут.

```
countNumber = 0;

  onClickButtonAction(){

   message = “Poruka broj ” + countNumber; countNumber++;

  }
```

У овом примеру декларисана је променљива countNumber чија је иницијална вредност 0. Идеја је да се након клика на дугме та вредност инкрементира. Метод onClickButtonAction исписује поруку и инкрементира вредност. Како би овај метод био позван треба га повезати са ХТМЛ атрибутом.

2. Повезати ХТМЛ атрибут са извршним кодом, односно методом. Као што се у методу интерполације стринга користе витичасте заграде {}, тако се за повезивање атрибута користе заграде (). Наиме, повезивање се постиже тако што се име догађаја који се везује за елемент смести између заграда и након тога му се додели текстуална вредност која представља име метода за извршавање.

```
<button (click) = “onClickButtonAction()”>Prikazi poruku</button>

<p>{{message}}</p>
```

Битно је напоменути да се, за разлику од ЈавaСкрипт кода, где се клик везивао за onClick догађај, у Ангулар коду клик везује за click догађај. Повезивање се може извршавати за сва својства и догађаје. Добра пракса је да се одради console.log() за елемент над којим желимо да вршимо повезивање како бисмо видели која својства и догађаје он поседује.

5.5.3 Креирање догађаја

Поред повезивања са унапред предефинисаним догађајима, могуће је повезивање и са догађајима који се накнадно креирају у самом коду компоненте.

Кораци за креирање догађаја унутар компоненте су:

1. Дефинисати атрибут типа EventEmitter<> и унутар његовог шаблона проследити жељени објекат
2. Додати декоратор @Output([‘име догађаја’]) како бисмо омогућили приступ овом атрибуту, односно догађају. Овај декоратор омогућује да компонента слуша одређени догађај
3. Након ова 2 корака догађај је креиран и компонента може да га опслужује. Интерфејс EventEmitter садржи метод emit који се користи како би се повезали подаци унутар објекта који је прослеђен у шаблону приликом креирања догађаја.

```
  @Output() componentChanged = new EventEmitter<{name: string, age:
number}>();

  name: string;

    age: number;
```

```
  onComponentChanged(){

    this.componentChanged.emit({

      name : this.name,

      age: this.age

    });

  }
```

5.5.4 Повезивање компоненте прегледа са тајпскрипт кодом компоненте

Декоратор @ViewChild омогућава нам да, из кода, приступимо неком елементу из прегледа компоненте. То се постиже тако што се на тај елемент

дода локална референца[12], а затим се направи атрибут унутар компоненте са @ViewChild декоратором коме је прослеђена локална референца елемента у форми текстуалног уноса.Атрибут коме се припаја вредност елемента прегледа је типа ElementRef. Позивом *this.itemName.nativeElement.value* добијамо вредност елемента.

```
@ViewChild('itemName') itemName : ElementRef;

value:string = this.itemName.nativeElement.value;
```

5.5.5 Пројекција садржаја

Уколико бисмо при инстанцирању компоненте, између тагова за отварање и затварање, покушали да додамо неки садржај, апликација би се успешно превела, али се тај садржај не би приказао.

Како би компонента постала још скалабилнија и флексибилнија, уведен је механизам пројекције садржаја. Овим механизмом прослеђујемо додатни ХТМЛ код повезан са Ангулар кодом унутар компоненте. Унутар шаблона компоненте у којој желимо да пројектујемо неки садржај додајемо таг <ng-content></ng-content>.

```
@Component({
  selector: 'complete-component',
  template: `EmpDetails: <ng-content></ng-content> {{ emp.EmpName }}`
  })
```

Могуће је користити и више ознака које представљају неки садржај. То се постиже тако што се додаје атрибут select унутар <ng-content> ознаке.

```
<ng-content select=”x1”></ng-content>
 <ng-content select=”x2”></ng-content>
```

5.5.6 Животни век компоненте

Свака компонента, почев од настанка, тј. инстанцирања, па све до уништења, пролази кроз различите фазе које описују њено стање. У зависности од стања, могуће је извршити сљедеће функције:

[12] Локална референца је механизам који омогућава референцирање неком ХТМЛ елемента путем ручно унетог имена. Ово име се користи као нека врста идентификатора тог елемента. Додаје се тако што се у оквиру тага елемента упише тараба, а затим жељено име. На пример: <input value=”Unos” type=”text” #tekstualniUnos></input>. У овом примеру, ознака tekstualniUnos представља локалну референцу на овај input елемент.

- ngOnChanges()
- ngOnInit()
- ngDoCheck()
- ngAfterContentInit()
- ngOnChanges()
- ngOnInit()
- ngDoCheck()
- ngAfterContentInit()
- ngAfterContentChecked()
- ngAfterViewInit()
- ngAfterViewChecked()
- ngOnDestroy()

Ангулар оквир у модулу @angular/core садржи интерфејсе који се могу имплементирати унутар компоненте како би се позвала одређена од горе наведених функција приликом неког тренутка у животном веку компоненте. Нпр., уколико назначимо да компонента имплементира интерфејс OnInit, тада унутар класе компоненте дефинишемо методу ngOnInit() која се покреће сваки пута када је компонента иницијализована. Такав механизам је користан за припрему података који се налазе у компоненти, али и приликом уништавања компоненте.

Три најважније методе које се покрећу у животном веку компоненте су: ngOnChanges() - покреће се сваки пут приликом промене улазних својстава при повезивању података,

ngOnInit() - покреће се након првог ngOnChanges() позива, и

ngOnDestroy() - покреће се приликом уништавања компоненте, односно њеног уклањања из приказа неке родитељске компоненте.

Користећи парадигму реактивног програмирања којом на функционалан начин обликујемо модел апликације, уобичајено је да се користи класа Observable из библиотеке RxJS. Том класом омогућено је активно праћење промене неког података и то тако да се, нпр. приликом иницијализације компоненте у функцији ngOnInit(), претплатимо (енг. subscribe) на Observable објекат који ће слати податке, а приликом промене о томе обавестити компоненту. С обзиром да компонента активно слуша све промене података јер је претплаћена на Observable инстанцу, односно заузела је ресурс, исти тај

ресурс ради ефикасности треба одјавити(енг. unsubscribe), што се може учинити приликом позива ngOnDestoy()

5.6 Директиве

Директива представља механизам којим мењамо структуру, понашање или изглед ДОМ-а. Најчешће коришћена врста директиве су, претходно описане, компоненте. Компонента је директива с шаблоном који дефинише приказ унутар претпраживача. Постојање шаблона код компоненти је уједно и оно што их разликује од осталих типова директива, па се зато често обрађују као засебна целина.

Технички гледано, директива је Тајпскрипт (енг. TypeScript) класа која садржи чланове којима утичемо на ДОМ, структуру, изглед и понашање. Као и код компоненти, ова класа такође мора да садржи декоратор у коме је дефинисан селектор директиве. Директива се користи тако што се њен селектор придода тагу компоненте, односно елемента прегледа којим желимо да манипулишемо. Будући да је директива представљена класом, она мора имати своју декларацију и дефиницију. Како би се назначило да је нека класа директива, користи се декоратор @Directive који се увози из @angular/core библиотеке. Као и код компоненти, декоратору директиве се прослеђује објекат у коме се дефинише та директива. Један од атрибута овог објекта је селектор у коме се дефинише име директиве, односно ознака помоћу које можемо да приступимо директиви. Како би се омогућио приступ без навођења угластихзаграда, ове заграде се наводе унутар селектора.

Директиве се користе тако што се придруже елементу и на тај начин омогућавају манипулисање њиме и његовим својствима. Имајући то у виду, јасно је да исход коришћења директиве не би требало да се мења у зависности од грађе елемента. На пример - уколико позивамо директиву која мења боју позадине неког елемента, исти исход би требало да се догоди независно од тога да ли мењамо боју h1 или p елемента.

Директиве се деле у три категорије:

- Компоненте
- Структуралне директиве(eng. structural directives)
- Атрибутне директиве (eng. attribute directives)

Ангулар апликација долази са мноштвом предефинисаних и корисних структуралних и атрибутних директива. Поред овога, могуће је развити и нове, ручно дефинисане директиве сходно потребама саме апликације.

5.6.1 Структуралне директиве

Структуралне директиве мењају структуру ДОМ-а. С обзиром да су елементи ДОМ-а видљиви унутар прегледа, може се рећи да ове директиве утичу на садржај унутар претраживача, односно на преглед доступан кориснику. Ове директиве могу дуплирати, уклонити или преместити неки елемент унутар ДОМ-а. Предефинисане структуралне директиве унутар Ангулар апликације су NgIf, NgFor и NgSwitch.

NgIf директива ва директива се користи како бисмо могли да приказујемо нешто у зависности од неког услова. У зависности од логичке променљиве (енг. Boolean variable), ова директива уклања, односно приказује, елемент ком је придружена. Уколико је променљива истинита, елемент остаје садржан унутар ДОМ-а. У супротном, елемент бива уклоњен.

```
1 <p *ngIf="true">Елемент у ДОМ.у</p>

2 <p *ngIf="false">Еелемент није у ДОМ-у</p>
```

Директива ngIf својим понашањем подсећа на hidden својство садржано унутар сваког ХТМЛ елемента. Међутим, послтоји велика разлика између ова два начина приказивања елемената. Наиме, уколико је услов hidden својства задовољен, елемент ће бити сакривен, односно невиљив у оквиру прегледа, али ће ДОМ и даље садржати тај елемент. Стога, и даље ће бити могућ сваки вид манипулације тим елементом. С друге стране, директива ngIf у потпуности уклања елемент из ДОМ-а и онемогућава било какав приступ том елементу. Ово понашање је од посебне важности уколико се ради о кардиналним деловима унутар Ангулар апликације.

На пример, код RouterModule модула(овде додај линк) описаног у поглављу…, користи се инстанца RouterOutlet директиве, Претпоставимо да апликација не дозвољава приказ садржаја, до кога се стиже путем усмеривача, уколико корисник није ауторизован. У овом случају, у зависности од статуса ауторизације, треба онемогућити приказ заштићеног садржаја, и то се ради

користећи hidden атрибут. Уколико бисмо користили ngIf директиву, која условно уклања RouterOutlet директиву, тада бисмо добили грешку у извршавању кода, с обзиром да та директива у неком тренутку не би била садржана у ДОМ-у.

Директива **ngFor** омогућава да итерирамо кроз поље које садржи већи број елемената (енг. array). Како би ова директива имала приступ пољу, оно мора да буде декларисано у опсегу компонент у оквиру чијег је приказа позвана ова директива. Најчешће се ова директива користи за креирање више инстанци неког ХТМЛ шаблона, или приликом коришћења неког вида повезивања података.

Додатне локалне променљиве шаблона које се могу користити приликом итерирања помоћу ngFor директиве су:

- Index - индекс итерације
- First - логичка променљива која означава да ли се ради о првом елементу унутар поља
- Last - логичка променљива која означава да ли се ради о последњем елементу унутар поља
- Even - логичка променљива која означава да ли је индекс паран број
- Odd - логичка променљива која означава да ли је индекс непаран број

Додатни механизам који се може користити приликом позива ngFor директиве је trackBy атрибут. Овај атрибут се користи како бисмо постигли побољшање перформанси. Користи се тако што му се придружује име атрибута који садржи неки јединствени податак (нпр. Примарни кључ који се налази у бази података), а који се налази унутар сваког елемента кроз који итерирамо. Приликом итерирања кроз елементе поља, за сваки елемент стварамо нови скуп елемената унутар приказа, тако да, уколико се поље кроз које итерирамо динамички промени, trackBy механизмом се проверава да ли је неки од елемената поља остао исти. Ово проверавање се врши користећи тај јединствени атрибут. Уколико је неки елемент поља остао исти, он се не уклања из ДОМ-а.

NgSwitch директива se, kao и ngIf директива, ова директива се такође користи за динамичко уклањање односно приказивање елемената из ДОМ-а. Разлика је у томе што, код ngSwitch директиве, одлуку о приказу вршимо у зависноси од гранања случајева. Сама логика ове директиве се, у потпуности, може заменити ngIf директивом, али нам ngFor омогућује јаснији запис, који уједно може бити и економичнији у погледу величине кода.

У наставку је приказан пример ngSwitch директиве.

```
<div [ngSwitch]="12">
  <p *ngSwitchCase="12">12</p>
  <p *ngSwitchCase="13">13</p>
  <p *ngSwitchCase="14">14</p>
</div>
```

5.6.2 Атрибутне директиве

Поред структуралних директива, Ангулар поседује и атрибутне директиве. Помоћу ових директива манипулишемо атрибутима елемената. Ове директиве не мењају структуру ДОМ-а, већ изглед и понашање. Уграђене атрибутне директиве су ngStyle, ngClass и ngNonBindable.

ngStyle директива: Ова директива се користи како би се неком ХТМЛ елементу динамички придружио ЦСС стил. Самој директиви, која се повезује са елментом путем селектора, прослеђујемо објекат који дефинише њен ЦСС стил. На пример, уколико бисмо хтели да текст унутар неког елемента буде црвене боје, директиву позивамо кодом [ngStyle] = “color:red”. Као што се види из примера, ова директива прихвата објекат који дефинише стил. Тај објекат представља локалну променљиву шаблона, или пак атрибут компоненте у оквиру које се позива. Будући да се, као дефиниција стила, овој директиви прослеђује објекат, изглед и сам стил тог елемента могу динамички да се мењају.

ngClass директива: Ова директива омогућује динамичко додавање ЦСС класе елементу коме је придружена. Поступак је идентичан као и код ngStyle директиве, с ти да ngClass директиви прослеђујемо објекат чији су атрибути имена ЦСС класа, а вредности логичке променљиве. У зависности од истинитосне вредности тих атрибута, класе ће бити примењене, односно игнорисане.

ngNonBindable директива: Ова директива се додаје унутар елемента који не желимо интерпретирати у смислу Ангулар синтаксе. Најчешће се ова директива користи на страницама које говоре о самом Ангулар оквиру, односно подучавају о њему, како би се код приказао без претходне интерпретације.

5.6.3 Креирање сопствених директива

Поред уграђених директива, могуће је креирати и сопствене директиве уколико апликација то захтева. Директиве се, као и компоненте, дефинишу уз помоћ ТајпСкрипт класа.

Креирање структуралне директиве:

1. Дефинисати директиву:
   Креирати Тајпскрипт класу и додати јој @Directive декоратор. Овом декоратору треба проследити објекат у ком је дефинисан изглед директиве. Један од параметара овог објекта је selector чија вредност представља име директиве. Име директиве се уписује у селектор са угластим заградама како би се касније могло приступити директиви директно, без навођења заграда.
2. Додати компоненту у module.ts фајл у одељак Directives
3. Како би се могло приступати елементу коме је директива придружена, прослеђује се објекат типа ElementRef. Елемент којим се иницијализује овај објекат је домаћин (енг. Host element).
   constructor(private elementRef:ElementRef){}
4. Након дохватања елемента, он може да се користи како би се манипулисало његовим изгледом, понашањем и слично. То се ради тако што се позивају операције над атрибутом nativeElement.
   ```
   elementRef.nativeElement.style.backgroundColor = "red";
   ```
5. Придружити име директиве жељеном елементу
   <p imeDirektive>Ово је директива! </p>

5.6.4 Рендерер

Директна манипулација над елементом, као што је описано у претходном поглављу, није препоручива. Како бисмо избегли директно приступање

елементу и његовим пољима, користи се рендерер (енг. Renderer). Овај објекат омогућује манипулацију елементима ДОМ-а користећи методе за приступ. Иницијализација и приступ рендереру се такође врше из конструктора, као и код ElementRef.

```
  constructor(private elementRef:ElementRef, private      renderer:Renderer2){

    this.renderer.setStyle(this.elementRef.nativeElement, 'background-color',
"red" );

  }
```

5.6.5 Директиве и догађаји

Како бисмо омогућили динамичко мењање стилова коришћењем директива, морамо да омогућимо да та директива реагује на одређене догађаје који се дешавају над придруженим елементом. То се ради коришћењем декоратора @HostListener и @HostBinding. Ови декоратори служе за праћење стања домаћина.

HostListener: Декоратор @HostListener назначава догађај на који желимо да одреагујемо (нпр. Клик мишем). Ради тако што му се у параметрима проследи име догађаја, а он се привеже методу који желимо да извршимо.

На пример, уколико желимо да извршимо неки код након што корисник пређе мишем користићемо догађај mouseEnter као што је приказано у примеру. Приступ подацима догађаја је, код овог декоратора, омогућен објектом типа Event који се прослеђује као параметар позваног метода.

```
  @HostListener('mouseenter') mouseOver(eventRef: Event){

    this.renderer.setStyle(this.elementRef.nativeElement, 'background-color',
"blue" );

  }
```

HostBinding: Помоћу овог декотатора, назначавамо које својство елемента домаћина желимо да надоградимо, односно да мењамо. То се ради тако што се дода атрибут у класу директиве, на њега се дода декоратор @HostBinding коме се прослеђује име својства које желимо да повежемо за тај атрибут.

```
@HostBinding('style.backgroundColor') backgroundColor = ‘transparent’;
```

Након овога, мењањем атрибута backgroundColor унутар директиве, мењамо боју позадине домаћег елемента.

Поред ова два декоратора, код директива се , као и код компоненти, може користити декоратор @Input како би се вршило повезивање својстава.

Креирање структуралне директиве:

1. Поновити кораке 1 и 2 из одељка (Креирање атрибутивне директиве)
2. Како би се омогућило мењање структуре ДОМ-а потребно је имати приступ шаблону и прегледу. Приступ овим објектима се такође добија кроз параметре конструктора, као што је описано за елемент и рендерер у претходном поглављу. Стога, објекат који омогућује приступ шаблону је типа TemplateRef, док је објекат који представља преглед типа ViewContainerRef.

```
  constructor(private templateRef:TemplateRef<any>, private
view:ViewContainerRef){  }
```

3. Након овога, коришћењем објеката шаблона и прегледа можемо да дефинишемо понашање директиве
4. Након што смо дефинисали шта ће директива радити, треба повезати ту директиву са елементом. Ово се ради уз коришћење * уз име директиве.

```
<p *customDirective="uslov"></p>
```

5.7 Сервиси

С обзиром да све врсте апликација, независно од платформе, из дана у дан постају све функционалније и комплексније, нужно је правилно расподелити њихове функционалности. Једно од најбитнијих својстава добре апликације је ажурираност података. Уколико бисмо променили неки податак, нужно је да та промена одмах буде видљива у свим деловима апликације, како због корисничког искуства, тако и због апликативне логике. Стога, тај податак мора бити представљен као јединствена инстанца неке класе, односно јединствена вредност неког типа података.

Један од механизама који обезбеђује ажурираност података је претходно описано повезивање података. Међутим, овај механизам обезбеђује само повезаност на релацији родитељ-дете. Остале компоненте родитеља, али и деца детета неће имати приступ податку. Једини начин да се омогући приступ из других компоненти је емитовање догађаја који ће прослеђивати податке као аргументе. Овај начин постизања ажурних података са собом повлачи и много писања кода. Саме компоненте треба да се користе за повезивање података, њихово приказивање и праћење интеракције корисника. Међутим, извор података, који се смештају унутар компоненте, не треба да буде у самој компонент, односно у њеном опсегу важења.

Механизам који омогућава да податке смештамо ван компоненте, али да их потом користимо унутар компоненти је сервис. Сервиси представљју "глобалне" објекте унутар апликације који садрже податке који се ,потом, смештају у компонентама. Сервис је инстанца неке класе или вредност неког типа података доступна у свим деловима апликације. Идеја сервиса базирана је на концепту убризгавања зависности (енг. Dependency injection). Кажемо да је део апликације А зависан од дела апликације Б, односно обрнуто.

У погледу зависности, уобичајен случај је да једна класа користи другу за рад, Лоша пракса је инстанцирати објекат од кога класа зависи унутар зависне класе. Оваква пракса се назива чврсто спајање (енг. tight coupling). У том случају долази до отежаног тестирања и одржавања кода. Пракса супротна овој, темељи се на идеји да се инстанцирани објекат, који представља зависност, проследи путем конструктора класе. Овакав метод назива се лабаво спајање (енг. loose coupling) и управо он представља метод на коме се базира убризгавање зависности код сервиса.

5.7.1 Убризгавање зависности

Ангулар нуди врло флексибилан начин за убризгавање зависности. Идеја се темељи на четири ентитета - токен, убризгавач (енг. injector), снабдевач (енг. provider) и зависност.

Низ снабдевача је поље које региструје све зависности унутар аппликације или деа апликације. Сваки објекат, којим је дефинисан снабдевач, садржи атрибут provide којим се назначује јединствени идентификатор

зависности, односно токен те зависности. Другим речима, снабдевач мапира токен одговарајуће зависности. Та зависност се може дохватити у виду инстанце класе, ЈавaСкрипт објекта, поља или променљиве. Токен зависности може бити стринг, име класе (тзв. Type token) или инстанца генеричке класе InjectionToken. Коришћење стринга као токена се избегава јер доводи до колизије.

Снабдевач, зависно од прослеђеног токена дефинише резултат који ће бити убризган. Други атрибут објекта, којим је дефинисан снабдевач, може бити useClass, useExisting, useValue или useFactory.

Атрибут useClass означава да ће снабдевач креирати јединствену инстанцу класе користећи прослеђени токен. Та инстанца ће бити убризгана свим осталим класама које је захтевају као зависност. Интерно се таква инстанца смешта унутар регистра зависности (енг. dependency registry) како бисмо исту инстанцу класе користили у различитим деловима апликације.

Атрибут **useExisting** означава да ће се, уз већ регистровани снабдевач, дефинисати нови који користи вредности постојећег. Атрибут useValue означава да ће снабдевач креирати променљиву

Атрибут **useFactory** означава да ће снабдевач позвати функцију. Уколико се, унутар те функције, инстанцира неки објекат, он се такође чува унутар регистра. Та функција може само да прими параметре који се налазе унутар регистра.

Снабдевач се дефинише, односно прослеђује, компонентама и модулима путем поља providers. Другим речима, модули, компоненте и директиве могу да имају своје снабдеваче.

```
providers: [
  {provide: 'api', useValue: api_url},
  {provide: 'value', useValue: 'init'},
  {provide: MainHttpService, useClass: MainHttpService},
  {provide: 'FactoryDependency', useFactory:
        (value)=>{
            return new ServiceClass(value)
        }, deps: ['value']}
]
```

Механизам убризгавања зависности стара се о томе да све зависности унутар снабдевача буду регистроване, али и да по потреби могу да буду

убризгане тамо где су потребне. За само убризгавање стара се убризгавач. Он, у зависности од прослеђеног параметра, односно токена зависности, дохвата инстанцу из регистра и убризгава је. Убризгавач се додаје помоћу декоратора **@Inject,** а смештамо га пре параметра који означава зависност унутар конструктора класе која садржи зависност.

```
  constructor(@Inject('api') private api_url: string) { }
```

У овом примеру, снабдевач с токеном “api” снагдева вредношћу атрибут “api-url”.

Поред @Inject декоратора, користи се **@Injectable** декоратор како би се назначило убризгавање зависности. Уколико класа поседује декоратор који садржи мета податке (енг. metadata), као што су класе компоненти и декоратора, нема потребе за коришћењем @Injectable декоратора. Наиме, декоратори за компоненте и директиве у себи већ садрже мета податке потребне за убризгавање зависности. Најчешће се, овај декоратор, користи при убризгавању зависности између два сервиса.

Поступак креирања сервиса и додавања зависности:

1. Креирати класу и дефинисати атрибуте, односно методе којима треба приступати
2. У оквиру декоратора компоненте треба додати нову секцију - “providers”

```
 @Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css'],
   providers:[UserService]
 })
```

Уколико се провајдер дода у app-module, тај сервис постаје доступан целој апликацији. Ово се ради уколико је подребно коришћење сервиса у оквиру неког другог сервиса. Уколико се пак провајдер сервиса дода у неку компоненту, та инстанца сервиса постаје доступна у тој компоненти и у свим компонентама деце.

Овде посебно треба обратити пажњу, будући да, уколико бисмо провајдер истог сервиса додали у неку од тих компоненти деце, тиме би се креирала нова инстанца сервиса и она би прегазила претходно креирану инстанцу родитељске компоненте.

3. Убризгати референцу на сервис унутар конструктора компоненте

```
constructor(private userService: UserService){ }
```

Након ова три корака, сервис је спреман за коришћење и дељени подаци постају доступни.

5.8 Рутирање

Унутар једностране апликације приказују се само компоненте. Међутим, оне се смештају на такав начин да корисник има утисак рада са вишестраном апликацијом. Овакво понашање постиже се тако што увек постоји фиксан приказ једне компоненте, док се остали прикази динамички мењају. Фиксна компонента служи за промену садржаја унутар прегледа и назива се навигација. Унутар навигације су смештени линкови, дугмад или слични елементи који, након што су притиснути, генеришу нови приказ. Овај механизам интеракције између навигатора и осталих компоненти назива се рутирање, и омогућује га Ангуларов специјални модул - RouterModule.

У наставку је дат опис поступка креирања рута и повезивања тих рута са навигатором.

1. Пре свега, потребно је увести RouterModul из `@angular/router` библиотеке
2. Након овога, потребно је дефинисати руте којима ћемо динамичк стварати приказ унутар претраживача, односно, кретати се унутар апликације. То се ради тако што се дефинише константан низ објеката који садрже информације о рутама. Овај низ се такође увози из `@angular/router` библиотеке и његов тип је Routes. Објекти смештени у низу имају по два атрибута, а то су “path”, који представља путању, и “component”, који се односи на компоненту која ће бити учитана.

```
const appRoutes: Routes = [
  { path :'', component: HomeComponent},
```

```
  { path :'about', component: AboutComponent},
  { path :'shoppingList', component: ShoppingListComponent}
  ];
```

3. Поље типа Routes, у коме су садржане руте, прослеђује се статичкој методи RouterModule.forRoot која инстанцира модул који садржи све директиве, руте и сервисе потребне за рад са усмеривачем (енг. router). Све ово се ради у одељку imports, у класи app-module.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpModule,
  RouterModule.forRoot(appRoutes)
  ],
```

4. Унутар приказа у ком желимо да се приказују компоненте регистроване у рутама, убацујемо RouterOutlet директиву. На истом приказу можемо дефинисати и навигацију, међутим чешће се прави одвојена компонента за навигацију.

```
<div class="col-md-12">
  <router-outlet></router-outlet>>
  </div>
```

5. Креирање навигације се ради тако што се навигациони елементи, односно елементи на које корисник треба да кликне да би се нешто десило, претворе у линкове за рутирање. Ово се ради тако што се на линк, дугме или неки други елемент, дода директива RouterLink, чијем селектору прослеђујемо текстуално поље у коме је садржана адреса. Ангулар ће затим дохватити руте из поља креираног у другом кораку (поље које садржи низ дефинисаних рута) и упоредити их са прослеђеним параметром. Уколико је параметар валидан, односно уколико се налази у предефинисаном низу дозвољених рута, извршиће се рутирање.

```
<ul class="nav navbar-nav">
  <li routerLinkActive= "active"><a routerLink="" >Recipes</a></li>
```

```
  <li routerLinkActive= "active"><a routerLink="'about'">About </a></li>

  <li routerLinkActive= "active"><a routerLink="shoppingList">Shopping
List</a></li>

</ul>
```

RouterLinkActive директива омогућује да се линку који је активан придружи ЦСС класа 'active' како бисмо додатним ЦСС кодом нагласили на ком линку се налазимо и тиме побољшали корисничко искуство.

Директива **RouterLink** може да прими апсолутну или релативну путању ка некој компоненти. Уколико користимо апсолутну путању, она ће се увек срачунавати у односу на корени модул. Стога ће, уколико бисмо проследили путању до “about” стране као routerLink = “about”, адреса коју гађамо бити localhost:4200/about.

С друге стране, уколико бисмо користили релативну путању, она би се увек рачунала у односу на тренутну адресу, односно адресу са које је корисник извршио клик на линк. Тада бисмо за путању routerLink = “/about”, позвану из shoppingList компоненте, имали адресу *localhost:4200/shoppingList/about.*

Поред статичког додељивања вредности линкова sa рутирање, постоји и динамички начин за њихово мењање, односно креирање.Како бисмо могли да мењамо линкове, односно руте, динамички, потребан нам је приступ инстанци рутера, али и приступ тренутно активној рути уколико желимо да имамо релативну путању. Уколико користимо апсолутну путању, тренутно активирана рута нам није потребна.

Инстанцу рутера добијамо коришћењем механизма убризгавања зависности у конструктору. На исти начин се долази и до инстанце тренутно активне руте (енг. ActivatedRoute).

```
  constructor(private route : Route) { }

  ngOnInit(){

  this.router.navigate(["api", "v1", "shoppingList"]);

  }
```

Код: дохватање инстанце рутера и мењање навигације до путање localhost:4200/api/v1/shoppingList. Параметар навигације је низ стрингова који представљају путању.

```
  constructor(private router:Route, private
activeRoute:ActivatedRoute){}

   ngOnInit(){

this.router.navigate(["api","v1","shoppingList"],{relativeTo:this.activ
eRoute});

}
```

5.8.1 Прослеђивање параметара

Уколико желимо да дефинишемо руту која у себи садржи неке параметре, потребно је да то дефинишемо приликом иницијализације низа рута садржаног у app-module модулу. Ово се постиже тако што се у назив руте уметне назив параметра испред ког се додаје ':' симбол. На пример, рутом "users/:id/:name" прослеђујемо параметре "id" и "name". Овим параметрима се приступа преко објекта типа ActivatedRoute који се убризгава путем конструктора. То се ради тако што се дохвати атрибут "snapshot", а потом из њега, преко атрибута "params" дохватимо жељени атрибут.

```
ngOnInit(){
  this.user = {
    id: this.router.snapshot.params["id"];
    name: this.router.snapshot.params["name"];
  }
}
```

Међутим, проблем са овим приступом се јавља уколико из већ учитане компоненте кликнемо на линк који води натраг на исту компоненту, али са различитим подацима у рути. На пример, уколико бисмо, са странице localhost:4200/users/1/Ana, имали линк који води на следећег корисника, који би се такође учитао у оквиру исте компоненте, али би имао различите параметре. Наиме, клик на такав линк би учитао нову путању, односно localhost:4200/users/2/username, али би кориснички подаци приказани на тој

страни остали непромењени. Ово се дешава зато што Ангулар, по свом уобичајеном понашању, не учитава поново компоненту на којој се већ налазимо већ само мења путању. Ова особина платформе је добра јер тако штедимо перформансе. Како бисмо реаговали на догађај, односно промену параметра, треба да пратимо стање тог параметра. Ово се ради тако што се, на месту параметра “snapshost”, користи парметар “params” који представља Observable објекат који реагује на промене.

```
ngOnInit(){

  this.user = {

    id: this.router.snapshot.params["id"];
    name: this.router.snapshot.params["name"];

}

  this.router.params.subscribe(

    (params:Params) => {

      this.user.id = params['id'];

      this.user.name = params['name'];

    }

  )

}
```

Уколико знамо да компонента не позива саму себе, односно нема потребе за динамичким учитавањем података из параметара, онда можемо да користимо први приступ, без “params” објекта. Међутим, пракса је да се користи други приступ како би се избегле грешке.

Уколико додатно желимо усмеривач унутар усмеривача, Користимо атрибут “children” унутар објекта који представља руту. Вредност тог атрибута је ново поље рута.

```
{path: 'product/:id', component: ProductComponent, children: [

  {path: 'details', component: ProductDetailsComponent},

  {path: 'order', component: ProductOrderComponent}

  ]

}
```

С обзиром да се у том случају креира нови усмеривач, потребно је поново дефинисати RouterOutlet директиву. Ову директиву дефинишемо унутар родитељске компоненте. Уколико рута има прослеђених параметара, њима се приступа тако што се, унутар компоненте детета, убризга инстнца ActivatedRoute класе и, након тога, претплаћујемо се на параметре родитеља(this.activatedRoute.parent.params).

5.8.2 Кретање коришћењем чувара (енг. Guards)

Кретање унутар усмеривача се додатно може ограничити коришћењем чувара (eng. guards). Чувари су посебне методе које враћају логичку променљиву или посматрача (eng. Observable) који шаље логичку променљиву у зависности од тога да ли је кориснику дозвољено кретање унутар усмеривача. Чувари имплементирају ‘интерфејсе CanActivate и CanDeactivate и тиме дозвољавају, односно не дозвољавају усмеривачу приказ неке компоненте. Све чуваре за неку руту стављамо унутар поља које је вредност атрибута canActivate унутар објекта који дефинише руту.

```
@Injectable()

export class IsAuthorizedGuard implements CanActivate {

  constructor(private authService: AuthService){}

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> | Promise<boolean> | boolean {

    return this.authService.isAuthorized();

   }

}
```

5.9 Форме

Принцип форми, унутар ХТМЛ документа, заснива се на постојању поља за унос података и дугмета за слање тих података. Подаци се путем форме шаље даље, до неког сервиса у коме се врши обрада података, или се пак шаље директно на РЕСТ (енг. REST) крајње тачке (енг. endpoints). У класичним веб апликацијама, приликом слања форме, шаље се цела страница на којој се форма налази. Подаци који се прослеђују, парсирају се унутар ХТМЛ документа и прослеђују се у ХТТП захтеву. Приликом прослеђивања података, треба

водити рачуна о њиховој исправности. Ово се постиже увођењем валидатора (енг. validators) у контроле форме.

Контроле форме су:

- Улазна јединица за текст (енг. Text Input)
- Улазна јединица за датотеку (енг. File Input)
- Оквир за означавање (енг. Check Box)
- Радио дугме (енг. Radio Button)
- Падајући изборник (енг. Drop Down Button)
- Дугме за подношење форме (енг. Submit Button)

Обрада форми у Ангулар апликацијама је у многоме другачија, у односу на ону код конвенционалних апликација. Наиме, будући да је Ангулар апликација по својој структури једнострана апликација, не можемо да имамо прослеђивање целе странице приликом прослеђивања форме. Стога, унутар Ангулар апликације имамо многе додатке, обезбеђене од стране Ангулар платформе, за рад са формама. Код ових апликација, подаци форме се енкапсулирају унутар ЈавaСкрипт објекта, односно ЈСОН (енг. JSON) објекта, путем ХТТП протокола.

Постоје два принципа за коришћење форми унутар Ангулар апликација. Први принцип рада форме заснива се на концепту повезивања података, с обзиром да се форма налази унутар приказа неке компоненте. Подаци унешени путем форме се повезују за атрибуте компоненте, а из компоненте се шаљу до сервиса, или се пак обрађују унутар саме компоненте. Овај принцип се назива се шаблонски принцип, с обзиром да се ослања на податке који се уносе користећи приказ.

Поред шаблонског принципа и уношења података из приказа, имамо могућност да целу логику форме убацимо унутар кода компоненте у којој се форма налази. Овакав приступ приказивања форме и прикупљања података назива се реактивни приступ.

5.9.1 Форме засноване на шаблону

Шаблонске форме, односно форме засноване на шаблону (енг. Template-driven form), дефинишу се унутар шаблона неке компоненте. Ове форме се лако праве, али, како би рад са њима био могућ, мора се увести модул “Forms Module” унутар главног модула у ком правимо апликацију. Ово је неопходно како бисмо користили директиве NgForm и NgModel.

NgForm директива се уграђује на елемент форме унутар прегледа компоненте. Након овога можемо да приступимо форми користећи локалну променљиву шаблона.

NgModel директива се користи како би се означило да неки елемент представља контролно поље форме.

Поступак креирања форме код шаблонском приступа:

1. Увести FormsModule модул у главни модул апликације
2. Креирати форму у приказу, односно направити жељени распоред контролних поља у оквиру <form> </form> тагова
3. Додати ngModel директиву унутар сваког контролног елемента који желимо да прослеђујемо
4. Додати name атрибут свим контролним елементима како би се омогућило повезивање тог елемента са одређеним именом
5. Проследити форму:
   ! Не користити дугме унутар форме за позив метода за прослеђивање.
   Ово дугме прослеђује целу форму, одсносно целу страницу
   Уместо овог приступа, Ангулар је увео директиву ngSubmit која се уграђује унутар елемента форме.

```
<form (ngSubmit) = "onSubmit()">
```

5.9.2 Приступање подацима шаблонске форме:

Постоји два начина да се приступи подацима који су прослеђени унутар форме. Први је **коришћењем локалне референце**, а други је приступ коришћењем @**ViewChild** декоратора.

Како би се приступило подацима коришћењем локалне референце, потребно је:

1. Додати локалну референцу на елемент форме. Овиме се назначава Ангулару да омогући приступ елементу форме. Тај елемент се затим прослеђује као параметар метода за прослеђивање података форме ngSubmit.

   <form (ngSubmit) = "onSubmit(forma)" #forma = "ngForm">

2. Додати метод за прослеђивање форме у ТајпСкрипт класу компоненте. Овај метод очекује објекат типа NgForm као параметар.

```
onSubmit(form: NgForm){
  console.log(form.value);
}
```

Како би се приступило подацима коришћењем декоратора @ViewChild потребно је у ТајпСкрипт коду компоненте позвати:

```
@ViewChild('forma') signUpForm : NgForm;
```

Овај приступ се најчешће користи када желимо да приступимо форми пре него што се притисне дугме за прослеђивање

5.9.3 Валидација форме

Код шаблонског приступа, валидација елемената форме се постиже користећи директиве. Ангулар има уграђене директиве за валидацију, али такође препознаје и постојеће директиве, попут директиве required. Пример уграђене директиве је директива email, којом се постиже валидација електронске поште.Имена директива могу да се нађу тако што се тражи кључна реч “validator” у званичној документацији: https://angular.io/api?type=directive. Све означено словом Д је директива и може се користити у коду.

Ангулар додаје валидационе класе елементима у приказу. Овиме се омогућује стилизовање елмената у зависности од њихове валидности. Неке од валидационих класа које се додељују елементима су “ng-invalid”, “ng-touched” i “ ng-valid”.

Приказивање валидационих порука:

Како би се омогућило приказивање валидационих порука , потребно је прво додати локалну референцу на елемент, а затим је повезати са ngModel-ом. Након овога, ова локална референца може да се користи унутар директиве ngIf како би се приступило форми и проверило њено стање. У наредном примеру приказана је валидација помоћу локалне референце #email.

```
<span class="help-block" ngIf = "!email.isValid && email.touched" >
```

5.9.4 Реактивни приступ

За разлику од шаблонског приступа, где се прослеђивање форме у потпуности врши из прегледа, код реактивног приступа се форма дефинише, иницијализује и прослеђује из кода.

Поступак креирања форме код реактивног приступа:

1. Увести ReactiveFormsModule модул у главни модул апликације
2. Креирати форму у приказу, односно направити жељени распоред контролних поља у оквиру <form> </form> тагова
3. Декларисати атрибут који ће представљати форму. Овај атрибут треба да буде типа FormGroup, а иницијализује се приликом иницијализације компоненте. FormGroup се увози из @angular/forms библиотеке. signupForm: FormGroup;
4. Унутар ngOnInit() метода, иницијализовати атрибут форме.

Конструктору FormGroup објекта прослеђује се JCOH објекат који садржи елементе форме. Елемент форме се креира тако што се, као кључ проследи име параметра, односно елемента форме, а као вредност се прослеђује објекат типа FormControl. Конструктору овог објекта могу да се проследе три атрибута.

- Први атрибут је иницијална вредност. Овај атрибут је обавезан, али уколико не желимо да елемент форме има иницијалну вредност, прослеђујемо нулл објекат.
- Други атрибут је низ валидатора. Овај атрибут није обавезан и прослеђује се само уколико желимо да додамо неку валидацију форми. Трећи атрибут је низ асинхроних валидатора. Овај атрибут такође није

обавезан и прослеђује се само уколико желимо да имамо асинхрону валидацију.

```
ngOnInit(){
this.signupForm = new FormGroup({
  'username' : new FormControl(nul)),
  'email': new FormControl(null),
   'gender': new FormControl('female')
 });
}
```

Повезивање атрибута форме са формом у прегледу:

1. директиву formGroup елемента form повезати са атрибутом који садржи форму у коду

```
<form [formGroup] = “signupForm”></form>
```

2. сваки контролни елемент повезати са одређеним елементом форме користећи директиву formControlName

```
<input id=”username” formControlName=”username” />
```

Прослеђивање форме:

Као и код шаблонског приступа, повезати директиву ngSubmit са извршавањем неког метода у класи компоненте. Главна разлика је у томе што се, код реактивног приступа, форми приступа директно из кода јер је она поље у класи, а не коришћењем локалне референце као код шаблонског приступа.

```
<form (ngSubmit) =”nekiSubmitMetod()”>
```

Уколико желимо да додамо валидацију, треба да проследимо додатне аргументе конструктору FormControl објекта. Постоје валидатори које нам обезбеђује Ангулар, али поред тога постоји могућност креирања сопствених валидатора. Ангулар валидатори се налазе унутар Validators класе и приступа им се позивањем Validators.imeValidatora. Може се проследити један или више валидатора.

```
New FormControl(null, Validators.required);
New FormControl(null, [Validators.required, Validators.email]);
```

Како би се приступило елементима форме, не само из кода, већ и из шаблона, користи се метод гет(енг. get), са именом елемета као атрибутом, над пољем које представља форму.

```
<span *ngIf=”signupForm.get(‘username’).valid  &&
signupForm.get(‘username’).touched”/>
```

На овај начин омогућује се провера валидности, али и приступ елементима форме за неке друге потребе.

5.10 Хттп комуникација (енг. Http comunication)

Хттп комуникација омогућава међусобну интеракцију између сервиса на вебу. Ова комуникација се заснива на захтев-одговор систему (енг. Request-Response system) коришћењем Хттп протокола. Овај протокол омогућава да клијент пошаље захтев удаљеном серверу и прими одговор од сервера. Сервер, по добијању захтева, обради тај захтев, креира одговор у зависности од извршења операције, и проследи одговор назад до клијента.

Код Ангулар платформе, ова комуникација се обавља користећи Хттп модул (енг. Http module) и компоненту Http.

Хттп комуникација у Ангулару се омогућава на следећи начин:

1. Додати HttpModule у секцију import унутар главног модула

```
imports: [
    BrowserModule,
    FormsModule,
    BrowserModule,
    AppRoutingModule,
    HttpModule,
    ReactiveFormsModule
  ],
```

2. Унутар конструктора компоненте која шаље хттп захтев, увести Http компоненту

```
constructor(private http:Http){}
```

3. Проследити захтев тако што се над објектом хттп компоненте, позове одређени метод. Неки од метода су get, post, put, update и delete. На примеру је показан позив метода put који омогућава слање објекта у оквиру захтева. Резултат извршавања је објекат посматрача (енг. Observable).

```
return this.http.put("https://library-b739f.firebaseio.com/books.json",
this.getBooks());
```

4. Како би се приступило одговору који враћа сервер, морамо да се претплатимо (енг. Subscribe) на објекат посматрача враћен у претходном методу.

```
this.storeBooksToDb().subscribe(
            (response :Response) => {
                console.log(response);
            }
        );
```

Објекат који се добија као одговор од стране хттп метода, омогућа приступ одговору од сервера, али и потенцијалним грешкама које се дешавају приликом комуникације са сервером.

6 Пројекат

6.1 Апликација Библиотека

Практични пројекат представља апликацију која омогућава основне операције унутар библиотеке.

6.1.1 Опис апликације

Постоје три целине у оквиру апликације, књиге, аутори и корпа. Корисник може да унесе нову књигу, измени постојећу, види листу свих књига и кликом на неку од њих види детаље. Исте операције су могуће и са ауторима.Поред ових операција, могуће је и додати књигу у корпу. Корпа приказује све књиге које су претходно у њу додате и крајњу цену тих књига.

6.1.2 Циљ апликације

Циљ ове апликације је приказ свих претходно описаних Ангулар функционалности кроз пример. Апликација садржи компоненте, сервисе, директиве, хттп комуникацију и сбе остале Ангулар функционалности поменуте у ранијим поглављима.

6.1.3 Структура апликације

Апликација се састоји из четири главне компоненте, а то су: HeaderComponent, RecipeComponent, AuthorComponent и ShoppingCartComponent. Све компоненте су уграђене у главну AppComponent компоненту.

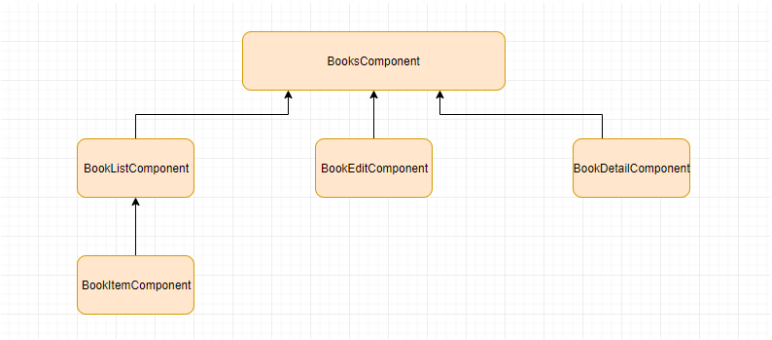
Компонента HeaderComponent представља заглавље. У оквиру ове компоненте нема других компоненти, и ово је једина компонента која је увек видљива. Ова компонента служи за приказ навигације ка другим компонентама у систему.

Slika 2: Приказ HederComponent компоненте

Компонента BookComponent омогућава приказ рецепата и манипулације над њима. Ова компонента у себи садржи компонентеBook, BookEdit и BookDetails. BookList компонента омогућава приказ листе књига, које се у том

Slika 3 : Шема компоненти књиге

тренутку налазе у бази. Кликом на неку од тих књига, приказују се детаљнији подаци књиге. Приказ детаља омогућава компонента BookDetails, а након тога можемо да изменимо књигу кликом на акције, па у оквиру акција кликом на дугме измени. Овиме се окида компонента BookEdit која омогућава измену постојеће компоненте, или додавање нове.На слици 3 приказана је шема компоненти које формирају BooksComponent компоненту.



Компоненте које формирају компонентз аутора имају идентичан распоред као и компоненте које формирају књигу. Дакле, AuthorsComponent се састоји из три компоненте, а то су AuthorEdit, AuthorDetails и AuthorList компонента. AuthorList компонента садржи AuthorItem компоненту која представља једну ставку листе.

Поред ове четири главне компоненте, апликација садржи и једну директиву која омогућује приказ падајуће листе која се приказује кликом на дугме акције унутар компоненти за уређивање књига и аутора. Ова директива је смештена у фајлу dropdown.directive.ts и назива се DropdownDirective.

У остатку структуре, налазе се модели аутора и књиге, који представљају објекте који се смештају у базу.

Апликација такође садржи и сервисе. Сервиси који су коришћени су:

- BooksService - омогућује рад са књигама
- AuthorsService – омогућује рад са ауторима
- ShoppingCartService –омогућује рад са корпом

6.2 Покретање апликације

Како би се апликација покренула потребно је да на систему буде инсталиран node.js. Апликација користи @angular/core верзију 6.1.0 и Боотстреп фрејмворк верзије 3.3.7.

Покретање се врши тако што се, унутар директоријума апликације, у секвенци позову команде:

1. Npm install @angular/core
2. Npm install bootstrap@3.3.7
3. Ng serve

Уколико су све команде успешно извршене, приступа им се на адреси localhost на порту 4200.

6.3 Развијање апликације (енг. Deployment of application)

Како бисмо приказали апликацију на на неком од хостих сервера морамо да одрадимо њен развој (енг. Deployment) на том серверу. У даљем тексту ће бити приказан процес постављања апликације на Фајрбејс(енг. Firebase) хостинг серверу. Овај сервер омогућава бесплатан хостинг апликације.

Процес развоја:

1. Инсталирати фајрбејс алате глобално у систему позивом команде:
   npm install –g firebase-tools
2. Улоговати се на фајрбејс систем (претходно треба направити налог и пројекат на адреси https://console.firebase.google.com) командом:
   firebase login
3. Иницијализовати пројекат командом:
   firebase init
4. Након овога треба изгради пројекат у продукционом режиму. Ово се ради покретањем команде:
   ng build –prod.
   Ова команда смешта све потребне фајлове у директоријум /dist пројекта.

5. Финални корак је само развијање апликације командом:
   firebase deploy
   Уколико се ова команда успешно изврши као повратни резултат добијамо адресу на којој се налази апликација.

7 Закључак

Одлука о томе који реактивни фронтенд оквир односно коју платформу треба користити може бити јако исцрљујућ процес. Постоји доста за и против разлога за сваки од њих. Ангулар платформа омогућује реактиван приступ уз веома добро одрађен интерфејс командне линије који у многоме аутоматизује рад. Међутим, прво питање које се јавља приликом одлучивања о овој платформи је да ли желимо да имамо једностране апликацију. Поред овог, општег услова, у обзир се морају узети и променљиве попут циљане клијентске групе, платформе која се користи за развој, али се такође поставља и питање разумевања ТајпСкрипт језика будући да је цела платформа заснована на њему. У сваком случају, сличне, реактивне, технологије не стоје међусобно једна испред друге. Одлука се углавном доноси на основу експертизе у оквиру тима. Неке од предности које нам Ангулар платформа пружа су јасно дефинисан концепт објектно оријентисаног програмирања и веома добра документација, односно могућност долажења до знања о самој платформи.

8 Литература

[1]. Nate Murray, Felipe Coury, AriLerner, Carlos Taborda, Ng-book, *The Complete Book on Angular 5*

[2]. *Angular official*, https://angular.io/

[3]. M.Schwarzmüller, *Angular 2 – The Complete Guide*, https://www.udemy.com/the-complete-guide-to-angular-2

[4]. *Angular vs React vs Vue comparison*, https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

[5]. *Angular W3Schools Course*, https://www.w3schools.com/angular