

ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА
СТРУКОВНИХ СТУДИЈА

Игор Поповић

ВЕБ АПЛИКАЦИЈА „ВОДИЧ ЗА ANGULAR“

- завршни рад -



Београд, Септембар 2024.

Кандидат: **Игор Поповић**

Број индекса: **НРТ-50/18**

Студијски програм: **Нове Рачунарске Технологије**

Тема: **Веб апликација „Водич за Angular“**

Основни задаци:

- 1. Опис технологија**
- 2. Опис корисничког интерфејса и начина израде апликације**
- 3. Водич за Angular**

Предмет из кога се ради завршни рад:

Интернет програмирање

Ментор:

Београд, септембар 2024.године.

др Зоран Ћировић, проф. ВИШЕР

РЕЗИМЕ:

У раду је представљена веб апликације за учење Angular-а на српском језику. Angular је коришћен као основни framework за развој апликације, омогућавајући брз и ефикасан рад са компонентама и модуларизацију кода. Кориснички интерфејс је дизајниран коришћењем Angular Material библиотеке, која пружа модеран и конзистентан изглед апликације, док су CSS и друге библиотеке коришћени за додатно стилизовање и функционалност. За писање логике апликације и обраду података коришћен је TypeScript, који омогућава статичко типизирање и бољу структуру кода, чинећи апликацију скалабилном и лаким за одржавање.

Кључне речи: *Angular, Angular Material, CSS, TypeScript, веб апликација, учење програмирања.*

ABSTRACT:

The paper presents the web application for learning Angular in Serbian. Angular was used as the primary framework for the development of the application, enabling fast and efficient component-based development and code modularization. The user interface was designed using the Angular Material library, which provides a modern and consistent look to the application, while CSS and other libraries were used for additional styling and functionality. TypeScript was utilized for writing the application logic and data handling, offering static typing and better code structure, making the application scalable and easy to maintain.

Key words: *Angular, Angular Material, CSS, TypeScript, web application, programming learning.*

САДРЖАЈ

Садржај.....	4
1. Увод	6
2. Технологије	7
2.1. Angular	7
2.2. TypeScript	7
2.3. HTML & CSS	7
2.4. Angular Material	7
2.5. Додатне библиотеке	8
2.6. Node.js и TypeScript компајлер	8
3. Опис корисничког интерфејса и начина израде апликације	9
3.1. Структура Пројекта	9
3.2. Кориснички интерфејс	10
3.2.1. Насловна страна	10
3.2.2. Навигациони мени	11
3.2.3. Контакт форма	15
3.2.4. Приказ кода	16
3.2.5. Табеле и друге компоненте	18
3.2.6. Angular IDE	20
3.2.7. Футер	20
3.3. Начин израде апликације	21
3.3.1. Алати за развој	21
3.3.2. Архитектура апликације	21
3.3.3. Компонентна архитектура	22
4. Водич за Angular	26
4.1. Увод	26
4.2. Упознавање са Angular технологијом	29
4.2.1. Компоненте	29
4.2.2. Шаблони	32
4.2.3. Директиве	33
4.2.4. Убризгавање зависности	35
4.3. Највоља пракса	38
4.3.1. Сигурност	38
4.3.2. Приступачност	40

4.3.3. Учитавање по потреби	40
4.4. Водич за програмере	43
4.4.1. Детектовање промена	43
4.4.2. Рутирање и навигација	44
4.4.3. Форме	46
4.4.4. HttpClient.....	49
4.4.5. Рендеровање на серверској страни.....	52
4.4.6. Статички генерисани сајтови	53
4.4.7. Тестирање	53
4.4.8. Angular алатке (Dev Tools)	56
5. Закључак.....	57
6. Индекс појмова	58
7. Литература	59
8. Изјава о академској честитости	60

1. УВОД

Апликација развијена као део овог завршног рада има за циљ да омогући лакше и брже учење *Angular* алата на српском језику, кроз интерактивно и визуелно привлачно окружење. Ова веб апликација је дизајнирана да омогући корисницима да уче *Angular* концепте на начин који је приступачан и разумљив, уз подршку за практичне примере.

Апликација садржи секције за преглед теорије, интерактивне компоненте, као и примере имплементације кода. Поред тога, апликација у себи садржи и развојно окружење у ком корисници могу одмах да испробају свој код.

Апликација је реализована помоћу следећих технологија:

- *Angular*
- *TypeScript*
- *CSS*
- *HTML*
- *Angular Material*
- Друге библиотеке и алати за подршку функцијама и стилизовању

Архитектура апликације је заснована на компонентном моделу који је специфичан за *Angular*, где су различити делови апликације развијени као засебне компоненте које комуницирају међусобно преко сервиса и модула. Овакав приступ омогућава лакше одржавање, проширивање и тестирање апликације.

У наредним одељцима биће детаљније објашњене технологије коришћене у развоју, као и процес имплементације и дизајна корисничког интерфејса.

2. ТЕХНОЛОГИЈЕ

Технологије коришћене за креирање ове апликације обухватају *Angular*, *TypeScript*, *CSS*, као и разне библиотеке и алате који су интегрисани у апликацију. У овом подглављу биће укратко објашњен *Angular*, док ће остале технологије и алати бити детаљније описани.

2.1. ANGULAR

Angular је модерни веб фрејмворк (*framework*) отвореног кода који омогућава развој динамичких и реактивних једностраних апликација (*SPA*). Овај фрејмворк (*framework*) је изабран због своје модуларности, подршке за компонентно засновану архитектуру, као и због своје снажне заједнице и алатки које подржавају развој. *Angular* верзија која је коришћена у овој апликацији је 16.2.0, а апликација такође укључује и *Angular Material* за дизајн корисничког интерфејса. Детаљнија објашњења за *Angular* ће бити представљена у наредним одељцима рада.

2.2. TYPESCRIPT

TypeScript је надсет језик за *JavaScript* који додаје статичко типизирање и објектно-оријентисане карактеристике. *TypeScript* верзија 5.1.3 је коришћена за развој ове апликације, јер омогућава већу сигурност у коду и поједностављује рад са великим кодним базама. Коришћење *TypeScript* језика омогућава бољу организацију кода и лакше одржавање апликације.

2.3. HTML & CSS

HTML (*Hyper Text Markup Language*) је основни језик за креирање веб страница и структурање њиховог садржаја. *HTML* је коришћен у овој апликацији за дефинисање структуре и основних елемената странице, као што су наслови, параграфи, обрасци и други садржаји који чине основу корисничког интерфејса.

Док *CSS* (*Cascading Style Sheets*) је коришћен за стилизовање корисничког интерфејса апликације. Овај језик омогућава дефинисање визуелних аспеката апликације као што су боје, фонт, размештај елемената и друго. У комбинацији са *Angular Material*, *CSS* је коришћен за додатно прилагођавање изгледа и осећаја апликације, чиме је постигнут модеран и прилагођен дизајн.

2.4. ANGULAR MATERIAL

Angular Material је свеобухватна библиотека компоненти за *Angular*, осмишљена да олакша креирање визуелно привлачних и конзистентних корисничких интерфејса у складу са принципима *Material Design*, које је развио *Google*. Ова библиотека нуди велики број унапред дизајнираних компоненти које су у потпуности интегрисане са

Angular фрејмворком (*framework*), што омогућава брзу имплементацију без потребе за ручним креирањем основних *UI* елемената.

Међу најзначајнијим компонентама које *Angular Material* пружа су дугмад, обрасци, дијалози, табеле, менији, навигациони панели, иконице, алати за визуелизацију података и још много тога. Ове компоненте су прилагођене за *Angular* и подржавају реактивно програмирање, што омогућава динамичку интеракцију са корисницима апликације. Све компоненте су дизајниране тако да буду одговорне и оптимизоване за различите уређаје и резолуције екрана.

Употреба *Angular Material*, знатно је успела да убрза процес развоја, јер су многи визуелни и функционални елементи већ били доступни као део библиотеке, што је смањило потребу за писањем великог броја CSS и HTML кода. Верзија која је коришћена у овој апликацији је 16.2.2, која укључује најновије ажуриране компоненте и побољшања у складу са тренутним стандардима за *Material Design*. Ова верзија је одабрана због своје стабилности и компатибилности са верзијом коришћеном у *Angular* пројекту.

2.5. ДОДАТНЕ БИБЛИОТЕКЕ

Уз *Angular* и *Angular Material*, коришћене су и бројне друге библиотеке које су омогућиле додатну функционалност у апликацији. Неке од ових библиотека укључују:

- *FontAwesome*: Библиотека икона која је коришћена за додавање икона у кориснички интерфејс апликације.
- *Ng2 PDF Viewer*: Библиотека која омогућава преглед *PDF* докумената директно у апликацији.
- *Ngx Clipboard*: Библиотека која омогућава лако копирање текстуалног садржаја у клипборд.
- *Ngx Highlight.js*: Библиотека за синтаксно истицање кода у апликацији.

2.6. NODE.JS И TYPESCRIPT КОМПАЈЛЕР

За изградњу и развој апликације коришћена је верзија *Node.js* 18.19.0 у комбинацији са *TypeScript* компајлером верзије 5.1.3. *Node.js* је омогућио покретање развојног сервера и компилацију *TypeScript* кода у *JavaScript*, док је *TypeScript* компајлер осигурао да се код преводи у оптимизовани *JavaScript*.

Ове технологије и алати омогућили су развој стабилне и модуларне *Angular* апликације која ће корисницима пружити квалитетно искуство у учењу.

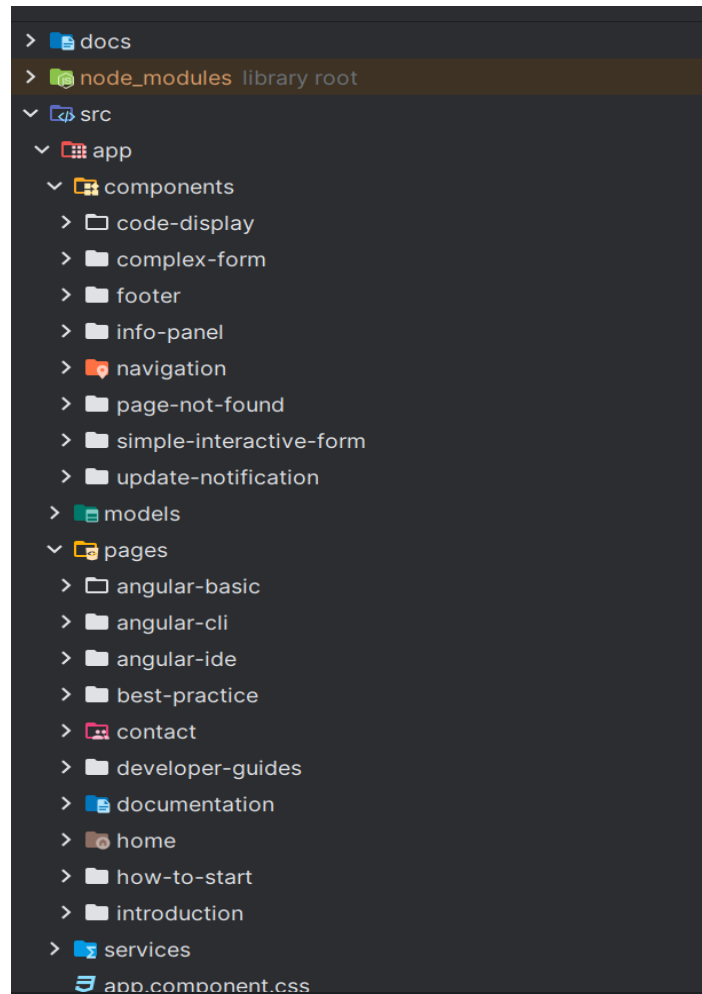
3. ОПИС КОРИСНИЧКОГ ИНТЕРФЕЈСА И НАЧИНА ИЗРАДЕ АПЛИКАЦИЈЕ

Апликација је развијена са циљем да омогући корисницима интерактивно окружење у коме могу да стекну практична знања о *Angular* фрејмворку (*framework*). Апликација је дизајнирана тако да буде пријатна за коришћење, визуелно атрактивна, и интуитивна, како би корисници могли да се фокусирају на учење без ометања. У наставку ће бити детаљно описана структура пројекта, као и све битне компоненте корисничког интерфејса.

3.1. СТРУКТУРА ПРОЈЕКТА

Структура пројекта је организована тако да обезбеди лаку навигацију кроз код и омогући једноставно одржавање и проширивање апликације. Пројекат је подељен у неколико главних директоријума, који се налазе унутар *src/app* директоријума:

- *components*: Садржи све глобалне компоненте које се користе широм апликације. Ове компоненте укључују *navigation*, *footer*, *info-panel*, *code-display*, *page-not-found* и друге. Свака компонента је осмишљена тако да буде максимално за виšekратну употребу и лако интегрисана у различите делове апликације. На пример, *navigation* компонента служи за приказ главног менија на свим страницама, док *footer* обезбеђује конзистентан приказ подножја апликације.
- *models*: У овом фолдеру налази ENUM фајл и модели који дефинишу структуру података коришћених у апликацији. Ово укључује типове података, интерфејсе и енуме који омогућавају бољу организацију и типизирање у *TypeScript* језику. Модели су дизајнирани тако да обезбеде структуру података која је лако разумљива и употребљива широм апликације.
- *pages*: Садржи све појединачне странице апликације. Свака страница има свој засебан фолдер у којем се налазе потребне компоненте и шаблони. На пример, странице као што су *angular-basic*, *angular-ide*, *documentation*, и *contact* се налазе у овом директоријуму. Ове странице су структуриране тако да пружају корисницима све потребне информације и примере за учење Angular фрејмворка (*framework*), са фокусом на организован и прегледан приказ.
- *services*: Садржи све сервисе који управљају логиком пословања и комуникацијом између различитих делова апликације. Сервиси су коришћени за руковање подацима, обраду података, и омогућавање комуникације са спољним *API* сервисима ако је потребно. Ова архитектура омогућава лако тестирање и одржавање, као и могућност проширења функционалности.



Слика 3.1. – Структура пројекта

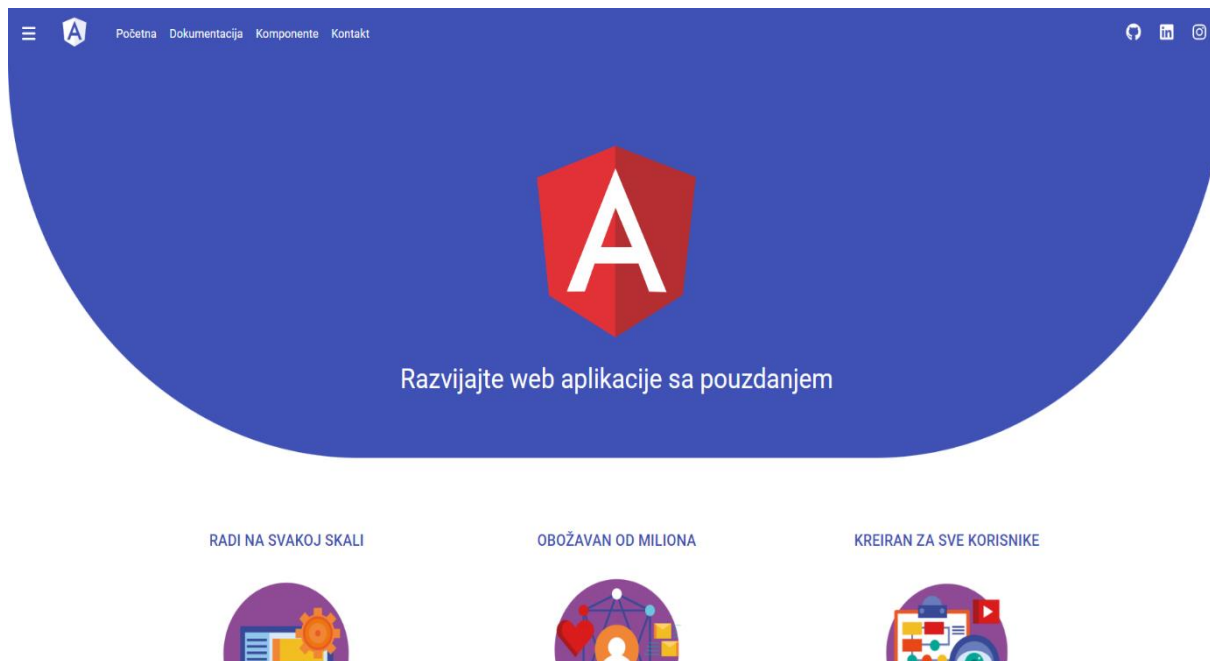
3.2. КОРИСНИЧКИ ИНТЕРФЕЈС

Кориснички интерфејс ове апликације је пажљиво дизајниран како би корисницима пружио интуитивно и пријатно искуство током употребе. Сви елементи интерфејса су организовани тако да омогуће лаку навигацију и приступ свим важним информацијама и функционалностима. Дизајн је минималистички, али визуелно атрактиван, са нагласком на употребљивост и приступачност.

3.2.1. Насловна страна

Дизајн странице је минималистички, али визуелно привлачан, са доминантном *Angular* црвеном и плавом бојом. Централни елемент је велики *Angular* лого са мотивационом поруком која позива кориснике да започну учење. Испод овог елемента, налазе се иконице које визуелно представљају главне карактеристике платформе, уз кратак опис сваке од њих.

Циљ наславне стране је да одмах привуче пажњу корисника и понуди брзе путеве до најважнијих секција сајта. Пажљиво одабране боје и јасна типографија омогућавају лаку читање и разликовање различитих елемената на страници.



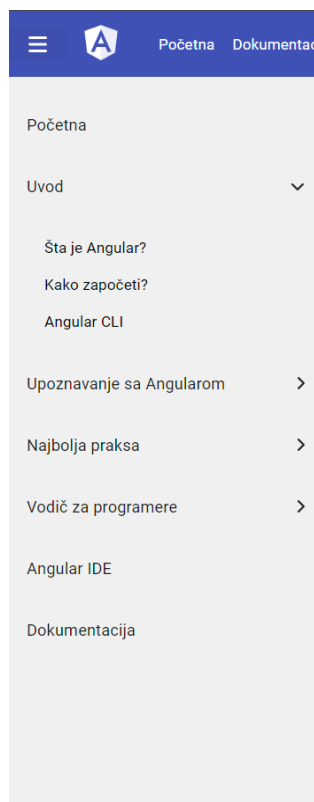
Слика 3.2.1. – Структура пројекта

3.2.2. Навигациони мени

Навигациони мени је кључни елемент апликације који омогућава корисницима да се лако и брзо крећу кроз различите секције сајта. Пажљиво дизајниран, мени је интуитиван и приступачан, како за искусне кориснике, тако и за оне који тек почињу са учењем.

У горњем левом углу апликације налазе се три основна линка који су видљиви на свим страницама: „Почетна“, „Документација“, „Компоненте“ и „Контакт“. Линк "Почетна" враћа корисника на почетну страницу апликације, "Документација" води до странице у којој се налази *PDF* о овом завршном раду, "Компоненте" воде до странице која објашњава шта су *Angular* компоненте, с обзиром да је то једна од најчешћих тема које се претражују и „Контакт“ води до странице у којој се налази форма за контактирање тима. Ови линкови су стално присутни у навигацији и омогућавају корисницима брз приступ најважнијим одељцима сајта.

На левој страни навигационе траке налази се "хамбургер" мени, представљен као три хоризонталне линије. Када корисник кликне на овај мени, отвара се бочна навигациона табла која садржи све странице апликације. Ово решење омогућава компактан приказ менија, који не заузима много простора на екрану, али пружа приступ свим функционалностима апликације.



Слика 3.2.2. – Навигациони мени

Мени који се приказује након клика на хамбургер икону садржи следеће секције и потсекције:

- Почетна:
 - Главна страница апликације која уводи кориснике у функције и могућности сајта.
- Увод:
 - Шта је *Angular*?: Објашњава основне концепте и сврху *Angular* фрејмворка (*framework*).
 - Како започети?: Упутства за покретање и основне кораке при раду са *Angular* фрејмворком (*framework*).
 - *Angular CLI*: Детаљи о командној линији *Angular*-а, укључујући како се користи за креирање пројеката и компоненти.

- Упознавање са Angular-ом:
 - Компоненте: Преглед Angular компоненти, како се креирају и користе у апликацији.
 - Шаблони (*Templates*): Објашњава како се креирају и користе шаблони.
 - Директиве: Информације о *Angular* директивама и њиховој улози у развоју апликација.
 - Убризгавање зависности (*Dependency injection*): Објашњава концепт *dependency injection* и како се он користи за управљање зависностима.
- Најбоља пракса:
 - Сигурност: Препоруке за обезбеђивање сигурности у *Angular* апликацијама.
 - Приступачност: Савети и технике за побољшање приступачности веб апликација.
 - *Lazy Loading*: Објашњава како се имплементира *lazy loading* за боље перформансе апликације.
- Водич за програмере:
 - Увод: Основне информације за програмере који желе да уче *Angular*.
 - Детектовање промена: Како *Angular* управља променама и како се користи овај механизам.
 - Рутирање и навигација: Детаљи о рутирању у *Angular* апликацијама и како се креирају сложене навигационе структуре.
 - Форме: Како креирати и управљати формама, укључујући реактивне форме.
 - *HTTP* клијент: Коришћење *Angular HTTP* клијента за комуникацију са серверима.
 - Рендеровање на серверској страни: Објашњава концепт *SSR (Server-Side Rendering)*.
 - Статички генерисани сајтови (*SSG*): Како *Angular* може да се користи за креирање статички генерисаних сајтова.
 - Тестирање: Упутства за тестирање *Angular* апликација, укључујући јединичне и интеграционе тестове.
 - *Angular* Алатке (*Dev Tools*): Преглед алата за дебаговање и оптимизацију.

- Angular IDE:
 - Интегрисани *Angular IDE* који омогућава корисницима да пишу, тестирају и дебагују код директно у апликацији.
- Документација:
 - PDF фајл који садржи читав завршни рад у себи.

Целокупан дизајн навигационог менија је осмишљен са циљем да пружи корисницима максималну приступачност и лакоћу коришћења. Боје су усклађене са основним бојама апликације, где доминира *Angular* плава боја са белим текстом, што обезбеђује добру читљивост и визуелни контраст.

Мени је прилагођен за рад на различитим уређајима, укључујући десктоп рачунаре, лаптопове, таблете и мобилне телефоне. Овај фокус на приступачност значи да ће се мени правилно приказивати и бити лако употребљив на било ком уређају, чиме се осигурава конзистентно корисничко искуство. Сви елементи менија су довољно велики и лако видљиви, што олакшава навигацију чак и на мањим екранима.

3.2.3. Контакт форма

Форма се састоји од три основна поља: име, емаил адреса и порука, као и додатног чекбокса (*checkbox*) за потврду сагласности да корисници могу бити контактирани путем е-поште. Ово је важан корак како би се осигурала приватност и безбедност корисника, поштујући *GDPR* смернице и осигуравајући да се лични подаци обрађују у складу са важећим регулативама.

Форма је у потпуности стилизована помоћу *Angular Material* компоненти, што доприноси елегантном и професионалном изгледу, усклађеном са остатком апликације. Боје су одабране тако да обезбеде добар визуелни контраст и читљивост, што је нарочито важно код обавезних поља која су јасно означена како би се избегле грешке приликом уноса података.

Слика 3.2.3. – Контакт форма

Контакт форма функционише преко интеграције са *EmailJs*, што омогућава слање порука директно са фронтенда без потребе за серверском страном. Ова интеграција је веома једноставна за коришћење и поуздана, а омогућава слободу у прилагођавању порука и формата е-поште. Када корисник попуни сва поља и кликне на дугме „Пошаљи“, подаци се шаљу путем *EmailJs* сервиса на унапред дефинисану адресу е-поште. Овај приступ омогућава брзу и ефикасну комуникацију, осигуравајући да све поруке стигну до одговорних лица, што побољшава корисничко искуство и подршку.

Да би корисницима пружила повратну информацију о успеху слања поруке, апликација користи *MatSnackBar* компоненту из *Angular Material*. Након успешног слања поруке, појављује се *snackbar* са поруком „Порука успешно послата!“, који остаје видљив три секунде пре него што се аутоматски затвори. У случају да дође до грешке приликом слања поруке, појављује се *snackbar* са поруком „Дошло је до грешке приликом слања поруке!“. Ове функционалности су имплементиране на следећи начин:

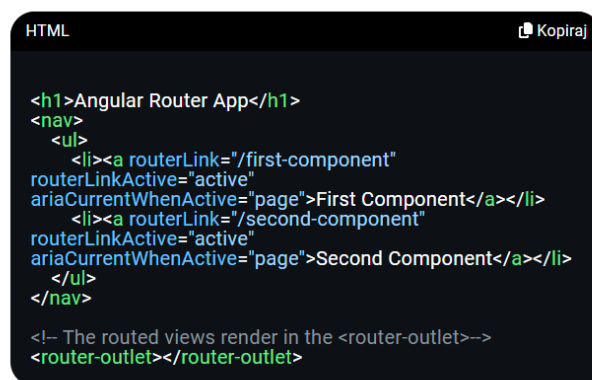


Слика 3.2.3. – Код за слање е-поште

Овај приступ осигурава да корисници буду информисани о резултату својих акција, било да је порука успешно послата или је дошло до неке грешке. Ово побољшава транспарентност и корисничко искуство, омогућавајући корисницима да благовремено реагују у случају проблема.

3.2.4. Приказ кода

На страницама као што су „Форме“ и „Рутирање и навигација“, компонента *code-display* се користи за јасан и читљив приказ примера кода.



Слика 3.2.4. – Приказ кода

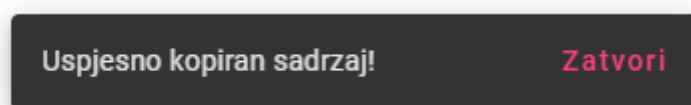
Ова компонента омогућава корисницима да прегледају код који је релевантан за одређене теме и концепте. Код је приказан са синтаксним истицањем, што повећава прегледност и олакшава разумевање.

Поред тога, корисници имају могућност да брзо копирају приказани код кликом на дугме „Копирај“, што је посебно корисно при учењу и брзом имплементирању решења у своје пројекте. Овај приказ кода је један од кључних елемената апликације који корисницима пружа практичну вредност, омогућавајући им да директно примене научене концепте.

Као и код контакт форме, када корисник копира код, појављује се `snackbar` који обавештава да је садржај успешно копиран. Ова функционалност је имплементирана на следећи начин:



Слика 3.2.4. – Код за копирање садржаја



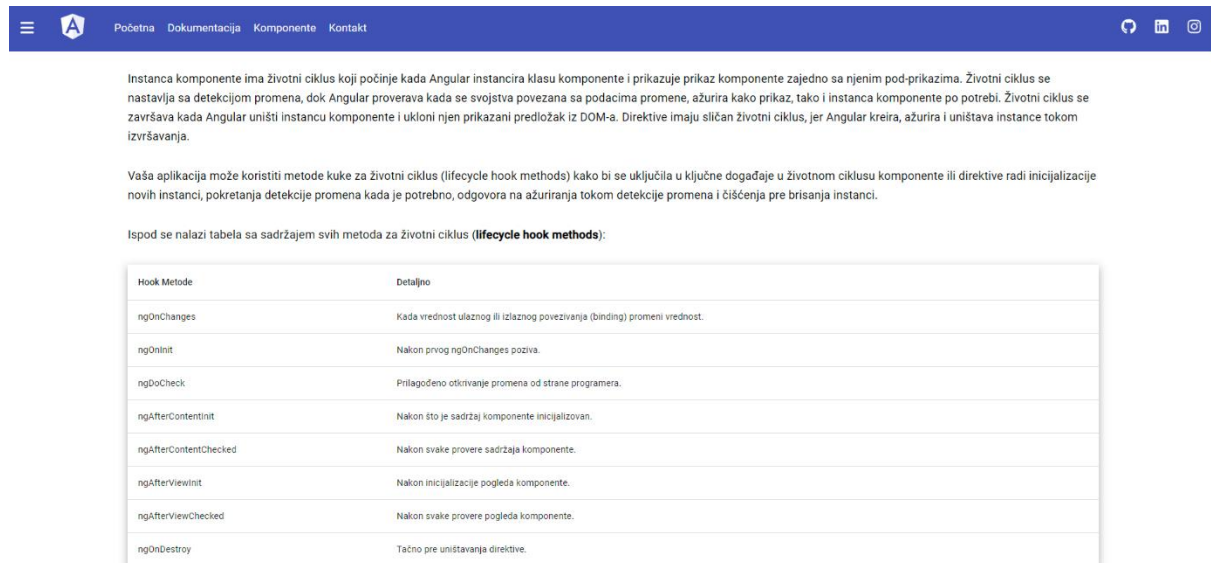
Слика 3.2.4. – `Snackbar` за приказ поруке

Овај `snackbar` обавештава корисника да је код успешно копиран у клипборд (*clipboard*), чиме се додатно побољшава корисничко искуство и обезбеђује јасна информација о резултату акције. Ово је нарочито корисно у ситуацијама када корисник брзо прелази са једног задатка на други и жели да буде сигуран да је садржај исправно сачуван.

3.2.5. Табеле и друге компоненте

На страницама као што су „Компоненте“ и „Како започети?“, табеле се користе за приказивање сложених информација у јасном и структурираном формату.

Ове табеле омогућавају корисницима да лако пореде податке и анализирају различите аспекте апликације. Свака табела је пажљиво дизајнирана у складу са *Angular Material* смерницама, што доприноси конзистентном и естетски пријатном изгледу целокупног интерфејса.



Instanca komponente ima životni ciklus koji počinje kada Angular instancira klasu komponente i prikazuje prikaz komponente zajedno sa njenim pod-prikazima. Životni ciklus se nastavlja sa detekcijom promena, dok Angular proverava kada se svojstva povezana sa podacima promene, ažurira kako prikaz, tako i instanca komponente po potrebi. Životni ciklus se završava kada Angular uništi instancu komponente i ukloni njen prikazani predložak iz DOM-a. Direktive imaju sličan životni ciklus, jer Angular kreira, ažurira i uništava instance tokom izvršavanja.

Vaša aplikacija može koristiti metode kuke za životni ciklus (lifecycle hook methods) kako bi se uključila u ključne događaje u životnom ciklusu komponente ili direktive radi inicijalizacije novih instanci, pokretanja detekcije promena kada je potrebno, odgovora na ažuriranja tokom detekcije promena i čišćenja pre brisanja instanci.

Ispod se nalazi tabela sa sadržajem svih metoda za životni ciklus (**lifecycle hook methods**):

Hook Metode	Detaljno
ngOnChanges	Kada vrednost ulaznog ili izlaznog povezivanja (binding) promeni vrednost.
ngOnInit	Nakon prvog ngOnChanges poziva.
ngDoCheck	Prilagođeno otkrivanje promena od strane programera.
ngAfterContentInit	Nakon što je sadržaj komponente inicijalizovan.
ngAfterContentChecked	Nakon svake provere sadržaja komponente.
ngAfterViewInit	Nakon inicijalizacije pogleda komponente.
ngAfterViewChecked	Nakon svake provere pogleda komponente.
ngOnDestroy	Tačno pre uništavanja direktive.

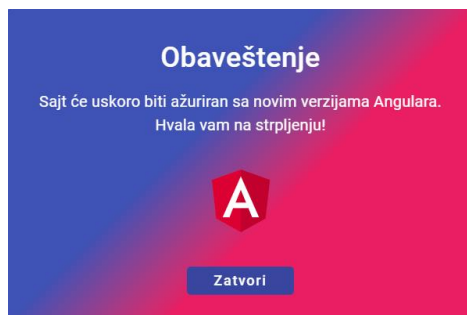
Слика 3.2.4. – Табела функција за животно циклус компоненте

Поред табела, апликација укључује и низ других компоненти које унапређују корисничко искуство и функционалност. Једна од тих компоненти је *complex-forma*, која представља сложенији образац са којим корисници могу да остваре интеракцију на различите начине.

Complex-forma се састоји од више поља за унос података, укључујући име, презиме, године, количину производа и цену производа. Свака од ових ставки има специфичну валидацију како би се обезбедила тачност и интегритет унетих података. На пример, поља за унос имена и презимена морају садржати најмање два карактера, док поља за године, количину и цену морају садржати вредности веће од нуле. Ова валидација је кључна за осигурање исправности уноса, чиме се смањује могућност грешака и побољшава корисничко искуство.

Слика 3.2.5. – Сложена форма

Још једна значајна компонента је *update-notification*, која служи за обавештавање корисника о предстојећим ажурирањима апликације. Ова компонента је дизајнирана тако да се појављује на видљивом месту и информише кориснике о важним изменама или надоградњама које ће ускоро бити примењене. На овај начин, корисници могу бити правовремено обавештени о променама које би могле директно да утичу на њихово коришћење апликације.



Слика 3.2.5. – Диалог за обавештење о новим изменама

Такође, ту је и *info panel*, који садржи важне напомене и информације које корисници треба да прочитају. Овај панел се користи за приказивање критичних или корисних обавештења, савета или подсетника који су од великог значаја за рад са апликацијом. *Info panel* је дизајниран тако да привуче пажњу корисника, осигуравајући да важне информације не прођу непримећено.

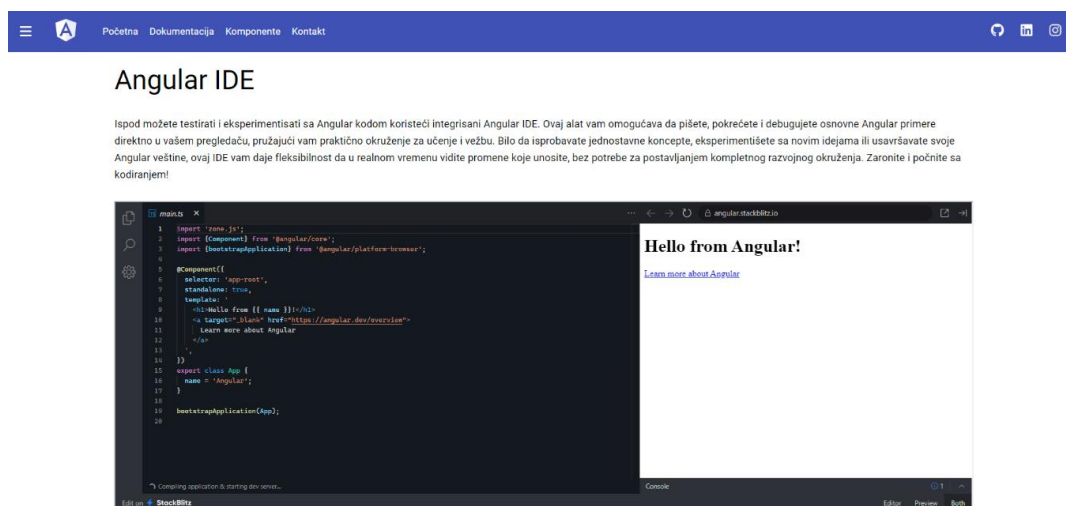
Da bi se eliminisao rizik od napada ubacivanjem skripti, Angular ne podržava korišćenje `<script>` elementa unutar svojih šablona. Angular jednostavno ignoriše `<script>` oznaku i upozorava korisnika putem konzole pregledača. Više informacija o sigurnosti možete pronaći na stranici posvećenoj sigurnosti.

[Prikaži više](#)

Слика 3.2.5. – Диалог за обавештење о новим изменама

3.2.6. Angular IDE

Страница посвећена *Angular IDE*, омогућава корисницима да пишу, покрећу и дебагују *Angular* код директно у претраживачу. Овај интегрисани алат је од велике користи за програмере који желе брзо да експериментишу са различитим концептима и виде резултате у реалном времену. *Angular IDE* пружа интуитивно и флексибилно окружење које омогућава брзу проверу измена кода, без потребе за постављањем комплетног развојног окружења.



Слика 3.2.6. – *Angular IDE* страница

Овај алат је посебно погодан за брзе тестове и прототипове, као и за оне који су нови и желе да стекну практична знања на једноставан и ефикасан начин. Корисници могу да измене код у интегрисаном едитору и одмах виде промене на страници, што значајно убрзава процес учења и развоја.

3.2.7. Футер

Футер апликације налази се на дну сваке странице и служи као извор додатних информација за кориснике. Он садржи контакт податке, линкове до важних секција апликације, као и друге корисне информације које могу бити потребне корисницима. Дизајн футера је једноставан и прегледан, а његова структура и садржај су усклађени са осталим деловима апликације, чиме се обезбеђује конзистентност у изгледу и коришћењу сајта.



Слика 3.2.7. – Футер

Футер такође пружа корисницима осећај завршености странице, са свим потребним информацијама лако доступним на једном месту. Боје и типографија су усклађени са остатком сајта, што обезбеђује визуелну хармонију и доприноси угодном корисничком искуству.

3.3 НАЧИН ИЗРАДЕ АПЛИКАЦИЈЕ

Израда ове апликације захтевала је свеобухватан приступ, од почетног планирања и избора технологија до финалног тестирања. Циљ је био да се креира апликација која не само да испуњава техничке захтеве, већ и пружа корисницима пријатно и интуитивно искуство. Овај одељак покрива кључне аспекте развоја апликације, укључујући употребљене алате, архитектуру и процес развоја.

3.3.1. Алати за развој

За развој апликације коришћен је *WebStorm*, један од најмоћнијих и најсавременијих *IDE* алата доступних на тржишту, специјално дизајниран за веб развој. *WebStorm* пружа низ напредних функција које значајно олакшавају и убрзавају процес развоја апликација. Међу најважнијим функцијама су:

- Аутоматско довршавање кода: *WebStorm* нуди паметно аутоматско довршавање кода које не само да убрзава писање кода, већ и смањује могућност прављења грешака.
- Моћан дебагер (*debugger*): Овај алат омогућава лако отклањање грешака у коду, са подршком за дебаговање у претраживачу и на серверу.
- Интеграција са *Git* системом: *WebStorm* је у потпуности интегрисан са *Git*-ом и другим алатима за контролу верзија, што омогућава једноставно праћење промена у коду и управљање верзијама пројекта.
- Подршка за *Angular*: *WebStorm* је идеалан за развој *Angular* апликација, са уграђеном подршком за *Angular CLI*, *TypeScript* и друге технологије које се користе у *Angular* пројектима.

Захваљујући овим и другим функцијама, *WebStorm* је значајно допринео ефикасности и продуктивности током развоја апликације, омогућавајући фокусирање на креирање функционалности и побољшање корисничког искуства.

3.3.2. Архитектура апликације

Апликација је организована око једног главног модула, названог *AppModule*, који служи као централна тачка за управљање свим аспектима апликације. *AppModule* је одговоран за регистровање свих компоненти, сервиса, и рутирања који су потребни за правилно функционисање апликације.

Све компоненте апликације, укључујући навигациони мени, футер, различите странице и интерактивне елементе, регистроване су у оквиру *AppModule*. На пример,

компоненте попут *NavigationComponent*, *FooterComponent*, *HomeComponent*, и различите странице као што су *IntroductionComponent* и *AngularCliComponent*, све су регистроване у *AppModule*. Ова регистрација омогућава да компоненте буду доступне у целој апликацији и да се могу једноставно користити у било ком делу апликације. Поред компоненти, *AppModule* управља и сервисима који су одговорни за обраду пословне логике и интеракцију са подацима. Ови сервиси су такође регистровани у *AppModule*, чиме се обезбеђује да су доступни свим компонентама које их захтевају. На тај начин, пословна логика и управљање подацима су централно организовани, што поједностављује одржавање и проширивање апликације. *AppModule* такође укључује и друге *Angular* модуле као што су *BrowserModule*, *AppRoutingModule*, *BrowserAnimationsModule*, и разне *Angular Material* модуле (нпр. *MatButtonModule*, *MatSnackBarModule*). Ови модули додају додатну функционалност апликацији, као што су анимације, управљање рутама, и стилизовани кориснички интерфејс. Коришћење ових модула омогућава лаку интеграцију различитих функционалности и осигурава конзистентност у изгледу и функционисању апликације.

Рутирање у апликацији је управљано из *AppModule* путем *Angular Router*, који је конфигурисан у *AppRoutingModule*. Овај модул дефинише све руте у апликацији, укључујући руте до различитих страница као што су *home*, *introduction*, *angular-cli*, *developer-guides*, и *contact*. Ово омогућава корисницима да се лако крећу кроз апликацију, док *Angular Router* аутоматски управља учитавањем одговарајућих компоненти за сваку руту. Овај начин рутирања обезбеђује брзо и једноставно кретање кроз апликацију, што побољшава корисничко искуство.

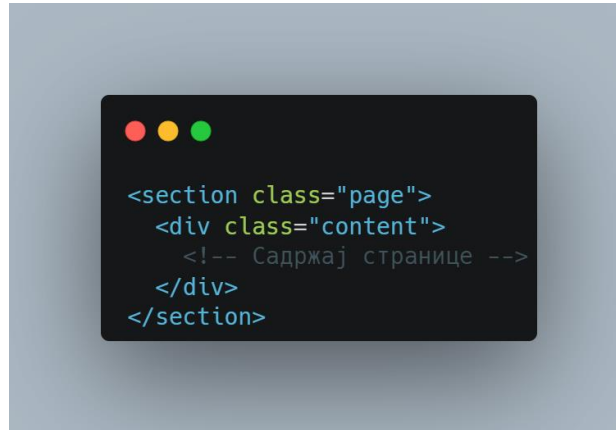
Централизовани приступ у организацији апликације, где је све регистровано у једном модулу, поједностављује структуру апликације и чини је лакшом за развој и одржавање. Све кључне функционалности су организоване и доступне на једном месту, што омогућава програмерима да брже и ефикасније раде, било да додају нове функције или врше измене постојећих. Овај приступ обезбеђује конзистентност у раду апликације и олакшава управљање пројектом у целини.

3.3.3. Компонентна архитектура

У развоју ове апликације, компонентна архитектура је одиграла кључну улогу у организацији и структури пројекта. Компоненте су основни градивни блокови и омогућавају поделу функционалности на мање, независне делове који могу бити лако управљани и поново коришћени. У овом пројекту, све функционалности су подељене на компоненте које су логички груписане у два главна директоријума: *components* и *page*. Ова подела омогућава јасну дистинкцију између основних, реупотребљивих компоненти и појединачних страница апликације, чиме се олакшава одржавање, проширење и организација кода.

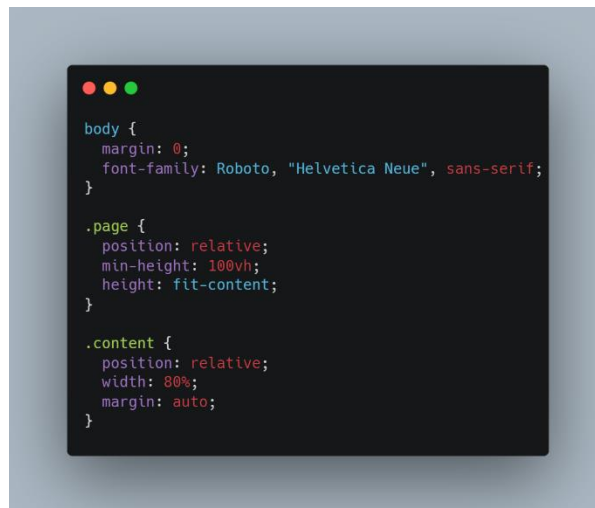
На пример, компонента *NavigationComponent* управља навигационим менијем апликације. Ова компонента је дизајнирана тако да садржи и хоризонтални мени са основним линковима (као што су „Почетна“, „Документација“, „Компоненте“ и „Контакт“), као и „hamburger“ мени који се отвара као бочна табла са детаљним линковима до свих страница апликације. Навигациони мени је имплементиран користећи *Angular Material* и *FontAwesome* иконе, што доприноси модерном и визуелно пријатном изгледу апликације.

Директоријум *pages* садржи компоненте које представљају појединачне странице апликације. Ове компоненте су дизајниране да буду специфичне за сваки одељак апликације, као што су „Почетна“, „Документација“, „*Angular CLI*“, „Форме“, и слично. Свака страница у апликацији има свој посебан шаблон који је дефинисан преко *HTML* и *CSS* датотека. Основни шаблон који се користи за све странице изгледа овако:



Слика 3.3.3. – *HTML* шаблон за *page* компоненте

Класе *page* и *content* су дефинисане у глобалном *CSS* фајлу (*style.css*), чиме се осигурава да све странице апликације прате исти визуелни стил и да су одзивне за различите уређаје. Ово су стилови који дефинишу основни изглед и понашање ових класа:



Слика 3.3.3. – *CSS* шаблон за *page* компоненте

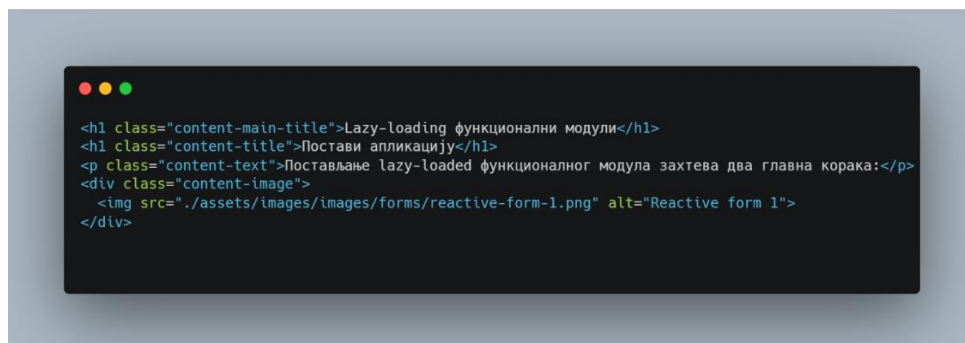
Овај *CSS* код осигурава да су све странице визуелно конзистентне и да се правилно приказују на свим уређајима, без обзира на величину екрана. Поред тога, додатно су дефинисане класе за управљање изгледом главних наслова, поднаслова,

текстова и слика, како би се обезбедило да свака страница изгледа уредно и професионално. Пример CSS кодова за формат главних елемената на страницама:



Слика 3.3.3. – CSS шаблон за приказ текста на page компоненти

Ови стилови осигуравају да су сви текстуални и визуелни елементи на страницама апликације доследни у свом изгледу, што корисницима пружа јасноћу и уједињено корисничко искуство. Пример *HTML* структуре странице која користи ове класе:



Слика 3.3.3. – HTML primer za naslove u tekst u page компонентама

Ови стилови осигуравају да су сви текстуални и визуелни елементи на страницама апликације доследни у свом изгледу, што корисницима пружа јасноћу и уједињено корисничко искуство. Пример *HTML* структуре странице која користи ове класе.

4. ВОДИЧ ЗА ANGULAR

Angular је један од најпопуларнијих и најмоћнијих фрејмворка (*framework*) за развој веб апликација. Развијен и подржан од стране *Google* корпорације, *Angular* омогућава програмерима да креирају динамичне, скалабилне и високо перформансне апликације. Са својом компонентно-базираном архитектуром, *Angular* олакшава развој сложених апликација кроз модуларизацију и вишестроку употребу кода.

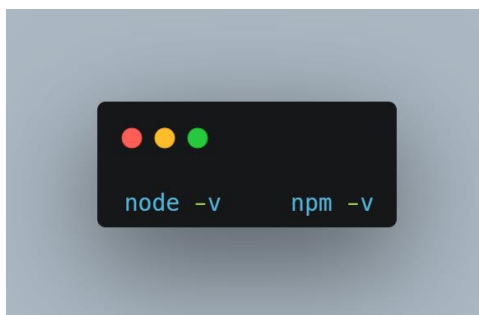
Овај водич је осмишљен да вас кроз корак по корак уведе у основе и напредне концепте, од иницијалне поставке окружења, преко креирања компоненти и рутирања, до управљања стањем и оптимизације апликације. Без обзира на то да ли сте почетник који тек започиње свој *Angular* пут или искусан програмер који жели да продуби своје знање, овај водич ће вам пружити потребне информације и примере који ће вам помоћи да савладате овај моћан алат. Наредни делови ће вас водити кроз различите аспекте рада, објашњавајући кључне концепте и технике које су неопходне за креирање успешних веб апликација.

4.1. УВОД

Да бисте започели развој *Angular* апликација, потребно је да припремите своје развојно окружење са одговарајућим алатима. Овај процес укључује инсталацију неколико кључних компоненти које ће вам омогућити да креирате, управљате и развијате своје апликације на ефикасан начин.

Node.js је *JavaScript runtime* окружење које вам омогућава да извршавате *JavaScript* код на серверској страни. Оно је основа за рад многих алата и пакета који су неопходни за развој *Angular* апликација. Уз *Node.js*, аутоматски добијате и *npm* (*Node Package Manager*), који је алат за управљање *JavaScript* библиотекама и пакетима. *npm* вам омогућава да лако инсталирате, ажурирате и управљате зависностима у вашем пројекту.

Можете преузети *Node.js* са званичне *Node.js* странице. Након инсталације, потребно је да проверите да ли су *Node.js* и *npm* успешно инсталирани. Ово можете урадити покретањем следећих команди у терминалу:

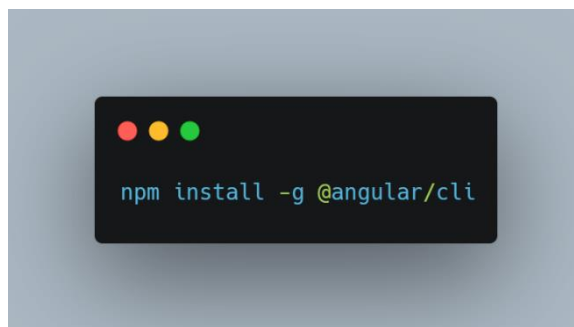


Слика 4.1. – Команде за проверу *node* и *npm* верзија

Поред *Node.js*, који је неопходан за извршавање *JavaScript* кода на серверској страни и управљање зависностима преко *npm*, за развој *Angular* апликација биће вам

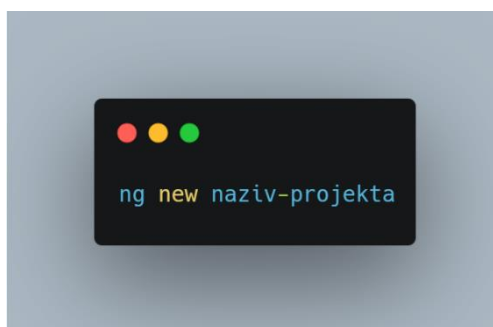
потребан и *Angular CLI* (*Command Line Interface*). *Angular CLI* је кључан алат у процесу креирања, развоја и управљања *Angular* пројектима.

Angular CLI не само да убрзава процес развоја, већ значајно побољшава продуктивност тиме што омогућава програмерима да се усредсреде на развој функционалности, а не на рутинске задатке као што су постављање пројекта или конфигурација окружења. Инсталација *Angular CLI* је једноставна и врши се преко *npm*, након чега је спреман за употребу са следећом командом:



Слика 4.1. – Команда за инсталацију глобалног *Angular CLI*

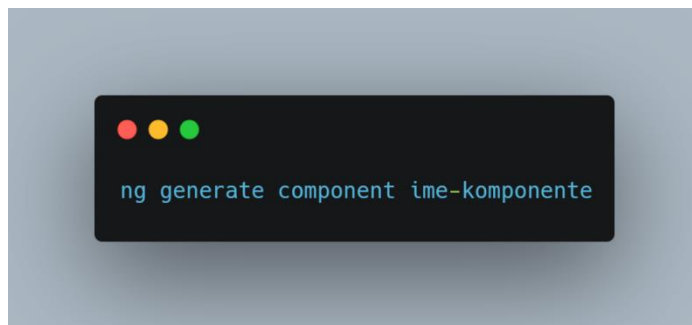
Након инсталације, *Angular CLI* омогућава креирање нових *Angular* пројеката једноставном командом:



Слика 4..1 – Команда за аутоматско генерисање *Angular* пројекта

Ова команда ће аутоматски генерисати основну структуру пројекта, укључујући све потребне конфигурационе фајлове, директоријуме и фајлове, омогућавајући вам да одмах започнете рад на апликацији.

Angular CLI такође поједностављује генерисање различитих ентитета унутар апликације, као што су компоненте, сервиси, модули и руте, чиме се омогућава бржи и једноставнији развој. На пример, генерисање нове компоненте врши се командом:



Слика 4.1. – Команда за аутоматско генерисање Angular компоненте

CLI не само да креира компоненту, већ и аутоматски ажурира све релевантне датотеке, чиме осигурава да је компонента интегрисана у апликацију на правилан начин. Такође пружа моћне алате за конфигурисање и управљање различитим окружењима за развој и производњу. Конфигурациони фајлови као што су *angular.json* и *tsconfig.json* омогућавају вам да прецизно подесите како ваша апликација функционише у различитим сценаријима, што значајно поједностављује управљање сложеним пројектима. Једна од најкориснијих функција је уграђени развојни сервер, који омогућава програмерима да брзо тестирају апликацију током развоја. Овај сервер аутоматски открива промене у коду и освежава апликацију у прегледачу у реалном времену, што значајно побољшава продуктивност и омогућава брзе корекције грешака. Сервер се покреће командом:



Слика 4.1. – Команда за покретање развојног сервера

Осим тога, *Angular CLI* нуди низ напредних опција за оптимизацију и тестирање апликације. Можете креирати оптимизоване верзије за производњу, смањити величину апликације и побољшати њене перформансе коришћењем функција као што су минификација и спајање фајлова. Поред тога, *CLI* подржава тестирање апликације помоћу алата као што су *Jasmine* и *Karma*, што омогућава темељно тестирање и обезбеђивање квалитета вашег кода. *Angular CLI* је изузетно моћан алат који значајно олакшава и убрзава развој *Angular* апликација, чинећи цео процес развоја једноставнијим и ефикаснијим.

Такође важно је да одржавате компатибилност између различитих верзија алатки како бисте осигурали да сви компоненти и делови вашег развојног окружења раде заједно без проблема. То значи да треба пажљиво бирати и ажурирати верзије различитих

алатки, као што су програмски језик, библиотеке, оквири и додаци, како би се избегли конфликти и неспојивости. Када користите усклађене верзије алатки, повећава се стабилност вашег развојног процеса и смањује вероватноћа грешака и проблема приликом развоја и изградње ваших апликација.

- *Node.js* и *npm*: Препоручује се коришћење *LTS (Long-Term Support)* верзије *Node.js* јер обезбеђује стабилност. *npm* ће бити аутоматски инсталиран уз *Node.js*.
- *Angular CLI*: Увек користите најновију стабилну верзију.
- *TypeScript (ts)*: *Angular* се заснива на *TypeScript* језику. У конкретном пројекту, верзију за *TypeScript* можете пронаћи у *package.json* датотеци у пољу *devDependencies*.

Angular	Node.js	TypeScript	RxJS
18.1.x 18.2.x	^18.19.1 ^20.11.1 ^22.0.0	>=5.4.0 <5.6.0	^6.5.3 ^7.4.0
18.0.x	^18.19.1 ^20.11.1 ^22.0.0	>=5.4.0 <5.5.0	^6.5.3 ^7.4.0
17.3.x	^18.13.0 ^20.9.0	>=5.2.0 <5.5.0	^6.5.3 ^7.4.0
17.1.x 17.2.x	^18.13.0 ^20.9.0	>=5.2.0 <5.4.0	^6.5.3 ^7.4.0
17.0.x	^18.13.0 ^20.9.0	>=5.2.0 <5.3.0	^6.5.3 ^7.4.0
16.1.x 16.2.x	^16.14.0 ^18.10.0	>=4.9.3 <5.2.0	^6.5.3 ^7.4.0
16.0.x	^16.14.0 ^18.10.0	>=4.9.3 <5.1.0	^6.5.3 ^7.4.0

Слика 4.1. – Табела компатибилних верзија за креирање Angular апликације

4.2. УПОЗНАВАЊЕ СА ANGULAR ТЕХНОЛОГИЈОМ

Основни градивни блокови *Angular* апликација укључују компоненте, шаблоне (*templates*), директиве и концепт убризгавања зависности (*dependency injection*). Сваки од ових елемената има кључну улогу у изградњи сложених и функционалних апликација. Разумевање њихових улога и начина на који међусобно интерагују је од суштинског значаја за успешно коришћење *Angular* фрејмворка (*framework*) и постизање високе ефикасности у развоју.

4.2.1. Компоненте

Компоненте су главни градивни елементи за *Angular* апликације. Свака компонента се састоји од:

- *HTML* темплејта који декларише шта се приказује на страници,
- *TypeScript* класе која дефинише понашање,
- *CSS* селектора који одређује како се компонента користи у темплејту,
- Опционо, *CSS* стилова примењених на темплејт.

Креирање компоненте у вашој апликацији може се извршити ручно путем креирања нових датотека и ручног додавања потребног кода за *HTML* шаблон, *TypeScript* класу и *CSS* стилове или коришћењем *Angular CLI* алата помоћу команде „*ng generate component* име-компоненте“, што аутоматски генерише све потребне датотеке за нову компоненту са основном структуром.

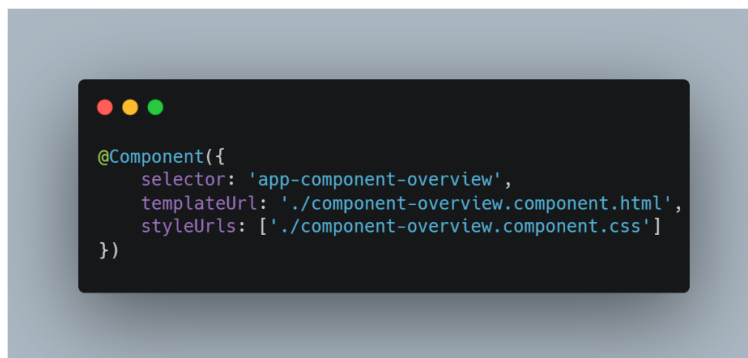
Иако је *Angular CLI* најједноставнији начин за креирање *Angular* компоненти, такође можете ручно направити компоненту. Да бисте ручно креирали нову компоненту, прво отворите директоријум вашег *Angular* пројекта и креирајте нову датотеку са називом „<име-компоненте>.component.ts“. На врху датотеке, додајте следећи `import statement`:



Слика 4.2.1. – Додавање *Component* из библиотеке *@angular/core*

Након тога, додајте *@Component* декоратор у који ћете укључити све потребне податке за компоненту. Овај декоратор укључује:

- *selector* - *CSS* селектор који одређује како се компонента користи у *HTML*.
- *templateUrl* - путања до *HTML* датотеке која садржи темплејт за компоненту.
- *styleUrls* - листа путања до *CSS* датотека које дефинишу стилове за темплејт компоненте.



Слика 4.2.1. – Пример @Component приликом ручног убацивања

На крају, додајте декларацију класе која укључује логику за компоненту. Овим корацима креирали сте основну *Angular* компоненту која је спремна за употребу у вашој апликацији.



Слика 4.2.1. – Пример класе у једној компоненти

Али такође, *Angular CLI* има у себи уграђене команде које све ове поступке извршавају аутоматски, чиме значајно олакшавају и убрзавају процес развоја. Да бисте искористили ове могућности, следите следеће кораке:

- Отворите терминал или командну линију.
- Креирајте нови *Angular* пројекат. У терминалу, одаберите директоријум у који желите да се креира пројекат и унесите следећу команду, где замените назив-пројекта са именом које желите за ваш пројекат:
 - `ng new назив-пројекта`
- Позиционирајте се унутар директоријума пројекта. У терминалу, унесите следећу команду:
 - `cd назив-пројекта`
- Креирајте нову компоненту користећи *Angular CLI*. Замените име-компоненте са жељеним именом нове компоненте:
 - `ng generate component име-компоненте`

Коришћењем *Angular CLI*, ови кораци аутоматски генеришу основне датотеке потребне за компоненту, укључујући *HTML* темплејт, *TypeScript* класу и *CSS* датотеку. Ово убрзава процес креирања компоненти и одржава конзистентност у структури пројекта.

Такође, веома је битно нагласити животни циклус компоненте. Инстанца компоненте има свој животни циклус који почиње када *Angular* инстанцира класу компоненте и приказује њен приказ заједно са под-приказима. Животни циклус се наставља кроз процес детекције промена, где *Angular* прати када се својства повезана са подацима промене, ажурирајући и приказ и инстанцу компоненте по потреби. Животни циклус се завршава када *Angular* уништи инстанцу компоненте и уклони њен шаблон из *Document Object Model (DOM)*. Директиве прате сличан животни циклус, јер *Angular* креира, ажурира и уништава њихове инстанце током извршавања. Ваша апликација може користити методе куке за животни циклус (*lifecycle hook methods*) како би се укључила у кључне догађаје у животном циклусу компоненте или директиве, омогућавајући иницијализацију нових инстанци, покретање детекције промена када је потребно, реаговање на ажурирања током детекције промена и чишћење пре него што се инстанце уклоне. Испод се налази слика табела са садржајем свих метода за животни циклус (*lifecycle hook methods*):

Hook Metode	Detaljno
<code>ngOnChanges</code>	Kada vrednost ulaznog ili izlaznog povezivanja (binding) promeni vrednost.
<code>ngOnInit</code>	Nakon prvog <code>ngOnChanges</code> poziva.
<code>ngDoCheck</code>	Prilagođeno otkrivanje promena od strane programera.
<code>ngAfterContentInit</code>	Nakon što je sadržaj komponente inicijalizovan.
<code>ngAfterContentChecked</code>	Nakon svake provere sadržaja komponente.
<code>ngAfterViewInit</code>	Nakon inicijalizacije pogleda komponente.
<code>ngAfterViewChecked</code>	Nakon svake provere pogleda komponente.
<code>ngOnDestroy</code>	Tačno pre uništavanja direktive.

Слика 4.2.1. – Слика табеле са свим функцијама за животни циклус компоненте

4.2.2. Шаблони

Шаблон (*Template*) представља део компоненте који дефинише како ће подаци бити приказани корисницима кроз *HTML* структуру. Овај *HTML* предлогач обично садржи динамичке делове који се попуњавају подацима које компонента приказује. Шаблон комбинује *HTML* са његовим директивама, омогућавајући динамичко рендеровање података и интеракцију са корисницима.

У оквиру шаблона могу се користити различите *Angular* директиве, интерполације `{{ }}`, услови *ngIf*, итерације *ngFor*, повезивање догађаја (*event binding*), двосмерно повезивање (*two-way binding*), као и позивање метода дефинисаних у *TypeScript* коду компоненте.

4.2.3. Директиве

Директиве представљају моћан алат у *Angular* окружењу који омогућава манипулацију садржајем веб странице, проширење функционалности елемената, као и динамичко прилагођавање корисничког интерфејса. Основна сврха директива је да промене понашање и изглед елемената на страници или пружи додатну функционалност тим елементима.

Постоје три врсте директива: структурне директиве, атрибутне директиве и директиве компоненти.

Структурне директиве су одговорне за измену структуре странице. Оне могу да додају, уклоне или замене елементе у оквиру документа у зависности од одређених услова. На пример, структурне директиве попут **ngIf* или **ngFor* омогућавају динамичко управљање приказивањем елемената на страници у зависности од задатих критеријума. Ова врста директиве омогућава програмерима да прилагоде изглед и функционалност странице у реалном времену, чиме се повећава интерактивност и прилагођеност корисничког интерфејса.



Слика 4.2.3. – Структурна директива

Атрибутне директиве служе за модификацију изгледа или понашања постојећих елемената на страници. Оне функционишу тако што мењају стилове, класе или друге атрибуте елемената. Примери атрибутних директива су *ngClass* и *ngStyle*, које омогућавају динамичко прилагођавање изгледа елемената на основу променљивих или услова. Атрибутне директиве су корисне за увођење додатних функционалности у елементе без потребе за изменом основног кода елемента.



Слика 4.2.3. – Атрибутна директива

Директиве компоненти омогућавају креирање нових компоненти или проширивање постојећих, додајући им специфичне функционалности. Ова врста директиве је једна од основних у *Angular* окружењу, јер се свака компонента дефинише уз помоћ `@Component` декоратора. Поред тога, директиве компоненти могу се користити за креирање прилагођених директива које додају ново понашање или функционалност елементима и компонентама на страници. Оне играју кључну улогу у модуларности и поновној употреби кода у *Angular* апликацијама.



Слика 4.2.3. – Атрибутна директива

Директиве су кључни део који омогућавају програмерима да динамички манипулишу *HTML* шаблонима и пруже богатије корисничко искуство. Комбинација различитих врста директива пружа велику моћ у развоју комплексних корисничких интерфејса, омогућавајући већу флексибилност и интерактивност на страници. Ове могућности чине *Angular* фрејмворк (*framework*) веома прилагодљивим и ефикасним за развој савремених веб апликација.

4.2.4. Убризгавање зависности

Када развијате мање делове система, као што су модули или класе, можда ће вам бити потребне функционалности других класа. На пример, може бити неопходно користити *HTTP* сервис за комуникацију са серверском страном. *Dependency Injection*, односно, је образац дизајна и механизам за креирање и испоруку делова апликације другим деловима који их захтевају. *Angular* подржава овај образац дизајна и омогућава вам да га користите у вашим апликацијама како бисте повећали флексибилност и модуларност. У *Angular* окружењу, зависности обично представљају сервиси, али могу бити и вредности попут стрингова или функција. Ињектор (*Injector*) за апликацију, који се аутоматски креира приликом покретања, инстанцира зависности када су потребне, користећи конфигурисане пружаоце за сервис или вредност. *Dependency Injection* је један од основних концепата за *Angular*. *Dependency Injection* је интегрисан у овај оквир и омогућава класама са *Angular* декораторима, попут компоненти, директива и сервиса, да конфигуришу зависности које су им потребне.

Постоје две главне улоге: потрошач зависности и пружалац зависности. *Angular* олакшава интеракцију између потрошача и пружалаца зависности коришћењем апстракције назване ињектор (*injector*). Када се затражи зависност, ињектор проверава свој регистар да види да ли је већ доступна инстанца. Ако није, нова инстанца се креира и чува у регистру. *Angular* креира ињектор на нивоу целе апликације током процеса покретања, као и било које друге ињекторе по потреби. У већини случајева није потребно ручно креирање ињектора, али је важно разумети да постоји слој који повезује пружаоце и потрошаче зависности.

На пример, ако постоји класа названа *HeroService* која треба да се користи као зависност у компоненти, први корак је додавање декоратора *@Injectable* како би се показало да класа може бити убачена као зависност.



Слика 4.2.4. – Додавање *@Injectable* декоратор на класу

На пример, ако постоји класа названа *HeroService* која треба да се користи као зависност у компоненти, први корак је додавање декоратора *@Injectable* како би се показало да класа може бити убачена као зависност.

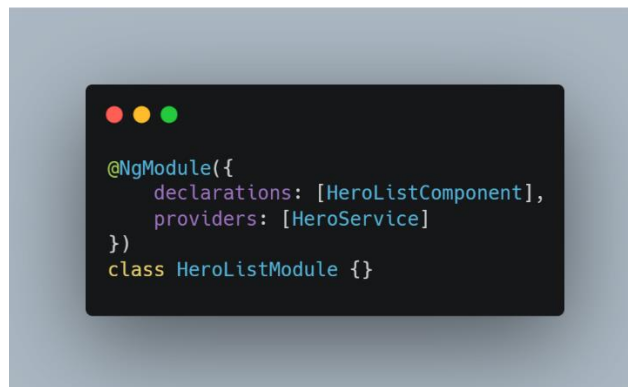
Следећи корак је учинити ову класу доступном у *Dependency Injection* систему. Зависност се може обезбедити на више нивоа. На нивоу компоненте, *HeroService* постаје доступан само тој компоненти и њеним подкомпонентама.

A screenshot of a code editor showing the definition of a component. The code is as follows:

```
@Component({
  selector: 'hero-list',
  template: '...',
  providers: [HeroService]
})
class HeroListComponent {}
```

Слика 4.2.4. – Доступност класе на нивоу компоненте

На нивоу модула, *HeroService* постаје доступан свим компонентама и сервисима унутар тог модула.

A screenshot of a code editor showing the definition of a module. The code is as follows:

```
@NgModule({
  declarations: [HeroListComponent],
  providers: [HeroService]
})
class HeroListModule {}
```

Слика 4.2.4. – Доступност класе на нивоу модула

На нивоу корена апликације омогућава да се зависност користи у свим деловима апликације. Ово се постиже додавањем поља *providedIn: 'root'* декоратору *@Injectable*.



```
@Injectable({  
  providedIn: 'root'  
})  
class HeroService {}
```

Слика 4.2.4. – Доступност класе на нивоу модула

Када обезбедите сервис на нивоу корена, *Angular* креира једну заједничку инстанцу *HeroService* и убацује је у било коју класу која је затражи. Ово такође омогућава да *Angular* оптимизује апликацију уклањањем сервиса из финалне верзије ако се не користи, што је познато као *tree-shaking*.

Најчешћи начин за убацивање зависности је декларисање истих у конструктору класе. Када *Angular* креира нову инстанцу компоненте, прегледа типове параметара у конструктору и обезбеђује одговарајуће зависности. Друга опција је коришћење методе *inject* за убацивање зависности.



```
@Component({ ... })  
class HeroListComponent {  
  constructor(private service: HeroService) {}  
}
```

Слика 4.2.4. – Убацивање зависности у конструктор



```
@Component({ ... })  
class HeroListComponent {  
  private service = inject(HeroService);  
}
```

Слика 4.2.4. – Убацивање зависности преко функције *inject*

Када *Angular* открије да компонента зависи од сервиса, прво проверава да ли ињектор већ има постојеће инстанце тог сервиса. Ако тражена инстанца сервиса још не постоји, ињектор креира једну користећи регистрованог пружаоца и додаје је ињектору пре него што врати сервис.

Када су све тражене услуге разређене и враћене, *Angular* може позвати конструктор компоненте са тим услугама као аргументима. Овај приступ омогућава једноставно управљање зависностима и доприноси лакшем одржавању и развоју апликације.

4.3. НАЈВОЉА ПРАКСА

Приликом развоја *Angular* апликација, примена најбољих пракси је кључна за обезбеђивање безбедности, приступачности и перформанси апликације. Смернице које следе помажу да апликације буду функционалне, ефикасне и оптимизоване за кориснике. Увођење ових пракси не само да побољшава корисничко искуство, већ и осигурава да апликације задовољавају највише стандарде у индустрији.

4.3.1. Сигурност

Cross-site scripting (XSS) омогућава нападачима да убаце злонамерни код у веб странице. Такав код може, на пример, украсти корисничке податке и податке за пријављивање или извршити радње које се представљају као корисник. Ово је један од најчешћих напада на веб. Да бисте блокирали XSS нападе, морате спречити злонамерни код да уђе у *Document Object Model* (DOM). На пример, ако нападачи успеју да вас преваре да уметнете *script* таг у DOM, они могу покренути произвољан код на вашем веб сајту. Напад није ограничен на *script* тагове, многи елементи и својства омогућавају извршење кода, на пример, *img* и *a*. Ако подаци под контролом нападача уђу у DOM, очекујте сигурносне рањивости.

Да би систематски блокирао XSS грешке, *Angular* подразумевано третира све вредности као неверификоване. Када се вредност уметне у DOM путем везивања шаблона или интерполације, *Angular* врши санитизацију и ескапирање неверификованих вредности. Ако је вредност већ претходно санитизована и сматра се безбедном, то треба изкомуницирати означавањем те вредности као поуздану. За разлику од вредности које се користе за приказ, шаблони (*template*) се подразумевано сматрају поузданим и треба их третирати као извршиви код. Никада не креирајте шаблоне спајањем корисничког уноса и синтаксе шаблона. То би омогућило нападачима да убаце произвољан код у вашу апликацију. Како бисте спречили ове рањивости, увек користите подразумевани *Ahead-Of-Time* (AOT) компајлер шаблона у продукцијским имплементацијама.

Додатни слој заштите може се обезбедити коришћењем *Content Security Policy* и *Trusted Types*. Ове функционалности веб платформе оперишу DOM, што је најефикасније место за спречавање XSS проблема. Овде их није могуће заобићи коришћењем других, нижих нивоа апликативног програмерског интерфејса (API). Из овог разлога, снажно се препоручује искористити ове могућности. Да бисте то постигли, конфигуришите *Content Security Policy* за апликацију и омогућите примену *Trusted Types*.

Санитизација је инспекција неповерљиве вредности, претварајући је у вредност која је сигурна за уметање у *DOM*. У многим случајевима, санитизација уопште не мења вредност. Санитизација зависи од контекста: Вредност која је безопасна у фајловима за стилизовање компоненти (*CSS*) може бити потенцијално опасна у *URL*.

Angular дефинише следеће сигурносне контексте:

- *HTML*: Користи се приликом тумачења вредности као *HTML*, на пример, приликом везивања за *innerHTML*.
- *Style*: Користи се приликом везивања *CSS* синтаксе у *style* својство.
- *URL*: Користи се за *URL* својства, као што је `<a href>`.
- *Resource URL*: *URL* који се учитава и извршава као код.

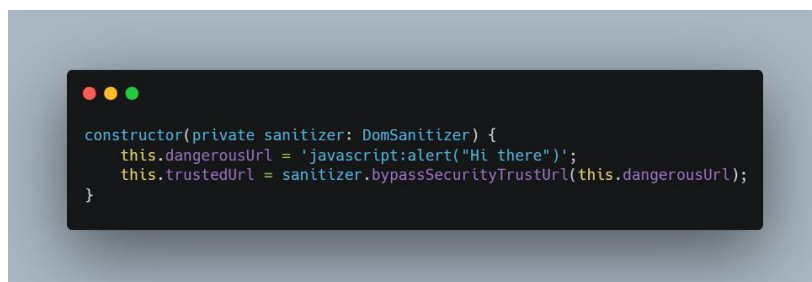
Angular врши санитизацију непроверених вредности за *HTML*, стилове и *URL*. Санитизација ресурсних *URL* није могућа јер садрже произвољан код. У режиму развоја, *Angular* штампа упозорење у конзоли када мора да промени вредност током санитизације.

Поверовање безбедних вредности: Понекад апликације заиста треба да укључе извршиви код, прикажу *iframe* са одређеног *URL* или конструишу потенцијално опасне *URL*. Како бисте спречили аутоматску санитизацију у овим ситуацијама, обавестите *Angular* да сте прегледали вредност, проверили како је креирана и осигурали да је безбедна. Будите опрезни. Ако поверите вредност која може бити злонамерна, увешћете сигурносну рањивост у вашу апликацију. Ако имате сумње, потражите професионалног ревизора сигурности.

Да бисте означили вредност као поуздану, убаците *DomSanitizer* и позовите једну од следећих метода:

- *bypassSecurityTrustHtml*
- *bypassSecurityTrustScript*
- *bypassSecurityTrustStyle*
- *bypassSecurityTrustUrl*
- *bypassSecurityTrustResourceUrl*

Обично, *Angular* аутоматски врши санитизацију *URL*, онемогућава опасан код и у режиму развоја, бележи ову акцију у конзоли. Да бисте спречили ово, означите вредност *URL* као поуздан *URL* користећи позив *bypassSecurityTrustUrl*:



Слика 4.3.1. – Убацавање зависности преко функције *inject*

Ако треба да претворите кориснички унос у поуздану вредност, користите метод компоненте. Следећи шаблон омогућава корисницима да унесу *ID YouTube* видеа и читају одговарајући видео у *iframe*. Атрибут *iframe src* је у контексту сигурносног ресурсног *URL*, јер непроверен извор може, на пример, убацити преузимања датотека које неопрезни корисници могу покренути. Да бисте то спречили, позовите метод на компоненти да конструише поуздан *URL* за видео, што омогућава да дозволи везивање у *iframe src*.

4.3.2. Приступачност

Програмери морају бити свесни да приступачност није само опција већ и неопходност. Веб користе разноврсне групе људи, укључујући особе са визуелним, слуховним или моторичким оштећењима. Због тога је од суштинског значаја да веб апликације буду дизајниране тако да буду доступне свим корисницима, без обзира на њихове могућности. Применом приступачних техника не само да се побољшава корисничко искуство за особе са инвалидитетом, већ се истовремено побољшава корисничко искуство за све кориснике.

Кључ за постизање ове приступачности лежи у коришћењу стандарда као што су *ARIA (Accessible Rich Internet Applications)* атрибуту и коришћењу нативних *HTML* елемената који су већ оптимизовани за приступачност. *Angular*, са својим богатим сетом алата и библиотека, попут *Angular Material* и *Component Development Kit (CDK)*, пружа програмерима све неопходне алате за креирање приступачних веб апликација. Ове библиотеке укључују компоненте и алате који подржавају различите аспекте приступачности, као што су *LiveAnnouncer* и *cdkTrapFocus*, који помажу у управљању корисничким фокусом и доступношћу садржаја. Поред тога, коришћење нативних *HTML* елемената где год је то могуће, као што су `<button>` или `<a>`, обезбеђује да се користе стандарди који су добро подржани од стране прегледача и алата за приступачност. Ово је посебно важно за креирање интерактивних елемената и контрола који су кључни за корисничко искуство. Такође, у случајевима када је потребно користити контејнере за нативне елементе, важно је осигурати да ови елементи остану доступни и да подржавају све потребне атрибуте и својства која су важна за приступачност. Ово се може постићи коришћењем пројекције садржаја у *Angular* компонентама, чиме се омогућава да нативни елементи буду део *API* компоненте, што додатно осигурава приступачност.

Пример ове праксе може се видети у *MatFormField* компоненти, која користи пројекцију садржаја за укључивање нативних контрола у свој *API*. Овај приступ омогућава креирање компоненти које не само да су флексибилне и моћне, већ и у потпуности приступачне свим корисницима.

4.3.3. Учитавање по потреби

Учитавање по потреби (*Lazy loading*) је техника која омогућава учитавање одређених модула или компоненти апликације само када су заиста потребни, а не одмах при покретању апликације. Овај приступ значајно умањује величину почетног пакета апликације, што доводи до бржег учитавања и бољег корисничког искуства, нарочито код великих апликација са бројним рутинама и функционалностима.

Основни принцип за *Lazy loading* је да се модул учитава тек када корисник приступи одређеној рути, што значи да модул није део почетног пакета. На пример, ако апликација има више секција које се ретко користе, али су ипак важне, те секције могу бити постављене да се учитавају само када корисник одлучи да их отвори. Да бисте користили *Lazy loading* за *Angular* модуле, користи се опција *loadChildren* у конфигурацији рута у вашем *AppRoutingModule* фајлу. Ова опција омогућава да динамички учитава модуле тек када је то неопходно, чиме се оптимизује перформанса апликације. Ево примера кода који приказује како се конфигурише рутинг за *Lazy loading*:

A screenshot of a code editor with a dark background and light-colored text. The code defines a route for 'items' using the loadChildren property to load the ItemsModule dynamically.

```
const routes: Routes = [{
  path: 'items',
  loadChildren: () => import('./items/items.module')
    .then(m => m.ItemsModule)
}];
```

Слика 4.3.3. – Пример за *Lazy loading*

У овом примеру, модул *ItemsModule* ће бити учитан само када корисник приступи рути *items*. У рутинг модулу *Lazy-loaded* модула, потребно је додати руту за компоненту, као што је приказано у следећем примеру:

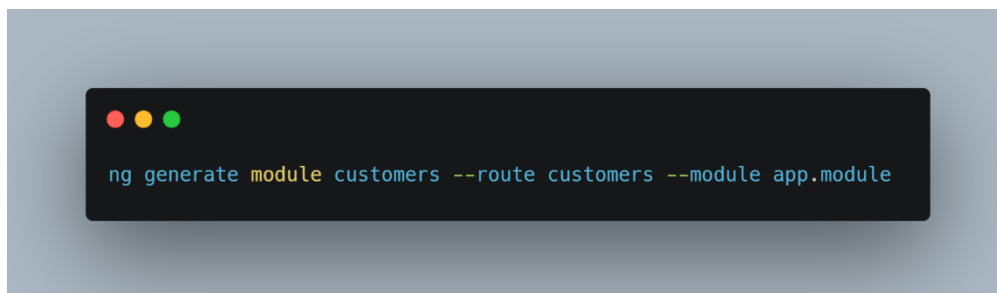
A screenshot of a code editor with a dark background and light-colored text. The code defines a route for the root path ('') with a component and lazy loading.

```
const routes: Routes = [{
  path: '',
  component: ItemsComponent
}];
```

Слика 4.3.3. – Пример за *Lazy loading*

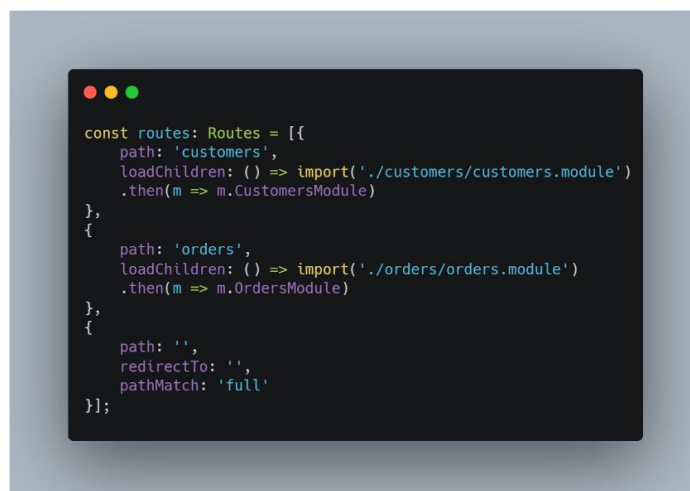
Такође, обавезно је уклонити *ItemsModule* из *AppModule*. Ово осигурава да се модул учитава само када је потребан, а не приликом почетног учитавања апликације. За постављање *Lazy-loaded* функционалног модула, следећи кораци су потребни:

- Креирајте функционални модул користећи *Angular CLI*, уз *--route* заставицу.
- Конфигуришите *AppRoutingModule* руте.



Слика 4.3.3. – Команда за креирање новог модула са рутом

Ова команда креира *customers* директоријум који садржи нови *Lazy-loadable* функционални модул *CustomersModule*, као и рутинг модул *CustomersRoutingModule*. Команда аутоматски декларише *CustomersComponent* и увози *CustomersRoutingModule* унутар новог функционалног модула. За правилно *Lazy loading* функционисање, уверите се да су руте исправно конфигурисане у *app-routing.module.ts* фајлу:



Слика 4.3.3. – Дефинисане руте са *loadChildren*

Прве две путање су руте ка *CustomersModule* и *OrdersModule*. Последњи унос дефинише подразумевану руту. Оваква конфигурација омогућава да се модули учитавају само када су заиста потребни, што побољшава перформансе апликације.

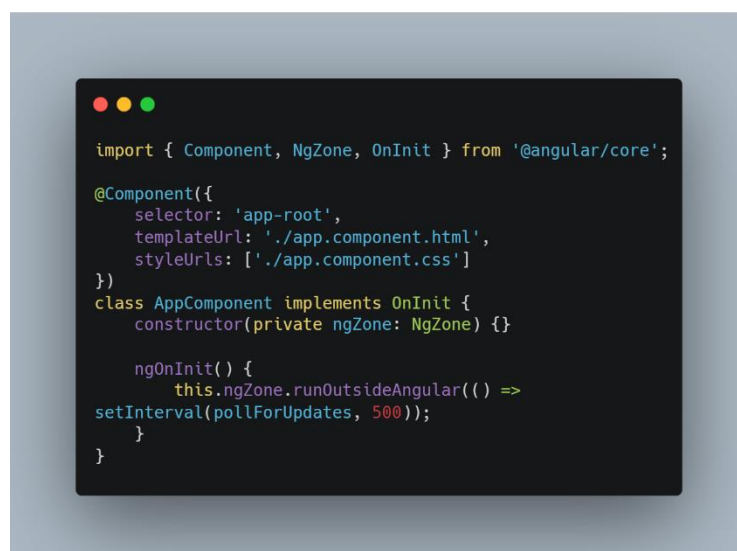
4.4. ВОДИЧ ЗА ПРОГРАМЕРЕ

Водич за програмере пружа детаљне информације и упутства за коришћење различитих *Angular* функционалности, омогућавајући програмерима да на прави начин примене овај моћан оквир у својим апликацијама. *Angular* је више од самог оквира, он представља свеобухватан систем који интегрише бројне библиотеке за управљање разноврсним аспектима развоја веб апликација. Ове библиотеке покривају све, од рутирања и управљања формама до сложенијих аспеката као што су комуникација између клијента и сервера.

4.4.1. Детектовање промена

Детекција промена и оптимизација рада представљају кључне аспекте који обезбеђују да апликације функционишу ефикасно и реагују на промене у подацима на правовремен и оптимизован начин. *Angular* периодично покреће механизам за откривање промена како би осигурао да се све измене у подацима правилно одразе у приказу апликације. Овај процес је дизајниран да буде веома ефикасан, али у комплексним апликацијама са великим бројем компоненти може доћи до успоравања уколико се детекција промена покреће сувише често или у непотребним ситуацијама.

Zone.js је механизам који *Angular* користи за праћење асинхроних операција као што су тајмери, мрежни захтеви и слушаоци догађаја. Када се детектује промена која може утицати на стање апликације, *Angular* активира циклус детекције промена како би ажурирао приказ. Међутим, постоје ситуације у којима ове промене не доводе до стварних измена у подацима, што може довести до непотребних позива за детекцију промена и тиме успорити апликацију. Да би се избегли непотребни циклуси детекције промена, *Angular* омогућава извршавање одређених задатака изван *Angular* зоне користећи метод *runOutsideAngular*. Ово је посебно корисно у случајевима када се користе библиотеке трећих страна које нису оптимизоване за механизам детекције промена. Пример кода који показује како се извршава задатак изван *Angular* зоне:



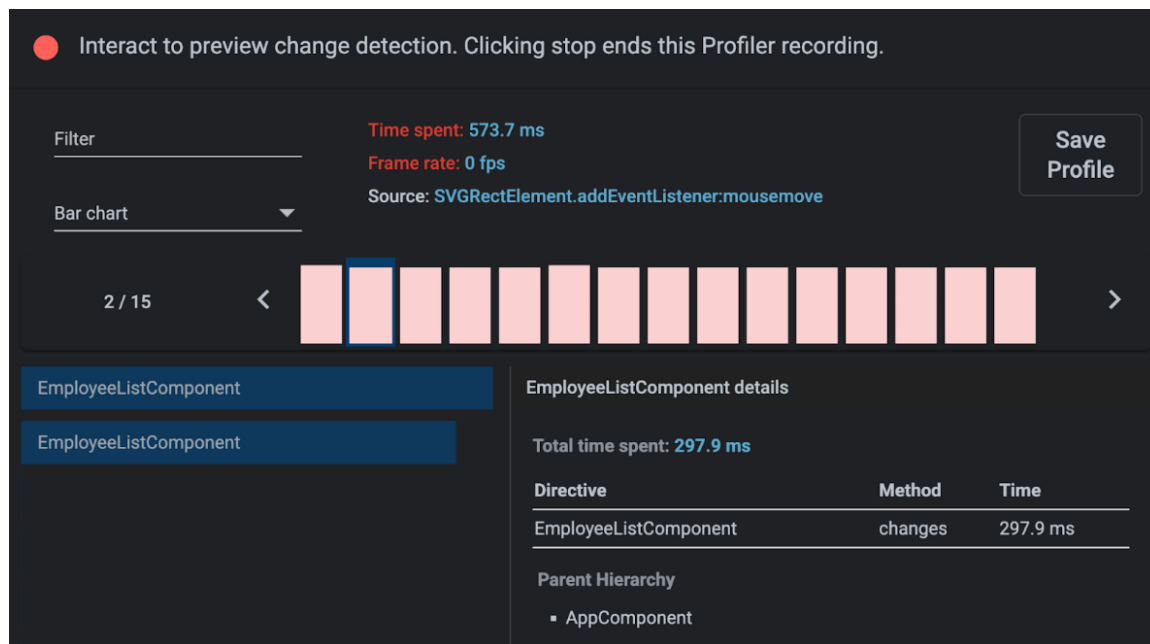
```
import { Component, NgZone, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
class AppComponent implements OnInit {
  constructor(private ngZone: NgZone) {}

  ngOnInit() {
    this.ngZone.runOutsideAngular(() =>
      setInterval(pollForUpdates, 500));
  }
}
```

Слика 4.4.1. – Дефинисане руте са *loadChildren*

Овај код показује како се метод *setInterval* покреће изван *Angular* зоне, чиме се избегава покретање непотребне детекције промена након извршења функције *pollForUpdates*. Када је у питању оптимизација, идентификовање и избегавање спорих рачунања је такође од великог значаја. *Angular DevTools* пружа алате за профилисање који омогућавају програмерима да идентификују компоненте које узрокују успоравање у циклусима детекције промена и да на тај начин оптимизују перформансе апликације.



Слика 4.4.1. – Циклус детекције промена у *Angular DevTools*

4.4.2. Рутирање и навигација

Angular рутирање у апликацијама са једном страном омогућава да се промени приказ на екрану без поновног учитавања странице. Уместо да се сваки пут иде на сервер по нову страницу, *Angular* користи рутер који омогућава приказивање или скривање делова екрана који одговарају одређеним компонентама. Корисници могу лако да прелазе између различитих приказа који су дефинисани у апликацији. *Angular Router* управља навигацијом интерпретирајући *URL* у прегледачу као инструкцију за промену приказа. Ово омогућава динамичну и брзу промену садржаја, без прекида у корисничком искуству. За почетак коришћења рутирања, потребно је да имате основно знање о компонентама и шаблонима, као и да имате постављену *Angular* апликацију.

Дефинисање основне руте укључује три главна корака: импортовање рута у конфигурациони фајл, креирање низа рута и њихово повезивање са одговарајућим компонентама. *Angular CLI* аутоматски обавља већину ових корака, али ако радите на пројекту који није креиран помоћу *CLI*, потребно је да ручно проверите исправност конфигурације.

```
export const appConfig: ApplicationConfig = {  
  providers: [provideRouter(routes)]  
};
```

Слика 4.4.2. – Провера исправности конфигурације

```
import { Routes } from '@angular/router';  
  
export const routes: Routes = [  
  { path: 'first-component', component:  
    FirstComponent },  
  { path: 'second-component', component:  
    SecondComponent },  
];
```

Слика 4.4.2. – Постављање низа рута

Сада када сте дефинисали своје руте, додате их у апликацију. Прво, додајте линкове ка двема компонентама и ажурирајте шаблон своје компоненте да укључује `<router-outlet>`. Ово омогућава да *Angular* ажурира приказ апликације компонентом за изабрану руту. Пример *HTML* кода за ово би изгледао овако:

```
<h1>Angular Router App</h1>  
<nav>  
  <ul>  
    <li><a routerLink="/first-component" routerLinkActive="active"  
      ariaCurrentWhenActive="page">First Component</a></li>  
    <li><a routerLink="/second-component" routerLinkActive="active"  
      ariaCurrentWhenActive="page">Second Component</a></li>  
    </ul>  
  </nav>  
<router-outlet></router-outlet>
```

Слика 4.4.2. – Пример за `router-outlet`

Редослед рута је веома важан јер Angular Router користи стратегију да први има предност, што значи да специфичније руте треба да буду постављене изнад мање специфичних. На пример, додавање *wildcard* руте за 404 страницу. Коришћењем ове конфигурације, ако корисник унесе URL који не одговара ниједној од дефинисаних рута, биће преусмерен на компоненту *PageNotFoundComponent*. Ако желите да додате преусмеравање, рецимо, са почетне странице на неку другу руту, можете то одрадити.



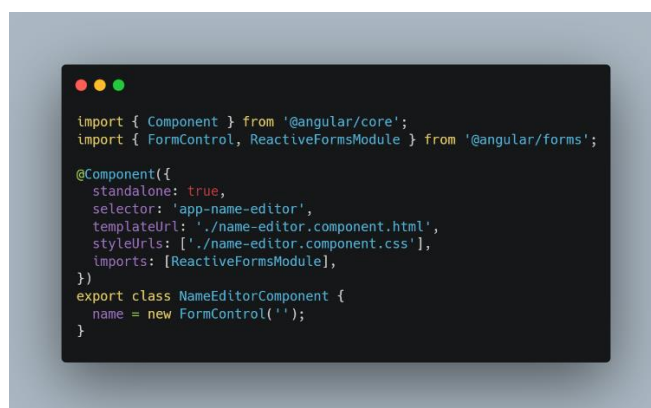
```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '', redirectTo: '/first-component', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent },
];
```

Слика 4.4.2. – Преусмеравање на */first-component* и *PageNotFoundComponent*

4.4.3. Форме

Управљање корисничким уносом путем форми је кључни део већине апликација. Форме су ту да омогуће корисницима да се пријаве, ажурирају своје профиле, унесу осетљиве информације и обаве бројне друге задатке који захтевају унос података. *Angular* пружа два приступа за рад са формама: реактивни и приступ базиран на шаблону. Оба приступа имају за циљ да бележе кориснички унос, валидирају га, креирају моделе форме и ажурирају податке. Оба приступа такође пружају средства за праћење промена у формама.

Реактивне форме (*Reactive forms*) су структуралније, робусније и пружају бољу контролу над уносима. Засноване су на моделу и пружају синхрони приступ подацима, што их чини погодним за веће и сложеније апликације. На пример, свака промена стања у реактивним формама генерише ново стање, што омогућава лакше управљање и тестирање апликација.



```
import { Component } from '@angular/core';
import { FormControl, ReactiveFormsModule } from '@angular/forms';

@Component({
  standalone: true,
  selector: 'app-name-editor',
  templateUrl: './name-editor.component.html',
  styleUrls: ['./name-editor.component.css'],
  imports: [ReactiveFormsModule],
})
export class NameEditorComponent {
  name = new FormControl('');
}
```

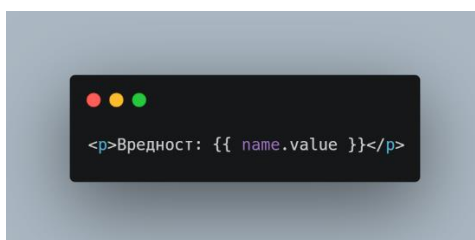
Слика 4.4.3. – Постављање реактивне форме у компоненти

Када је форма креирана у класи компоненте, треба је повезати са HTML елементом у шаблону.



Слика 4.4.3. – Повезивање реактивне форме са HTML

На овај начин, омогућава се двосмерна веза између контроле форме и њеног приказа. Вредност унесена у поље се одмах приказује у шаблону, а промена у моделу аутоматски мења приказ. Пример приказа тренутне вредности контроле форме у шаблону.



Слика 4.4.3. – Приказ вредности *FormControl*

Форме засноване на шаблону (*Template-driven*) представљају једноставнији приступ за управљање корисничким уносом у *Angular* апликацијама. Овај приступ је веома погодан за мање сложене форме и за ситуације где није потребно пуно логике на страни модела података. Уместо да се ослањају на сложену структуру модела и посматраче, форме засноване на шаблону директно користе *HTML* и директиве, чиме се постиже веома интуитиван и лак за коришћење систем за управљање подацима. Главна карактеристика форми заснованих на шаблону је коришћење директиве *ngModel*, која омогућава двосмерно везивање података између *HTML* елемената и *TypeScript* класе. То значи да свака промена коју корисник направи у форми одмах утиче на модел података у апликацији, и обрнуто свака промена модела података се аутоматски одражава у приказу форме.

Слика 4.4.3. – Повезивање *ngModel* са *name*

Овде, *ngModel* директива повезује унос корисника са својством *name* у *TypeScript* класи, омогућавајући тренутну синхронизацију података. Овај начин рада омогућава програмерима да брзо креирају једноставне форме без потребе за сложеним моделима података, што је веома корисно за мање апликације или апликације са мање сложеним захтевима.

Када је реч о валидацији података, *Angular* нуди низ уграђених алата који поједностављују овај процес. Једна од предности овог приступа је могућност лаког додавања правила за валидацију директно у *HTML* шаблон. На пример, можемо да користимо атрибут *required* како бисмо означили да је поље обавезно, и *Angular* ће аутоматски валидирати податке, при чему можемо приказати поруку о грешци ако услов није испуњен:

Слика 4.4.3. – Повезивање *ngModel* са *name*

Овакав начин рада омогућава да се валидација управља директно у шаблону, што чини процес једноставнијим и интуитивнијим, без потребе за додатном логиком у класи компоненте. Коришћењем ових уграђених *Angular* алата, програмери могу лако управљати статусом форме, приказивати грешке у валидацији и омогућити једноставне интерфејсе за унос података.

Форме засноване на шаблону су нарочито корисне за мале апликације или оне у којима је кориснички унос минималан. У оваквим сценаријима, овај приступ нуди брзо и лако решење које омогућава једноставан развој без потребе за сложеним подешавањима или напредним техникама.

4.4.4. HttpClient

HttpClient модул је веома моћан и флексибилан алат за рад са *HTTP* захтевима, омогућавајући вам да лако комуницирате са серверима и обрадите податке у различитим форматима као што су *JSON*, текст, или бинарни подаци. Овај модул поједностављује рад са *RESTful API*, што значи да можете лако слати и примати захтеве, као и руковати одговорима и евентуалним грешкама. Ово је веома важно за изградњу модерних веб апликација које често захтевају динамичку комуникацију са сервером. Увођењем *HttpClientModule* у *Angular* модул апликације, добијате приступ свим неопходним функционалностима за слање различитих типова захтева, укључујући *GET*, *POST*, *PUT* и *DELETE* захтеве. Ово је основни корак у постављању било које апликације која се ослања на серверску комуникацију.

Када желите да преузмете податке са сервера, најчешћи тип захтева је *GET* захтев, којим се информације преузимају без мењања стања на серверу. Овај захтев се обично користи за приказивање података корисницима. На пример, једноставна апликација која захтева листу података може користити *GET* метод како би та листа била приказана на страници:



```
import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-data-fetch',
  templateUrl: './data-fetch.component.html',
  styleUrls: ['./data-fetch.component.css']
})
export class DataFetchComponent implements OnInit {
  data: any;

  constructor(private http: HttpClient) {}

  ngOnInit() {
    this.http.get('https://api.example.com/data')
      .subscribe(response => {
        this.data = response;
      });
  }
}
```

Слика 4.4.4. – Пример позива *GET* методе

Овде *HttpClient* шаље *GET* захтев на дати *URL*, а одговор се обрађује асинхронно кроз *subscribe()* метод. Ово омогућава апликацији да настави са радом док чека податке, што је веома важно за добар кориснички доживљај.

Са друге стране, *POST* захтев се користи за слање података серверу, обично када се жели креирати нови ресурс. Овај захтев мења стање на серверу, као када корисник поднесе форму за регистрацију или унос нових података у базу. Ево једноставног примера слања података преко *POST* захтева.



```

import { HttpClient } from '@angular/common/http';
import { Component } from '@angular/core';

@Component({
  selector: 'app-data-submit',
  templateUrl: './data-submit.component.html',
  styleUrls: ['./data-submit.component.css']
})
export class DataSubmitComponent {
  constructor(private http: HttpClient) {}

  submitData() {
    const postData = { name: 'John', age: 30 };
    this.http.post('https://api.example.com/submit', postData)
      .subscribe(response => {
        console.log('Response:', response);
      });
  }
}

```

Слика 4.4.4. – Пример позива *POST* методе

Овде видимо како *post()* метода шаље објекат са подацима серверу. Када сервер прими захтев и обради податке, резултат се враћа и приказује у конзоли, што је одличан начин да потврдите успешност захтева.

Један од кључних аспеката у раду са *HTTP* захтевима је и руковање грешкама. У случајевима када дође до проблема, било због лоше мрежне везе, недоступности сервера или неправилно форматираних података, важно је да апликација може на одговарајући начин да реагује и прикаже кориснику разумљиву поруку о грешци. За то се користи *catchError* оператор, који омогућава хватање и обраду грешака.



```

import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';
import { catchError } from 'rxjs/operators';
import { throwError } from 'rxjs';

@Component({
  selector: 'app-data-fetch',
  templateUrl: './data-fetch.component.html',
  styleUrls: ['./data-fetch.component.css']
})
export class DataFetchComponent implements OnInit {
  data: any;
  error: string | undefined;

  constructor(private http: HttpClient) {}

  ngOnInit() {
    this.http.get('https://api.example.com/data')
      .pipe(
        catchError(error => {
          this.error = 'Failed to fetch data';
          return throwError(error);
        })
      )
      .subscribe(response => {
        this.data = response;
      });
  }
}

```

Слика 4.4.4. – Пример позива *POST* методе

catchError хвата сваку потенцијалну грешку која се може јавити током извршавања *GET* захтева. Ако дође до грешке, порука се поставља у својство *error*, које се касније може приказати кориснику у интерфејсу.

Angular *HttpClient* пружа робусну инфраструктуру за руковање свим аспектима *HTTP* комуникације, од слања једноставних захтева до руковања сложеним одговорима и грешкама. Овај модул омогућава лако и ефикасно интегрисање са било којим API сервисом, чинећи вашу апликацију спремно за напредне функције као што су динамично преузимање података, слање форме или слање великих количина података у базу.

4.4.5. Рендеровање на серверској страни

Рендеровање на серверској страни (*SSR*) представља технику која омогућава да *Angular* апликација буде рендерована на серверу пре него што се испоручи клијенту. Овај процес омогућава да корисници виде статички *HTML* одмах, пре него што се учита *JavaScript*, што знатно убрзава приказ странице, посебно за кориснике на споријим мрежама или уређајима са мање ресурса. Такође, овај начин приказивања садржаја побољшава оптимизацију за претраживаче (*SEO*), јер претраживачи имају лакши приступ рендерованом садржају.

Angular Universal је алат који омогућава *SSR* у *Angular* апликацијама. Када је *SSR* омогућен, сервер преузима иницијално рендеровање за *HTML*, док клијент након учитавања *JavaScript* кода, преузима потпуну функционалност апликације. Ово не само да убрзава учитавање, већ чини апликацију приступачнијом и омогућава боље рангирање на претраживачима. Први корак када постављамо *SSR* јесте додавање *Angular Universal* пакета у постојећи пројекат. Ово можете урадити командом `ng add @nguniversal/express-engine`.



Слика 4.4.5. – Додавање *Angular Universal* пакета у апликацију

Овај пакет инсталира потребне зависности и аутоматски генерише конфигурационе фајлове, као што су *server.ts* и *main.server.ts*, који су неопходни за *SSR*. *server.ts* служи као главни улазни фајл за *Express* сервер који ће управљати рендеровањем *Angular* апликације на серверу. Апликација која користи *SSR* се покреће на скоро исти начин као и обична *Angular* апликација, али се приликом покретања мора додати додатак `:ssr` на команде за изградњу и покретање. Команда за компајлирање пројекта додаје *SSR* подршку, а команда за покретање сервера омогућава серверу да управља рендеровањем. Клијентски и серверски делови апликације раде синхронизовано, где сервер одмах приказује рендеровани *HTML*, а клијент преузима и активира динамичке функције након учитавања странице.

Једна од највећих предности у вези *SSR* јесте побољшано време учитавања страница, што значајно утиче на корисничко искуство. Корисници на споријим мрежама брже добијају први садржај, што чини апликацију кориснијом и пријатнијом за употребу. *SSR* такође помаже у бољем *SEO* рангирању, јер претраживачи могу да анализирају и индексирају садржај. Међутим, постоје и одређени изазови. На пример, многи *Angular*

кодови, као што су они који користе *window* или *document API*, нису компатибилни са сервером, јер ти објекти не постоје на серверској страни. Такође, имплементација за *SSR* може захтевати додатне конфигурације за сервер и већи ниво одржавања. На крају, *SSR* је корисна техника за побољшање перформанси и *SEO* оптимизацију *Angular* апликација, али захтева пажљиво планирање и оптимизацију како би се искористиле све њене предности.

4.4.6. Статички генерисани сајтови

Статички генерисани сајтови (*SSG*) омогућавају унапред генерисање *HTML* страница током процеса изградње, уместо динамичког приказивања садржаја на страни клијента или сервера. Овим приступом се побољшава брзина учитавања страница, омогућавају се боље перформансе и побољшава се *SEO* оптимизација, јер претраживачи могу лакше да индексирају унапред генерисан садржај.

Angular користи *Angular Universal* за омогућавање статичког генерисања, омогућавајући вам да унапред генеришете *HTML* странице за све руте у вашој апликацији. Једном када се страница генерише током процеса израде, може се директно сервирати као статички *HTML*, *CSS* и *JavaScript*. Ово је посебно корисно за пројекте попут блогова или маркетиншких сајтова где се садржај не мења често.

Након инсталације *Angular Universal*, процес изградње се врши скоро исто као и код серверског рендеровања, с тим што користите команду *npm run build:ssg* да бисте генерисали статичке *HTML* странице за све дефинисане руте.

4.4.7. Тестирање

Тестирање је основни део развоја *Angular* апликација и игра кључну улогу у осигурању да апликација функционише како се очекује, без грешака и непредвиђених понашања. *Angular* долази са интегрисаним алатима за тестирање који омогућавају једноставно спровођење јединичних тестова, тестова компоненти и потпуно тестирање апликација (*end-to-end* тестова). Уз добру тестну покривеност, развојни тимови могу брже и сигурније имплементирати нове функционалности, знајући да постојећи код неће бити неочекивано нарушен. *Angular* користи два главна алата за јединично тестирање: *Jasmine*, као фрејмворк (*framework*) за писање тестова, и *Karma*, као алат за покретање тестова у различитим окружењима. *Jasmine* омогућава писање читљивих и интуитивних тестова, док *Karma* покреће те тестове у различитим претраживачима и приказује резултате у реалном времену.

Када креирате *Angular* пројекат уз помоћ *Angular CLI*, све што вам је потребно за тестирање већ је конфигурирано. *CLI* аутоматски генерише основне тестове за сваку компоненту, сервис или модул. Покретање тестова је једноставно помоћу команде *ng test*, која покреће *Karma* и приказује резултате у конзоли и претраживачу.

Јединично тестирање користи *Jasmine* синтаксу која омогућава груписање тестова у блокове помоћу *describe* и писање појединачних тестова помоћу *it*. На пример, ако имамо сервис који додаје два броја, тест би могао изгледати овако:



```

import { TestBed } from '@angular/core/testing';
import { CalculatorService } from './calculator.service';

describe('CalculatorService', () => {
  let service: CalculatorService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(CalculatorService);
  });

  it('should add two numbers correctly', () => {
    const result = service.add(2, 3);
    expect(result).toBe(5);
  });
});

```

Слика 4.4.7. – Пример јединичног тестирања користћи Jasmine

Овде се тестира да ли сервис исправно враћа збир два броја. Прво се креира тестно окружење и инстанцира сервис, а затим се проверава очекивани резултат.

Када су у питању компоненте, *Angular* пружа алате за тестирање компоненти у изолованом окружењу, симулирајући понашање апликације у реалним условима. На пример, тестирање да ли компонента правилно рендерује податке може изгледати овако:



```

import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MyComponent } from './my-component.component';

describe('MyComponent', () => {
  let component: MyComponent;
  let fixture: ComponentFixture<MyComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ MyComponent ]
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(MyComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should render the title', () => {
    const compiled = fixture.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('My Component Title');
  });
});

```

Слика 4.4.7. – Пример провере тестирања података

Овај тест креира инстанцу компоненте и проверава да ли је њен шаблон правилно рендерован, односно да ли *HTML* садржи исправан наслов.

Потпуни тестови (*end-to-end*) симулирају корисничке интеракције са апликацијом, тестирајући целокупну функционалност у условима који су веома слични стварној употреби. За разлику од јединичних тестова који се фокусирају на појединачне функције или компоненте, *end-to-end* тестови проверавају како различити делови апликације функционишу заједно, обезбеђујући да све компоненте и сервиси међусобно комуницирају како је предвиђено. Користећи алате као што је *Protractor*, *Angular* омогућава да се тестови извршавају у стварном претраживачу, што симулира акције правог корисника од кликова на дугмад, попуњавања формулара, до навигације између страница. Ови тестови прате комплетан кориснички ток и осигуравају да апликација реагује исправно на све улазе, као и да интерфејс адекватно приказује податке.

На пример, један *end-to-end* тест може проверити да ли корисник, након клика на дугме за пријаву, бива успешно преусмерен на другу страницу и да ли се исправна порука о пријави приказује. Ови тестови су важни јер пружају сигурност да ће апликација исправно функционисати у производњи, смањујући ризик од грешака у комплексним корисничким сценаријима.

4.4.8. Angular алатке (Dev Tools)

Angular алатке *DevTools* је користан алат за све који раде у *Angular* окружењу, омогућавајући детаљну анализу и дебаговање апликација. Један од највећих бенефита је преглед хијерархије компоненти кроз *Component Explorer*, који вам омогућава да видите тренутно активне компоненте у апликацији и прегледате њихова стања и податке. На овај начин можете открити потенцијалне проблеме са прослеђивањем података између компоненти или промене у стању које нису очекиване.

Profiler у *Angular DevTools* вам омогућава да мерите перформансе ваше апликације, тако што бележите различите акције корисника и пратите време које је потребно за рендеровање појединачних компоненти. Овај алат је посебно користан за идентификовање „уских грла“ у перформансама апликације, тако да се могу извршити неопходне оптимизације.

Инсталација и коришћење *Angular DevTools* је веома једноставна. Прво, потребно је да уђете у продавницу екстензија вашег претраживача, било да користите *Google Chrome* или *Microsoft Edge*, и пронађете *Angular DevTools*. Након што пронађете екстензију, можете је додати кликом на одговарајуће дугме, и *Angular DevTools* ће се инсталирати, а његова иконица ће се појавити поред траке за адресу у претраживачу. Када је екстензија инсталирана, потребно је подесити ваш *Angular* пројекат тако да искључите опцију оптимизације у *angular.json* датотеци. Ово је важно како би *Angular DevTools* правилно функционисао. Једноставно отворите датотеку и проверите да је *optimization* подешено на *false* у конфигурацији окружења у ком вршите дебаговање.



Слика 4.4.8. – Пример провере тестирања података

Након што подесите *Angular* пројекат и искључите оптимизацију, покрените апликацију у развојном режиму и отворите је у претраживачу. Кликните на иконицу *Angular DevTools* у траци екстензија, што ће отворити панел за дебаговање и анализу. У *Component* можете прегледати хијерархију компоненти, док *Profiler* омогућава праћење перформанси апликације. Коришћењем *Angular DevTools* редовно, лакше ћете откривати грешке и побољшавати ефикасност, што ће резултирати бољим корисничким искуством.

5. ЗАКЉУЧАК

Свака веб апликација за учење као и ова има за циљ да корисницима пружи једноставан и приступачан начин да савладају одређену технологију. Ова апликација је посебно креирана за почетнике како би лакше разумели *Angular framework* на српском, уз детаљна објашњења и примере који покривају све основне концепте. Кроз јасан и разумљив садржај, почетници могу да стекну потребна знања и брзо напредују у развоју.

Архитектура апликације је базирана на компонентном моделу, што омогућава лако одржавање и проширивање функционалности. *Angular Material* библиотека је коришћена за постизање модерног и уједињеног изгледа сајта, док су *CSS* и *TypeScript* помогли у дефинисању структуре и логике апликације. Користећи ове технологије, олакшана је изградња интерактивног корисничког интерфејса који омогућава пријатно корисничко искуство.

Сајт обухвата све неопходне аспекте за учење, укључујући теоријски део, примере из праксе, као и могућност примене стечених знања. Корисници кроз ову платформу добијају потпуно разумевање суштине и стичу основу за даљи самосталан рад на развоју веб апликација користећи *Angular*.

6. ИНДЕКС ПОЈМОВА

А

Angular 6, 7, 8, 25, 29, 34, 37, 42, 50, 55

Angular CLI 12, 21, 26, 41

Angular Material 6, 7, 8, 15, 18, 22, 39, 55

API 48, 50, 51

Архитектура апликације 6, 20, 21

В

Валидација 18, 47

Д

Директиве 32, 33, 34

Детекција промена 42, 43

К

Компоненте 9, 10, 25, 29

Креирање компоненти 29, 30

Комуникација са сервером 43, 44

Л

Lazy Loading 39, 40

М

Модуларност 7, 9

П

Приступачност 39, 40

Преусмеравање 43, 44

Р

Рутирање 11, 43, 44

С

SSR (Server-Side Rendering) 45

Стилови (CSS) 7, 17

Т

Тестирање 47, 48

TypeScript 7, 25

Ф

Форме 14, 42, 43

Х

HTML 7, 25

7. ЛИТЕРАТУРА

- [1] Angular 16 Documentation: <https://v16.angular.io/docs>
- [2] Angular Official Guide: <https://angular.dev/>
- [3] Muhammad, „*Angular Cookbook: Over 80 actionable recipes every Angular developer should know*“ (2020).
- [4] Fireship.io, *Angular Course* (2022): <https://fireship.io/courses/angular/>
- [5] Udemy, *The Complete Guide to Angular* (2023): <https://www.udemy.com/course/the-complete-guide-to-angular-2/>
- [6] W3Schools, *TypeScript Tutorial*: <https://www.w3schools.com/typescript/>

8. ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

Студент (име, име
једног родитеља и презиме):

Игор Жељко Поповић

Број индекса:

НРТ 50/18

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

Веб апликација „Водич за Angular“

1. резултат сопственог истраживачког рада;
2. да овај рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, __. __. 202__ године

Својеручни потпис студента
