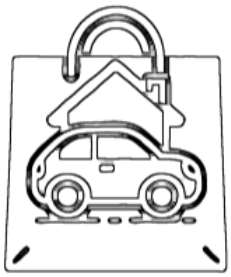
УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ“ ВО СКОПЈЕ

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Дипломски труд на тема

ПАМЕТЕН АГРЕГАТОР НА ОГЛАСИ СО ПЕРСОНАЛИЗИРАНО ИЗВЕСТУВАЊЕ



Reklama8

Изработил:

Дејан Симоновски 211039

Ментор:

проф. д-р Костадин Мишев

Скопје, декември 2025

СОДРЖИНА

АПСТРАКТ

Овој труд претставува развој на веб апликација за автоматско известување за нови огласи - **Reklama8**. Целта на апликацијата е да им овозможи на корисниците брз, едноставен и ефикасен начин за следење на огласи по нивни зададени критериуми, без потреба од постојано пребарување.

Во системот се применети современи технологии и архитектонски принципи. На клиентската страна е користен **Angular**, кој обезбедува динамичен и реактивен кориснички интерфејс преку едностранична апликација (SPA). На серверската страна е имплементиран **Spring Boot**, кој овозможува стабилен и скалабилен backend со REST архитектура, репозиториуми, сервиси и контролери, според **слоевит модел (layered architecture)**.

За безбедност е интегриран **JWT (JSON Web Token)** механизам, кој овозможува сигурна автентикација и авторизација на корисниците при пристап до заштитени ресурси. Дополнително, преку **Spring Boot Starter Mail** и **JavaMailSender**, системот автоматски испраќа е-пошта до корисниците кога се појавува нов оглас што одговара на нивните претплати.

Апликацијата претставува комплетно решение со можност за реална примена, демонстрирајќи интеграција на повеќе современи технологии, добра архитектура и фокус на корисничко искуство. Со ова, Reklama8 претставува практичен пример за модерен, безбеден и автоматизиран веб систем развиен со најновите веб технологии.

1. ВОВЕД

Секој ден има огромен број на нови огласи на најголемите страници за огласување во рамки на Македонија, со тоа се поставува прашањето, како некој може да го најде тоа што го бара, без притоа да го потроши цело свое време фокусирајќи на само таа цел? Одговорот е автоматизација.

Во овој труд ние ќе се обидеме да направиме веб апликација „Паметен агрегатор на огласи со персонализирано известување“, што, би било страница каде се собираат огласи од двете најголеми страници за огласи во Македонија, односно, Reklama5 и Pazar3 во моментот на правење на овој труд и тие огласи ќе бидат обиденети на еден заедничка страница, направена со помош на технологиите Angular, Java SpringBoot па дури и Python во делот скрипти за земање на огласите.

Нашата страница ќе овозможи корисниците да се најавуваат и да внесуваат детали за огласи што ги бараат. Можност за пребарување во иднина, односно, доколку оглас како бараниот не постои, а корисникот селектира да биде известен преку копчето „Notify me“, корисникот ќе биде известен преку електронска пошта кога нов оглас што ги исполнува тие критериуми ќе биде додаден.

Во кратки црти, овој проект има за цел да ги олесни процесите на пребарување и следење на нови огласи, што ќе го направи користењето на онлајн пазарите во Македонија поедноставно и поефикасно.

2. АНАЛИЗА НА ПОСТОЈНИ РЕШЕНИЈА

За да може да навлеземе подлабоко во оваа тема, треба прво да разгледаме што точно има моментално на македонскиот пазар за огласување.

2.1 Преглед на постојни веб апликации за огласи

На прв поглед, страниците имаат сè што би очекувале, односно, лента за пребарување, одбирање на место, категорија, подкатегорија, најава и можности за промовирање и слично.

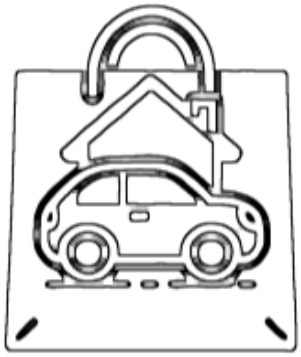
Двете страници се многу слични во својот изглед, модел на работа и начини на профит.

2.2 Недостатоци на постојни решенија

Едно нешто што недостига во двете страници е опцијата за известување доколку се појави некаков оглас којшто ние сме го барале цело ова време. На овој начин се губи премногу време, проверувајќи секој ден дали одреден тип на оглас се појавил, што значи дека само корисниците со доста слободно време би ги зграпчиле сите огласи со поволни цени, а останатите можеби не би приметиле дека огласот воопшто се појавил пред повторно да исчезне.

2.3 Потенцијални подобрувања

За да се реши овој проблем имаш едноставно решение, можност за најавување на страницата при што, ако се направи пребарување за одреден оглас (пр. Golf), а не постои таков, понудени сме со опција „Извести ме“ (Notify me), што би не известило на нашата електронска пошта веднаш штом се вчита таков оглас.



Слика 1. Комбинација од торбицата на Reklama5 со автомобилот и куќата од Pazar3 во значката на Reklama8, симболизирајќи нивен спој.

3. ВОВЕД ВО КОРИСТЕНИ ТЕХНОЛОГИИ

Во ова поглавје ќе дадеме краток преглед на основните технологии што ги користиме во проектот - што се тие, за што служат, и зошто се соодветни во контекст на нашиот систем. Главно ќе опфатиме Angular и Spring Boot, па малку и Python и зошто баш него го користиме за нашиот ползач.

3.1 Angular

Angular е open-source веб фрејмворк базиран на TypeScript, кој овозможува развој на едностранични апликации. Развиен и одржуван е од Google, што гарантира високо ниво

на стабилност, долгорочна поддршка и редовни ажурирања на безбедносни и функционални компоненти. Angular е еден од најкомплетните frontend фрејмворци во индустријата, кој овозможува целосен пристап кон развој на клиентска логика, структурирање на податоци и управување со интеракцијата со корисникот. [1]

Во процесот на избор на frontend технологија, беа разгледани најчесто користените модерни решенија: Angular, React и Vue.js. Иако сите три нудат брз и реактивен пристап кон развој на веб апликации, секој има различна филозофија. React претставува библиотека фокусирана на визуелниот дел на апликацијата и бара вградување на дополнителни библиотеки за routing, state management и dependency injection. Vue.js, пак, се издвојува по својата едноставност и лесно учење, но не нуди корпоративно ниво на поддршка и стандарди како Angular.

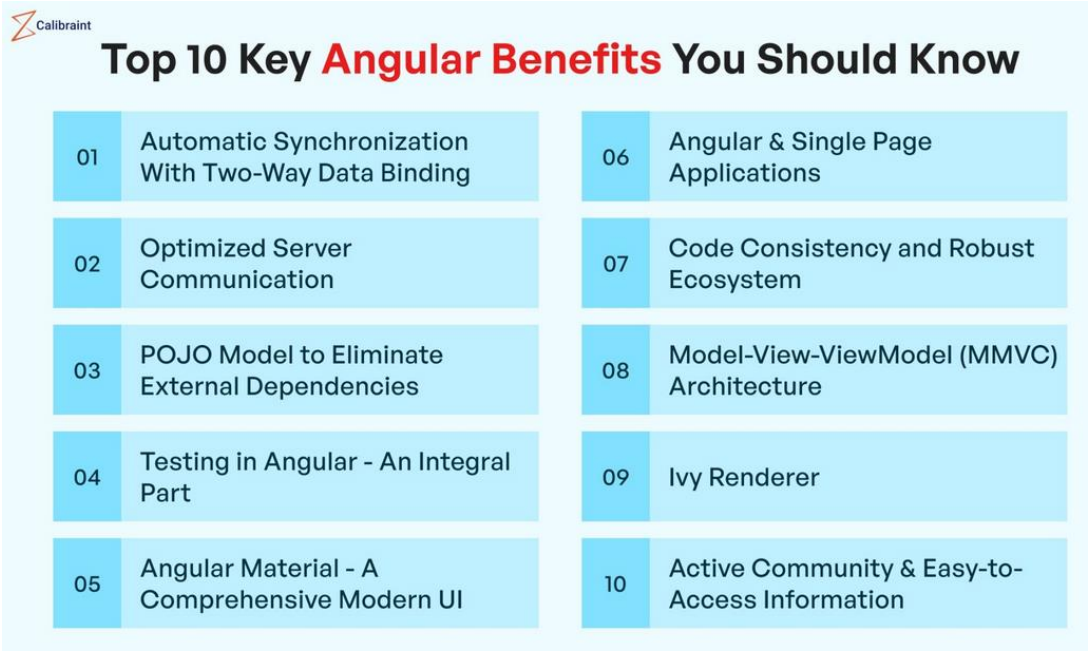
Angular, од друга страна, нуди целовит пристап со вградени решенија за:

- *Routing (управување со навигација помеѓу страници),*
- *Dependency Injection (вбризгување зависности и повторна употреба на компоненти),*
- *Reactive Forms & Validation (напредно управување со форми и проверка на податоци),*
- *HTTP Client (лесна комуникација со backend преку REST API),*
- *State Management (управување со состојбата на податоците),*
- *Вградено тестирање (unit и integration testing),*
- *Angular CLI (Command Line Interface) за автоматизација на генерирање компоненти, сервиси и модули.*

Овие карактеристики беа особено корисни при развојот на апликацијата, затоа што овозможија структуриран, модуларен и одржлив код, со јасна поделба помеѓу компонентите и сервисите. Благодарение на Angular, фронт-енд делот на апликацијата обезбедува интуитивен и динамичен кориснички интерфејс, кој лесно се поврзува со backend преку REST повици.

Angular исто така обезбедува силна „типизација“ преку TypeScript, што ја намалува можноста за грешки при развој, а системот за data binding и component-based архитектурата овозможуваат лесно одржување и проширување на функционалностите.[2]

Со тоа, Angular претставува идеален избор за овој проект, бидејќи комбинира високи перформанси, стабилност, безбедност и одлична интеграција со backend REST API развиен во Spring Boot.



Слика 2: Придобивки на користење Angular

3.2 SpringBoot

Spring Boot е Java фрејмворк со отворен код, кој претставува надградба на познатиот Spring Framework и е наменет за развој на продукциски подготвени апликации со минимална конфигурација. Развиен е од Pivotal Software (денес дел од VMware), со цел да го поедностави процесот на создавање серверски апликации и микросервиси преку автоматска конфигурација и вградено управување со зависности.

Spring Boot овозможува развивање на самостојни backend апликации кои можат веднаш да се стартуваат, без потреба од надворешен сервер, благодарение на вградените embedded сервери (како Tomcat, Jetty или Undertow). Овој пристап ја елиминира сложеноста на класичните Java EE решенија, каде што конфигурацијата беше долга и рачна, и овозможува брз почеток и флексибилно развивање на backend REST API сервиси.[3]

Во процесот на избор на backend технологија, беа разгледани повеќе современи решенија — Django (Python), Express.js (Node.js), Laravel (PHP) и .NET Core (C#). Иако секое од овие решенија има свои предности, Spring Boot беше одбран поради неговата зрелост, стабилност, интегрираност и сигурност, што е особено важно за систем кој комуницира со повеќе извори на податоци, извршува скрипти и испраќа автоматски известувања.

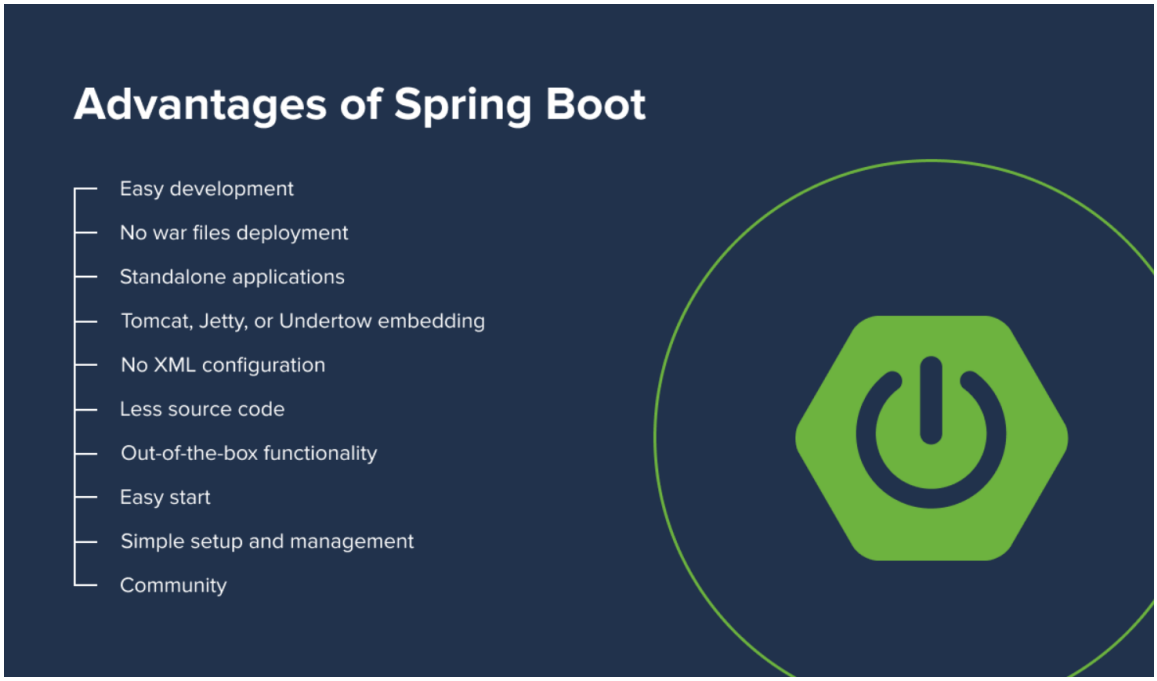
Клучните причини за избор на Spring Boot се[4]:

- *Auto-configuration – автоматска конфигурација на зависности и компоненти;*
- *REST поддршка – лесно дефинирање на REST контролери и JSON комуникација;*
- *Интеграција со бази на податоци преку Spring Data JPA и Hibernate;*
- *Безбедност и автентикација преку Spring Security и JWT (JSON Web Token);*
- *Поддршка за надворешни библиотеки, како што се Python скрипти, SMTP сервери, и JavaMailSender за испраќање email пораки;*
- *Вградени тестови и мониторинг, со што се олеснува одржувањето;*
- *Силна заедница и одлична документација, што го прави идеален за академски и професионални проекти.*

Во рамките на овој проект, Spring Boot ја извршува улогата на централен backend сервис кој:

- *ја обработува логиката за управување со огласи,*
- *ги поврзува податоците прибрани преку Python crawler,*
- *ги зачувува резултатите во база,*
- *и преку Spring Boot Mail сервис испраќа автоматски email нотификации до корисниците.*

Овој интегриран пристап резултира со робустен, модуларен и сигурен систем, кој лесно може да се прошири и одржува во иднина. Со тоа, Spring Boot ја демонстрира својата сила како основа за скалабилни и стабилни backend решенија, кои можат да се применат и во големи, продукциски системи.



Слика 3: Придобивки од користење SpringBoot

4. ДИЗАЈН И СПЕЦИФИКАЦИЈА НА СИСТЕМОТ

При дизајнот на овој систем се користени повеќе технологи, кои работаат заедно за да се постигне крајниот продукт, нашиот агрегатор на огласи.

4.1 Функционални барања

- *Регистрација и најавување на корисници*
- *Пребарување и филтрирање на огласи*
- *Прикажување на детали за оглас*
- *Автоматско собирање на огласи*
- *Претплата за известување*
- *Известувања по електронска пошта*

4.2 Нефункционални барања

- *Системот треба да ги земе и прикаже огласите од двете страници, зависно од прилагодувањата во скриптите, во рок од неколку секунди.*
- *Архитектурата треба да овозможи лесно додавање на нови извори на огласи*
- *Податоците за лозинките треба да бидат шифрирани*
- *Пристапот до API-то за известувања треба да биде ограничен за неавторизирани корисници.*
- *Системот треба да биде достапен 24/7 со минимални прекини.*

- *Да биде прилагоден за различни уреди (responsive design).*
- *Кодот треба да биде модуларен и добро документиран.*

4.3 Фази во развој на системот

За остварување на главната цел, дефинирани се неколку фази:

- *Интеграција повеќе извори на податоци*
- *Обработка и складирање на податоците*
- *Изградба на API за размена на податоци*
- *Авторизација*
- *Развој на кориснички интерфејс*
- *Имплементација на систем за известување*

Секој од овие чекори има своја улога во крајниот производ и ќе ви даде подобро познавање за тоа како се движеше изградбата на овој систем.

Интеграција на повеќе извори

За да се интегрираат повеќе извори во едно, направен е craweler(ползач) во програмскиот јазик Python. Тој се повикува во рамки на SpringBoot back-end апликацијата на секој 4 часа и ги зема најновите огласи.

```
@Scheduled(fixedDelay = 14400000) // Every 4 hours
public void fetchAndSaveListings() {
```

Слика 4. Приказ на автоматското повикување на функцијата за земање на огласи

Обработка и складирање на податоците

Ползачот оди по внесените страници и ги зема огласите според нивните HTML елементи, класи, ид, итн. После тоа сите огласи ги додава во една JSON(JavaScript Object Notation) датотека, од каде нашата Java апликација има пристап до овие податоци. Сите останати информации како корисници и известувања, ние ги чуваме во посебна база на податоци.

```
{
    "title": "OPEL  ASTRA H 1.4 TwinPort",
    "price": "3 999 ЕУР",
    "location": "Битола",
    "category": "Автомобили",
    "image": "https://media.pazar3.mk/Image/038d2fe1-b69b-4621-95b6-2e2d0e18c085/20251
    "link": "https://www.pazar3.mk/oglas/vozila/avtomobili/opel/Astra/prodazba/bitola/
    "time": "Денес 18:09",
    "source": "Pazar3"
},
```


Слика 5. Пример за JSON објект во рамки на системот

Причината зошто овие податоци за огласи не се додаваат директно во база е заради природата на овие огласи и нивната комплексност, односно, некој огласи повеќе пати се внесуваат, други останууваат многу долго време, дури и години а производот е одамна продаден, поради што ги прикажуваме само најновите огласи.

Изградба на API за размена на податоци

Нашиот front-end, односно, убавиот преден дел на нашата апликација што го гледаат крајните корисници е изграден во Angular, но, тој не може да работи без да биде снабден со информации, за тоа ни е потребен нашиот back-end во Java SpringBoot. Тука е направен контролер за справување со авторизација, како и контролер за справување со огласи.

Тука имаме можности за авторизација со помош на JWT(JSON Web Tokens) токени, кои се користат при најава, регистрација и претплатување на одреден оглас.

Авторизација

За да се постигне авторизација, во базата на податоци на серверската страна се чуваат корисниците, како и нивните претплати. Кога некој корисник не е пронајден во базата на податоци, тој се регистрира и ќе биде додаден во истата. При што, автоматски ќе биде најавен од системот и ќе му биде доделен токен.

Подетално објаснето, процесот се одвива кога корисникот се најавува во системот преку своето корисничко име и лозинка, апликацијата го испраќа барањето до back-end серверот. Доколку најавата е успешна, серверот генерира **JWT токен**, кој содржи информации за корисникот, како што се:

- *неговото корисничко име,*
- *улога*
- *време на истекување (expiration time). и др.*

Токенот се потпишува со тајна вредност(secret key) позната само на серверот, што спречува негова измена од надворешни лица. Генерираниот токен се враќа до клиентската апликација (Angular) како дел од одговорот на серверот. Angular го зачувува токенот во localStorage, зависно од безбедносните поставки.

При секое наредно барање кон серверот (на пример, при пребарување огласи или додавање нови претплати), клиентот го вклучува JWT токенот во HTTP header делот од барањето.

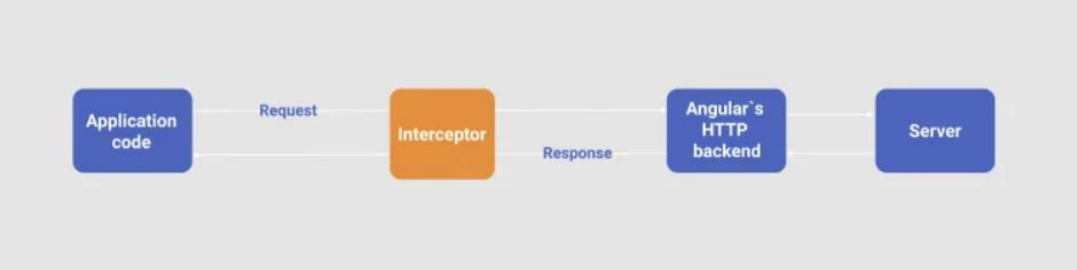
```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const token = this.authService.getToken();
    if (token) {
      request = request.clone({
        headers: request.headers.set("Authorization", `Bearer ${token}`)
      });
    }
    return next.handle(request);
  }
}
```

Слика 6. Interceptor кои во сите наши барања до серверот го додава JWT токенот

И секако, секое пристигнато барање на backend-от се филтрира преку JWT Authentication Filter, каде, се проверува дали токенот е автентичен, дали е истечен, дали содржи валидни податоци, доколку сето тоа е во ред, се дозволува пристап до серверот, во случајот, претплатата на одреден оглас.

Interceptors

Angular Interceptors(пресретнувачи) се многу моќни алатки што можат да се користат за управување со тоа како апликациите ги обработуваат HTTP барањата и нивните одговори. Тие играат клучна улога во имплементацијата на функции како што се евидентирање, автентикација, справување со грешки и друго, што доведува до појасен код, што секако е и полесен за оддржување.[6]



Слика 7: Шематски приказ за тоа како функционира пресретнувач

Развој на кориснички интерфејс

За да се прикаже содржината на корисникот, потребен е front-end дел, во нашиот случај тоа е Angular, кој е модерен framework(фрејмворк) развиен од Google, кој овозможува развој на динамички, едностранични веб апликации каде што промените на содржината се изведуваат без целосно повторно вчитување на страницата, што значително го подобрува корисничкото искуство.

Дополнително, достапна е двонасочна комуникација на податоци, со што може да се освежат податоците без притоа да се направи повторно отворање на целата страница во истиот момент кога податоците ќе се сменат на back-end што значително го подобрува корисничкото искуство.

За да комуницираме со back-end користиме RESTful API, што овозможува лесна комуникација со Spring Boot серверот преку HTTP барања.

```
return this.http.get<any>(this.apiUrl, { params }).pipe(
  map((response) => {
    return {
      data: response.data || [],
      total: response.total || 0,
      page: currentPage,
      pageSize: pageSize,
      location: location
    } as ListingResponse
```

Слика 8. Праќање на барање од Angular до Spring

Со тоа, се земаат сите огласи и соодветно се прикажуваат на корисникот преку циклус, односно, секој оглас оди во своја компонента „listing“ со сопствена HTML структура.

Имплементација на систем за известување

Кога корисникот нема да го најде бараниот оглас, тој е понуден со копче „Извести ме“ (Notify me), при притиснување на ова копче сме пречекани со грешка, за да се користи

оваа функционалност потребна е најава, како тоа се одвива е наведено погоре со JWT токени.

Откако корисникот ќе се најави и повторно ќе го притисне копчето, ќе добие известување со текст „ Notification subscribed successfully. You will be notified if a similar listing appears within 30 days.“, во превод „Известувањето е успешно додадено. Ќе бидете известени доколку сличен оглас се појави во следните 30 дена.“. Но зошто тие 30 дена? Ако таков оглас се појави, корисникот ќе биде известен по електронска пошта и ќе биде избришана претплатата за огласот, но што ако никогаш не се појави? Поради тоа сите претплати на некој начин имаат TTL(Time to live), рок од 30 дена.

Таа претплата се додава во базата на податоци и при секое вчитување на нови огласи, се врши проверка дали има застарени претплати, дали некој оглас соодвествува на некоја претплата, па истите да бидат избришани од базата и секако корисниците известени доколку таква е пронајдена.

Доколку таков оглас се појави, тогаш корисникот ќе биде известен преку електронска пошта, но, како се прави сето тоа?

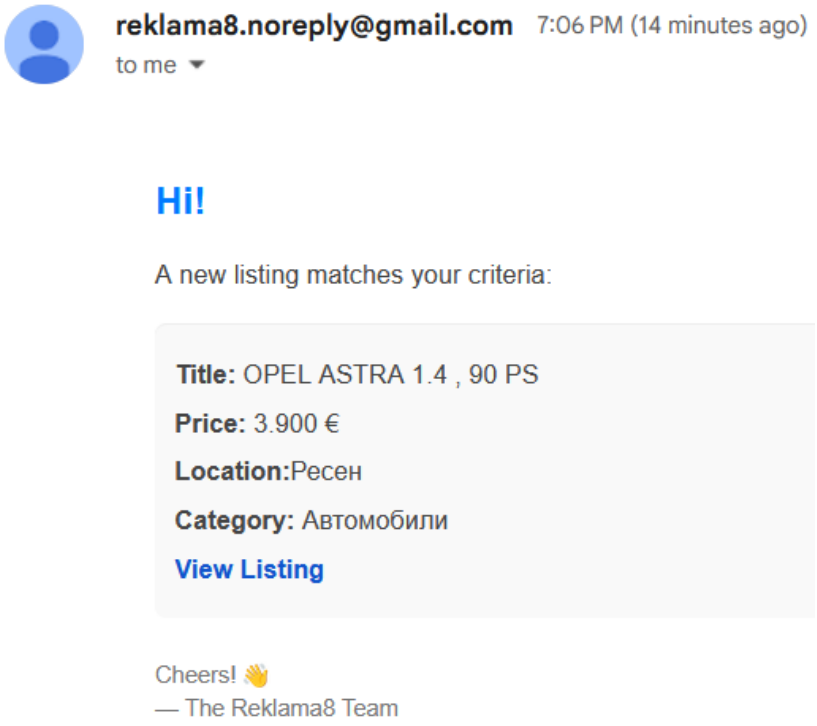
Тоа се прави преку SMTP протоколот, имаме посебна електронска пошта *reklama8.noreply@gmail.com* создадена за потребите на овој проект, поставен во application.properties на back-end делот од апликацијата.

```
#Mail
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=reklama8.noreply@gmail.com
spring.mail.password=**** **** **** ****
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com
```

Слика 9. Подесување на електронска пошта во Spring

Електронската пошта е поставена, форматот на пораката автоматски се пополнува кога тој оглас што го бараме ќе се појави, на крај се испраќа до крајниот корисник со цел да биде навремено известен. Пораката е направена со помош на MimeMessage, истото може да се направи и со SimpleMailMessage, разликата е во тоа што MimeMessage дозволува пораките да бидат стилизирани со помош на HTML и CSS, за разлика од

SimpleMailMessage што е само текст и нема можности за „шминкање“ на пораката и нејзино разубавување.



Слика 10. Пример пристигнато известување за оглас

4.4 Апликациски дел

Апликацијата која ја гледа корисникот е направена во Angular, таа го пристапува серверот за да ни прикаже информации за огласи, валидира најава, регистрација, и пренесува податоци за огласи за кој сакаме да бидеме известени до серверот.

Апликацијата е нашата „телефонска линија“ до SpringBoot серверот.

Односно, таа го овозможува следното:

- *Прикажување на огласи*
- *Валидација на корисници*
- *„NotifyMe“ функционалност*
- *Пагинација на огласи*

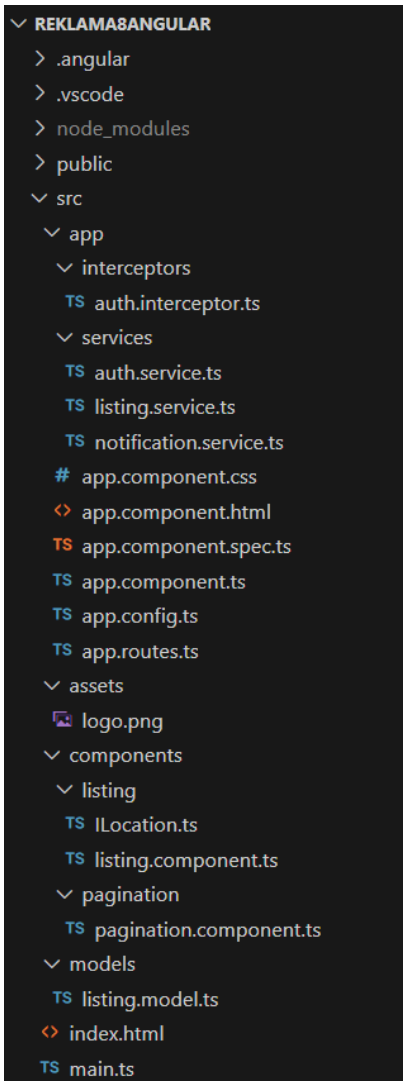
Тука се доведува до прашање, ако Angular врши толку многу одговорности, каква е улогата на SpringBoot серверот? Тоа е едноставно.

Angular само прави мали процесирања на податоците и испраќање на податоци, вистинското процесирање, складирање и сета логика се наоѓа на серверот.

Но, што точно прави делот апликација во претходно наведените точки?

Апликацијата не е способна сама да извршува бизнис логики, да прави складирање на податоците во база или какви било позадински процеси.

Но, сепак може да врши мали процесирања од типот, кога ќе испратеме JSON, тој го распакува на страната на клиентот и соодветно го прикажува, тој исто така може да чува JWT токени локално кај корисникот за да серверот не треба да складира податоци за секој најавен корисник.



Слика 11: Структура на апликациски дел

4.5 Сервел дел

SpringBoot престатува скалабилен сервер дел кој е базиран на Java. За разлика од Angular, тој може да врши доста процесирања, како и позадински задачи како вршење на функции

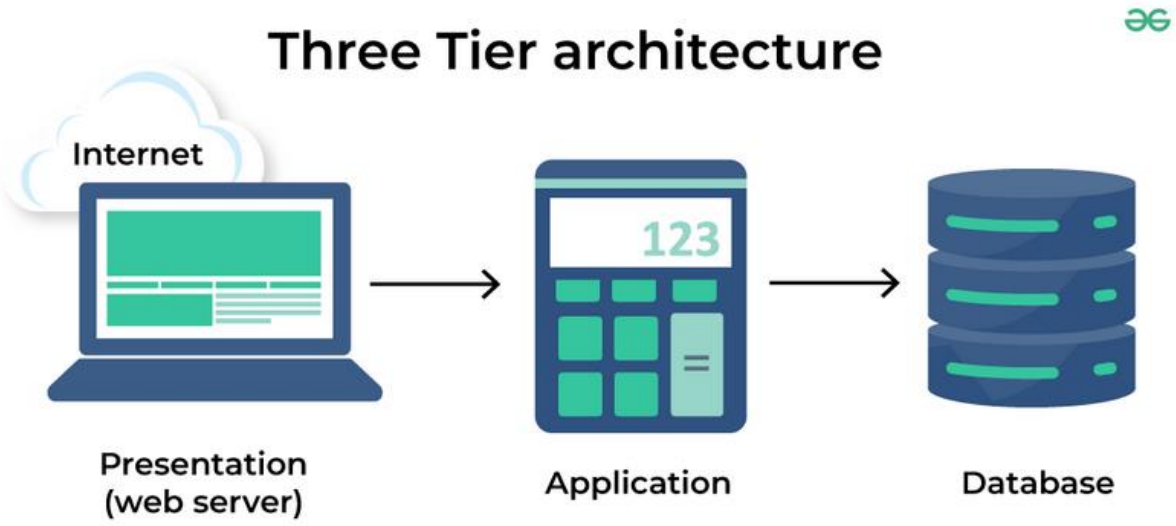
на одредени временски интервали, како на пример, собирање на нови огласи на секој 4 часа и испраќање на соодветни известувања.

Во овој дел од апликацијата се случува цела логика, односно, тука се регистрираат корисници и се врши нивно зачувување во база, се комуницира со различните скрипти за земање на огласи, се генерираат токени за автентикација на корисниците, како и зачувување на претплати за известување.

Слоевита архитектура

Во серверскиот дел се користи т.н слоевита архитектура која се состои од следното

- *Контролер*
- *Сервис*
- *Репозиторум*
- *Модел*



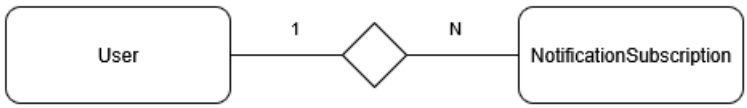
Слика 12: Слоевита архитектура

Контролерите се одбележани со ознаката **@RestController** бидејќи не се обични контролери, односно, не враќаат каков било резултат, туку JSON/XML.

Во контролер делот се примаат сите HTTP барања и се врши нивно процесирање и одговор. Во рамки на овие контролери се повикуваат сервиси, одбележани со **@Service**, тие кој ја содржат бизнис логиката на апликацијата, како проверка за постоечки корисник, споредба на нови огласи со критериуми за известување, итн...

За да може да се вчитуваат и зачувуваат податоци од база, се користи Репозиториум дел, всушност слој за пристап до базата на податоци, преку него се одвиваат сите CRUD операции, односно Create, Read, Update и Delete. Тој се означува со **@Repository** и прави имплементација на JpaRepository, кој овозможува лесен пристап до податоци, без рачно пишување на SQL барања.

Секако, во цел овој процес се доста битни и моделите, означени со нотацијата @Entity кој ни кажуваат какви податоци чуваме и каква е нивната структура. Тука се дефинираат релациите између податоците, односно, како тие комуницираат помеѓу себе.



Слика 13: Релации помеѓу ентитетите

Во нашиот случај, има само една релација помеѓу објектите од типот User(корисник) и NotificationSubscription(Претплата за оглас). Јасно се гледа дека еден корисник може да има повеќе претплати, и се разбира, една претплата може да има само еден корисник од кој таа е направена.

Оваа релација е класичен пример на **one-to-many (еден-кон-многу)** однос. Тоа значи дека **Корисникот** е „еден“ во односот.Претплатата е „многу“ во односот.

Во термини на бази на податоци, ова се реализира така што NotificationSubscription табелата ќе има странски клуч (foreign key) кој упатува на User табелата. Во овој конкретен случај, колоната user во NotificationSubscription ќе покажува на корисникот.

Други ентитети, како што се самите Listings(огласи), не се зачувуваат во база и поради тоа се изолирани, немаат релации со другите ентитени. Како што и наведовме претходно, тие се зачувуваат во JSON датотека заради нивната природа на брзо менување и комплексноста од нивна унификација поради повеќе извори, и можноста за додавање на уште страници од кои би можеле да се собираат огласи.

Користени библиотеки

Во рамки на развојот на системот Reklama8 се користат неколку дополнителни библиотеки кои ја олеснуваат имплементацијата на функционалностите и ја подобруваат читливоста, безбедноста и одржливоста на кодот. Би сакале да ги одвоиме следните:

- *Lombok*
- *JJWT*
- *Spring Boot Starter Mail*

Првата библиотека е Lombok, тој го автоматизира генерирањето на често користен код во Java класите. Со користење на анотации како @Data, @Getter, @Setter, @AllArgsConstructor, @NoArgsConstructor и други, се елиминира потребата за рачно пишување на повторлив код како гетери, сетери и конструктори.

Со тоа имаме повеќе придобивки, како намалување на boilerplate кодот, подобра читливост и одржливост на класите и секако автоматска генерација на методи при компилација.

```java
package mk.reklama8.model;

import jakarta.persistence.*;
import lombok.Data;

@Data  11 usages  dejan-simonovski +1
@Entity
public class Listing {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String price;
    private String location;
    private String category;
    private String image;
    private String link;
    private String time;
    private String source;
}
```

Слика 14: Употреба на @Data во Listing класата

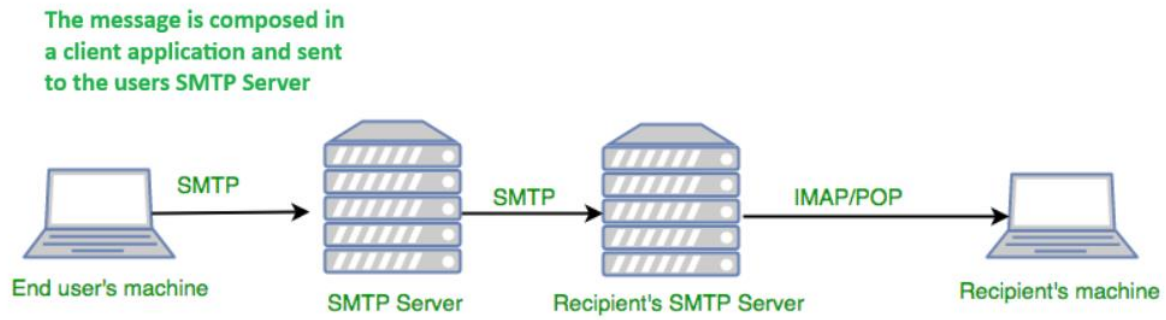
JJWT - io.jsonwebtoken е библиотека која овозможува работа со JSON Web Tokens (JWT), стандарден начин за безбедна размена на информации помеѓу клиент и сервер. Во системот Reklama8, JWT се користи за автентикација и авторизација на корисниците.

JWT токенот содржи потпишани податоци за корисникот (на пример, неговото корисничко име и улога) и се испраќа со секое барање до серверот за да се потврди идентитетот.

Главни компоненти на **JWT библиотеката (io.jsonwebtoken)**:

- *jjwt-api – дефинира интерфејси за креирање и читање токени*
- *jjwt-impl – имплементација на API-то*
- *jjwt-jackson – овозможува JSON обработка преку Jackson*

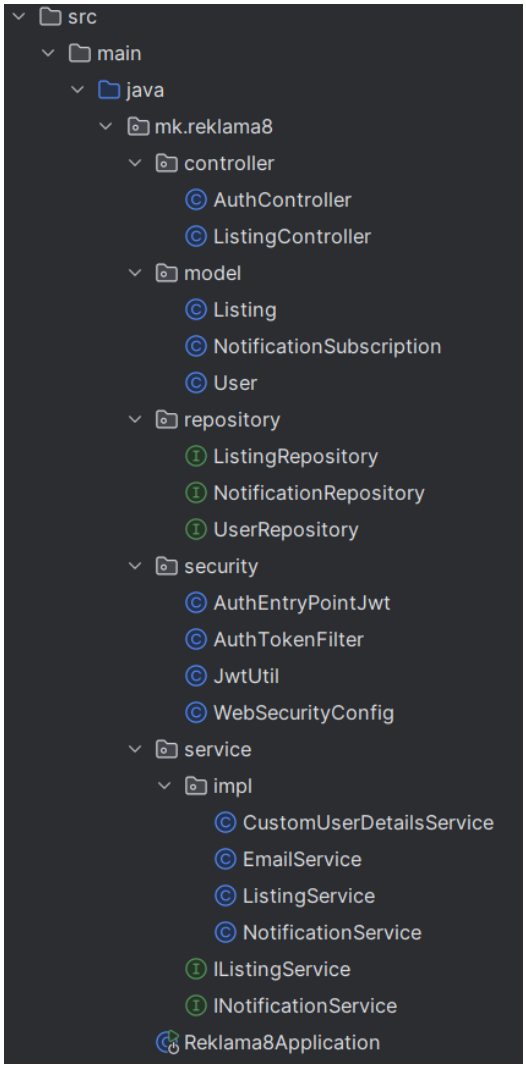
Spring Boot Starter Mail се користи за испраќање email пораки од апликацијата. Во проектот Reklama8, оваа библиотека овозможува системот да испраќа известувања до корисниците, кога ќе се појави нов оглас што одговара на нивните претплати.Spring Boot го користи JavaMailSender интерфејсот за испраќање MIME и текстуални пораки.



Слика 15: Како се испраќа една порака до корисникот

Класата EmailService ја инјектира зависноста JavaMailSender преку конструктор и ја користи за да креира и испраќа MIME пораки. Spring Boot автоматски го конфигурира JavaMailSender врз основа на вредностите поставени во application.properties.

Во оваа конфигурација, корисничката лозинка не се внесува директно, туку, на таа корисничка сметка се вклучува дво-факторна автентикација и се генерира app password, со кој, апликацијата може слободно да испраќа пораки до корисниците на Reklama8 при појава на соодветен оглас, без притоа да е потребен код од телефон, пристап до вистинската лозинка, или каква било интервенција од човек.



Слика 16. Слоевита архитектура во серверот на Reklama8

JavaMail API дефинира класи што ги претставуваат компонентите на поштенскиот систем. JavaMail не имплементира е-пошта сервер, туку ви овозможува пристап до е-пошта сервер користејќи Java API. За да го тестирате кодот, мора да имате пристап до е-пошта сервер. Иако спецификацијата JavaMail API не налага поддршка за специфични протоколи, JavaMail обично вклучува поддршка за POP3, IMAP и SMTP.[7]

Дополниелно може да запримете и дел за безбедност(security), тука се прави автентикација на сите барања кои доаѓаат на патеката listings/notify, со што не се дозволува никој што не е најавен да побара да биде известен за појавување на одреден тип на оглас.

4.6 Скрипта за собирање на огласи

Оваа скрипта е главна за земањето на огласите од сите страници, без него, нашата страница би немала смисла, би била само бела позадина.

Тој оди од страница до страница и зависно од нивните HTML елементи или нивните класи/ИД ги зема означените делови од страниците, како на пример

```
for ad in soup.find_all("div", class_="ad-desc-div"):
    title_elem = ad.find("a", class_="SearchAdTitle")
    price_elem = ad.find("span", class_="search-ad-price")
```

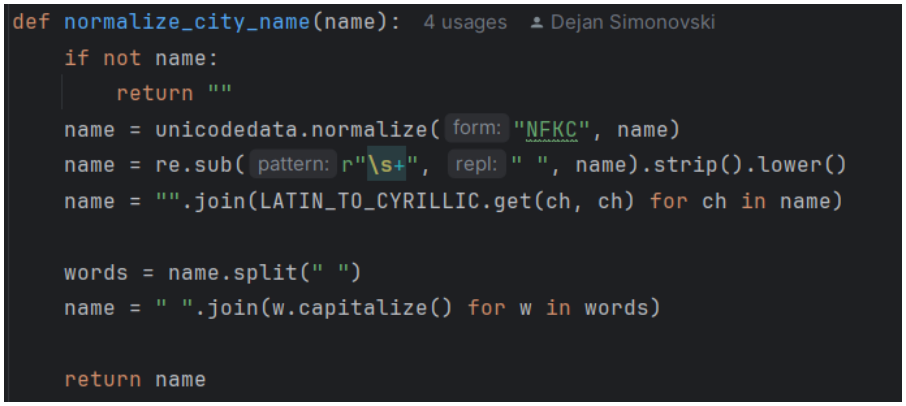
Во овој пример, одиме низ сите реклами на пазар3(огласи), означени со класа ad-desc-div, при што на секој оглас му ги земаме неговиот наслов означен со SearchAdTitle и неговата цена, односно, search-ad-price. Сето тоа се складира и се оди на следниот оглас, додека не стигнеме до крајот на нашето пребарување. Колку страници ќе бидат пребарувани и собирани зависи од која бројка е внесена во нашата SCRAPE_LIMIT променлива во main делот на скриптата.

```python
if __name__ == "__main__":
    SCRAPE_LIMIT = 20 # Number of pages to scrape from each site
    results = scrape_multiple_pages(scrape_pazar3, SCRAPE_LIMIT) + scrape_multiple_pages(scrape_reklama5, SCRAPE_LIMIT)

    print(f"Total listings scraped: {len(results)}")

    with open("listings.json", "w", encoding="utf-8", errors="replace") as file:
        json.dump(results, file, ensure_ascii=False, indent=4)

    print("Results saved to 'listings.json'")
```

Слика 17. Како се повикува делот за земање на огласи

Сепак, при ваквото земање на огласи од различни страници, секогаш има проблеми од типот, различно запишани градови како Kicevo / Kichevo / Кичево или Radovis / Radovish / Радовиш, поради што ние градовите ги влечеме од серверот (тоа се прави вака за да ако има промена подоцна, ги менуваме само на едно место наместо на три.), така имаме унифицираност на имињата, па дури и да имаме подопштини, Скопје секогаш ќе ги покажува и нив.

Уште еден од поголемите предизвици кои може да ги наброеме се грешките во записи, како, Скопје и Скопје, можеби не се приметува но првиот запис има латинично j, а вториот запис има кирилично. Тука настанува проблем во пребарување, поради што е потребна нормализација пред понатамошно процесирање.



Слика 18. Нормализација на имиња на градови

Во серверскиот дел, градовите се прикажуваат на латиница, но, серверскиот дел го зема нивниот кириличен формат бидејќи така и се користи на страниците за огласи, доколку тој не е пронајден, се проверува и за латиничната форма, поради тоа што градовите се зачувани како парови во делот сервер, како на пример:

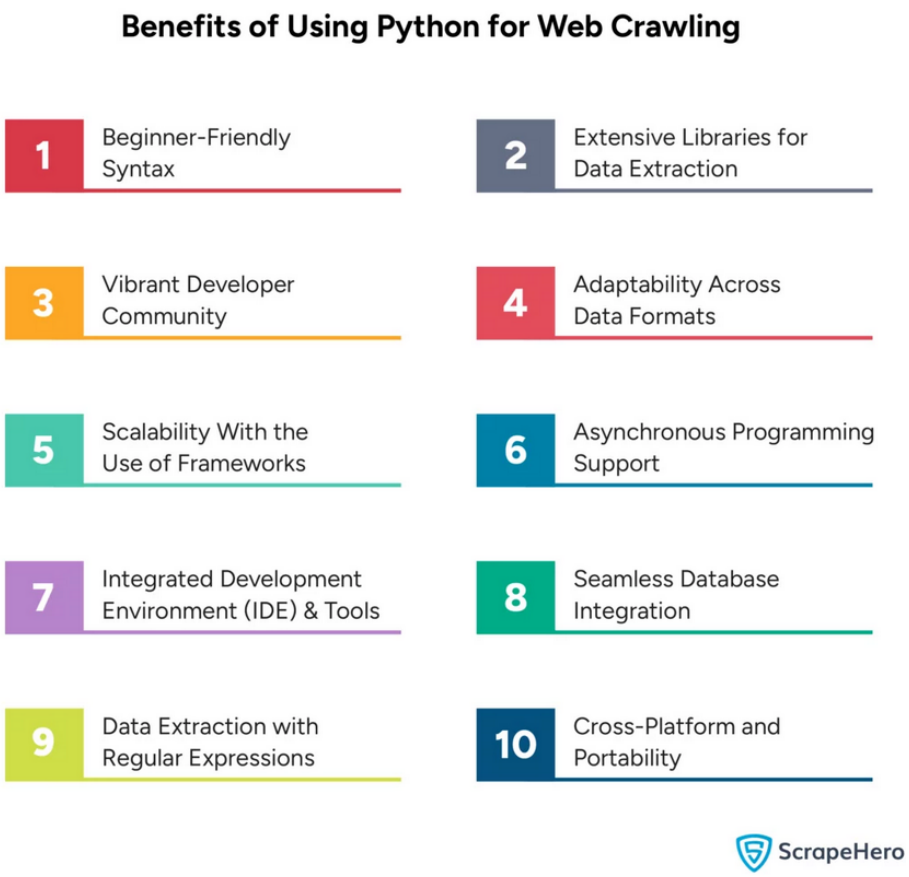
```
[
    { "cyrillic": "Охрид", "latin": "Ohrid" },
    …
    { "cyrillic": "Ресен", "latin": "Resen" }
]
```

Python

Python е популарен избор за web scraping поради својата едноставна синтакса и сестраната документација. Обемните библиотеки како BeautifulSoup и Selenium овозможуваат лесно парсирање на HTML и интеракција со динамични веб-страници,

поддршката за асинхронско програмирање преку asyncio и aiohttp овозможува ефикасно ракување со голем број на барања, лесната интеграција со различни бази на податоци го прави Python погоден за складирање на собраните огласи, поддршката за различни формати на податоци како HTML, XML, JSON и PDF го прави флексибилен за различни задачи на crawling.[5]

Овие карактеристики го прават Python идеален за развој на скрипти за собирање огласи, како што е случајот во нашиот проект.



Слика 19. Предности од користење на Python за crawler(ползач)

4.7 Комуникација помеѓу апликација и сервер

Кога корисникот ќе пристапи на страницата, се вчитуваат огласите од SpringBoot серверскиот дел, серверот преку контролерот за огласи(Listings Controller) пристапува до Python скриптата, crawler(ползачот), кој почнува со процесирање, одејќи по сите страници и собирајќи огласи, во овој случај само Reklama5 и Pazar3, иако повеќе страници може лесно да бидат додадени понатаму само со менување на оваа скрипта.

Откако ќе заврши процесот на собирање огласи, тие се зачувуваат во listings.json датотеката, која се вчитува во контролерот за огласи, се процесира и се враќа назад до Angular делот, огласите се распакуваат во убав формат за приказ на корисникот.

При листање на огласите, се испраќа информација за страница и број на огласи за да не се прикажуваат сите огласи одеднаш, односно, ова се прави за да има читливост на податоците како и оптимизација.

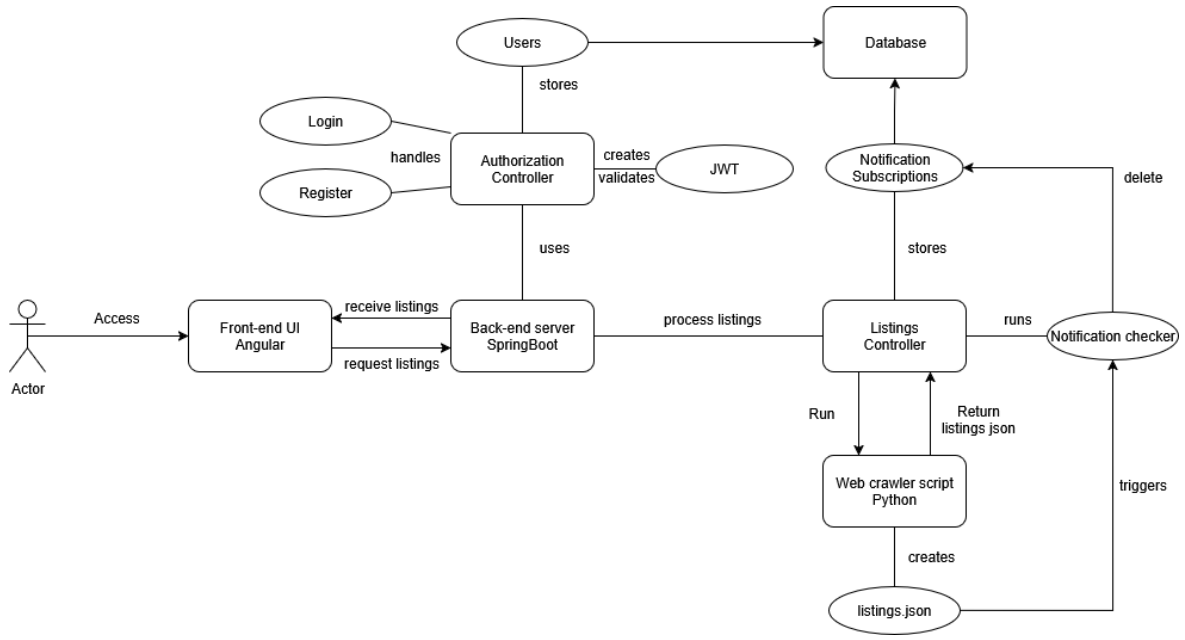
Доколку корисникот не пронајде некаков оглас што го пребарувал, понуден е со опција за да биде известен кога таков оглас би се појавил, при кликнување на ова копче,

повторно се пристапува до контролерот за огласи, тој креира нова претплата за известување која се складира во база.

За да има пристап до таа функционалност, корисникот мора прво да биде најавен, секако.

Тој е пренасочен до контролерот за авторизација, кој се справува за создавање на JWT токени, нивна валидација, како и најава и регистрација на корисници, од тука при регистрација корисникот, тој е автоматски најавен и доделен JWT токен, кој се испраќа назад до серверот при секое авторизирано барање од страна на Angular, со што се докажува идентитетот на корисникот. Сите корисници се складирани во базата на податоци за понатамошна употреба во претплатите за известување.

При секое вчитување на нови огласи, креирањето на нова JSON датотека предизвикува контролерот на огласи да проверува дали некој од новите огласи се совпаѓа со некоја од претплатите, доколку тоа е вистина, се испраќа електронска порака и се прави бришење на таа претплата од базата на податоци, доколку не се најде таков оглас во рок од 30 дена, исто се прави бришење на тој запис од база.



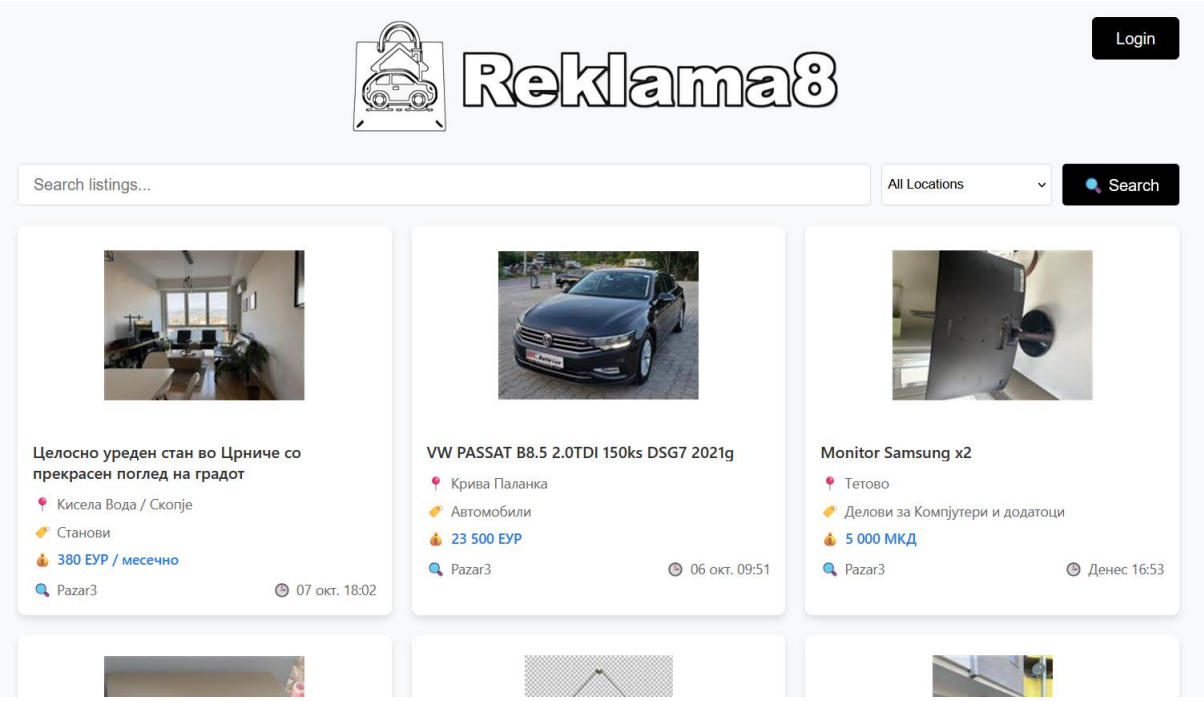
Слика 20. Комуникација помеѓу компонентите

5. Кориснички сценарија

Во овој дел се прикажани најважните кориснички сценарија од функционирањето на веб апликацијата Reklama8. Целта е да се демонстрира начинот на интеракција помеѓу корисникот и системот, како и да се прикаже практичната примена на функциите развиени во претходните поглавја. Преку следните примери се прикажува целиот тек на користење, од пребарување на огласи, најавување и претплата, до обработка на грешки и испраќање известувања по електронска пошта.

5.1 Преглед на огласи

При пристапување кон главната страница на системот, корисникот веднаш се среќава со интерфејсот за пребарување огласи. Горниот дел на страницата содржи полиња за пребарување по клучен збор и локација, со што се овозможува динамична и интуитивна навигација низ базата на достапни огласи. Резултатите се прикажуваат веднаш под полето за пребарување, во форма на картици во кој секоја содржи наслов, локација, категорија, цена и изворна страница од која е извлечен, со клик на огласот, огласот не редиректира до оригиналната објава на изворната страница каде може да се стекнеме со повеќе информации, како на пример детали за контакт.



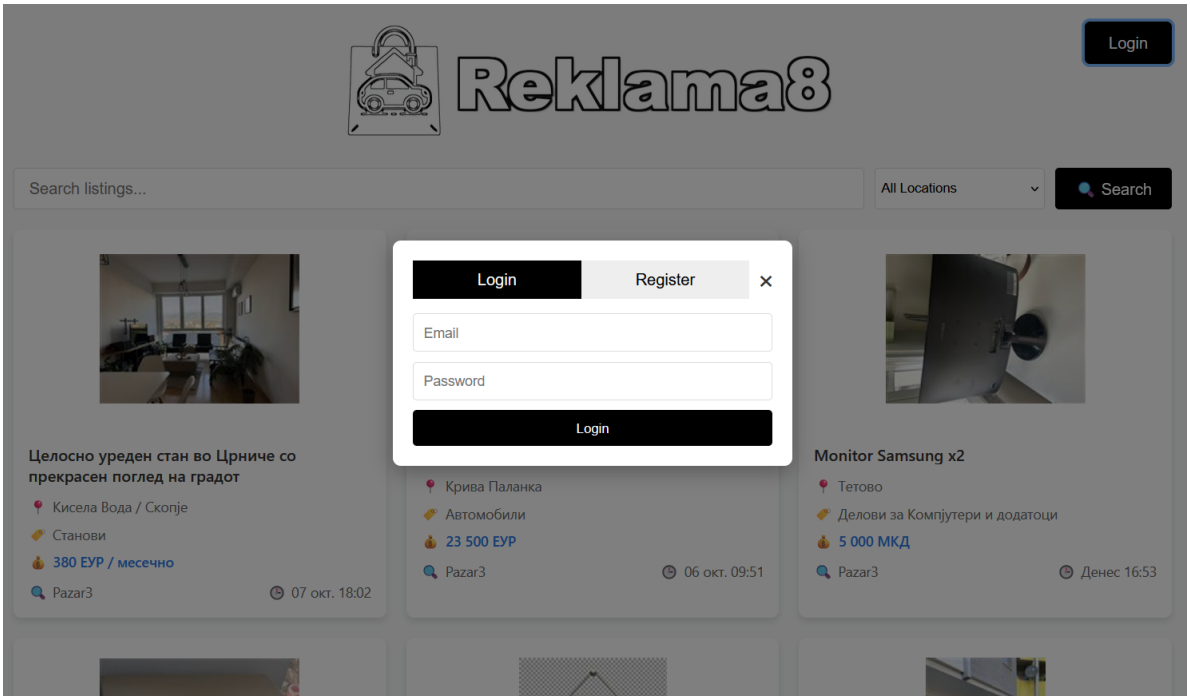
Слика 21: Почетна страница на Reklama8

Секој оглас е линкуван кон оригиналниот извор, со што се овозможува директен пристап до целосниот опис и контакт информации. Овој дел е изграден во **Angular**, каде што податоците се повлекуваат преку **HTTP барање кон backend API-то**. Spring Boot backend сервисот ја враќа JSON листата на огласи, која се прикажува на фронтендот користејќи структурирани компоненти.

Овој интерфејс е дизајниран така што дури и без најавување, секој корисник може да пребарува и да се информира за достапните огласи. Меѓутоа, за користење на напредни функционалности како зачувување претплати, е потребна автентикација.

5.2 Најава на корисник (Login/Register Modal)

Кога корисникот ќе се обиде да пристапи до функционалност што бара најавување, на пример претплата за известувања, системот прикажува **модален прозорец за најава**. Овој модал (popup) овозможува внесување на корисничко име и лозинка, без потреба од напуштање на тековната страница.



Слика 22: Модал за најава/регистрација

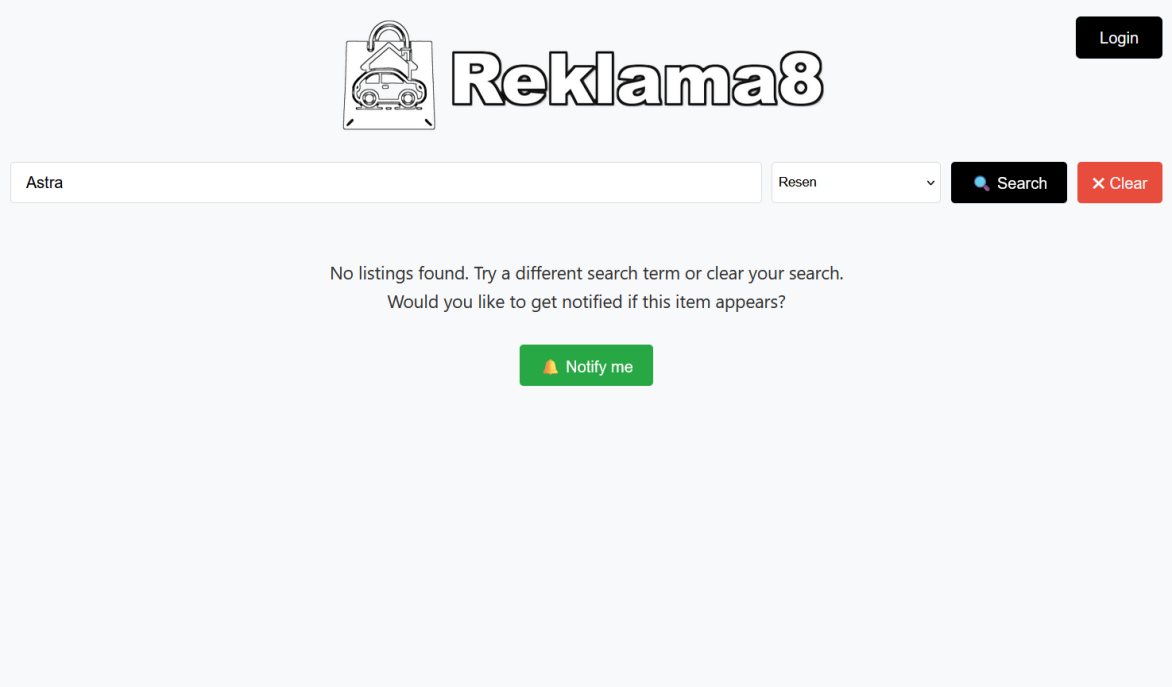
По потврдување на податоците, frontend апликацијата праќа POST барање до серверот на патеката /auth/login.

Spring Boot сервисот ги проверува кредeнцијалите во базата и, ако се валидни, генерира JWT (JSON Web Token). Овој токен потоа се враќа до клиентот, кој го зачувува во локалната меморија (localStorage) и го користи за идни барања.

JWT токенот служи како доказ за идентитет на корисникот, овозможувајќи сигурна и безбедна комуникација помеѓу клиент и сервер. На овој начин се избегнува потребата од класична сесија на серверот, а се зголемува безбедноста и скалабилноста.

5.3 Кога нема достапни огласи

Во одредени случаи, пребарувањето може да не врати резултати - на пример, кога нема огласи за конкретен модел автомобил во одреден град. Во такви ситуации, наместо празен екран, системот прикажува порака за немање резултати и понуда со копче „Извести ме“ (Notify Me).



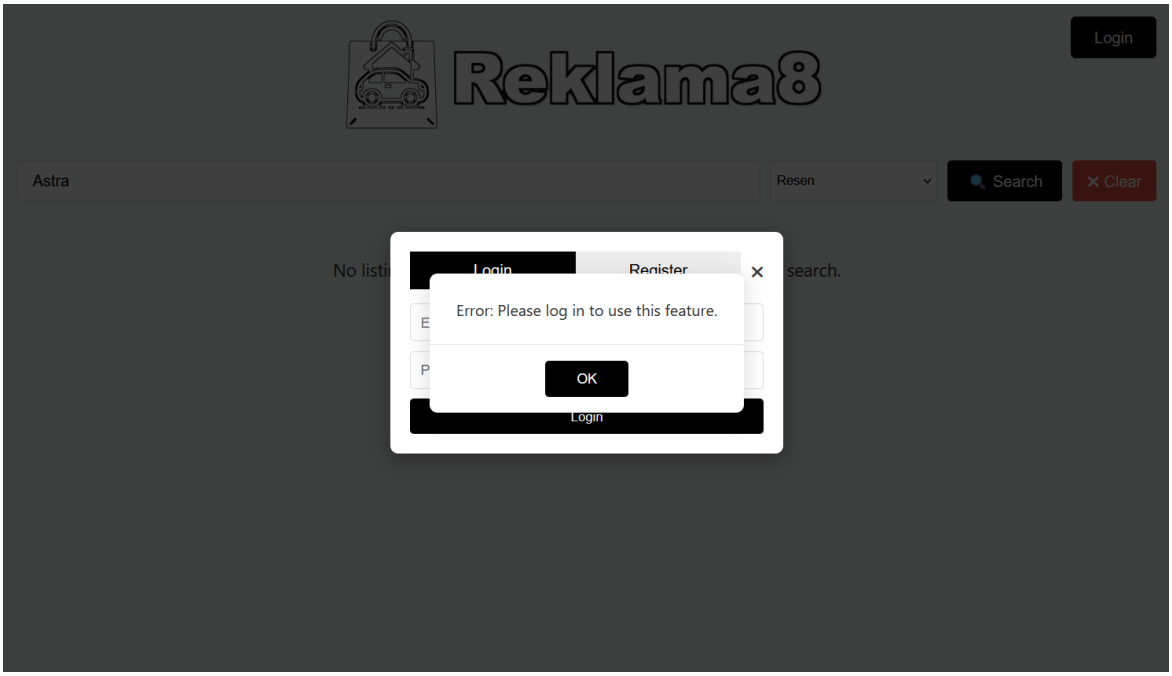
Слика 23: Опција за известување кога нема огласи

Со клик на ова копче, апликацијата му овозможува на корисникот да креира претплата за известување. Backend-от го зачувува барањето во табелата notification_subscriptions, заедно со параметрите на пребарувањето (клучен збор и локација). Подоцна, кога Python скриптата ќе најде нов оглас што одговара на тие критериуми, системот автоматски испраќа email известување. Оваа функционалност го прави Reklama8 интерактивна и корисна платформа што активно ги следи интересите на корисниците.

5.4 Грешка при обид за известување без најава

Ако не најавен корисник притисне „Notify Me“, апликацијата испраќа барање кон API-то /listings/notify. Spring Boot филтерот за безбедност проверува дали во барањето има валиден JWT токен.

Бидејќи токенот недостасува, серверот враќа HTTP статус 401 Unauthorized.



Слика 24: Грешка, ненајавен корисник.

На фронтенд страната, овој одговор се пресретнува од Angular HTTP interceptor, кој го препознава како проблем со автентикација.

Потоа системот прикажува порака:

„Ве молиме најавете се за да ја користите оваа функционалност.“ и автоматски го отвора login/register модалот. Овој механизам спречува неовластен пристап и претставува пример за интегрирана безбедност помеѓу frontend и backend деловите на системот. Сличен механизам има и за регистрирање на корисник кој веќе постои.

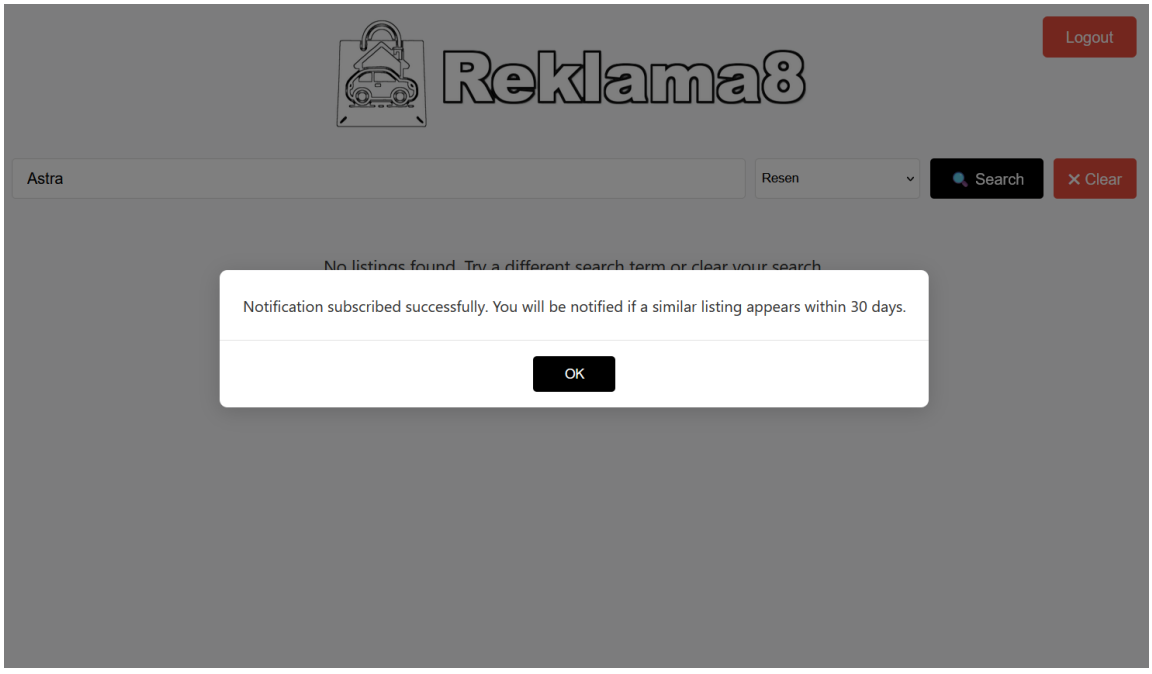
5.5 Грешка при неуспешна најава (Невалидни податоци)

Во случај кога корисникот внесува погрешно корисничко име или лозинка, backend сервисот враќа статус 401 Unauthorized. Angular го прикажува одговорот во форма на визуелна порака во рамките на модалот, без да се затвори формата.

Слика 25: Грешка, невалидни/непостоечки податоци.

Овој пристап им овозможува на корисниците веднаш да го коригираат внесот, без дополнителни чекори или повторно вчитување на страницата. Покрај тоа, ваквиот визуелен feedback ја подобрува употребливоста и го зголемува чувството на интерактивност на системот.

5.6 Успешна претплата



Слика 26: Успешно создадена претплата.

Ова сценарио ја демонстрира комуникацијата помеѓу апликацијата и серверот, барањето се испраќа преку REST API, серверот го обработува, зачувува во базата преку JPA, и враќа потврда.

Благодарение на Spring Boot Starter Mail и JavaMailSender, кога ќе се појави нов оглас кој ги исполнува критериумите на претплатата, системот автоматски испраќа е-пошта до корисникот(слика 9).

5.7 Резиме

Преку овие сценарија може да се забележи целосниот животен циклус на интеракција со системот Reklama8, од пребарување и најавување, до креирање претплати и примање email известувања.

Секој чекор е поддржан со безбедносен механизам, обработка на грешки и визуелна повратна информација кон корисникот. Овој модуларен и современ пристап гарантира стабилност, сигурност и одлично корисничко искуство, што е и главната цел на развојот на овој систем.

ЗАКЛУЧОК

Во рамките на овој труд беше изработен целосен систем за собирање на онлајн огласи, кој комбинира современи веб технологии со автоматизација и интегрирани сервиси за известување. Главната цел беше да се развие веб апликација што овозможува динамично прикажување на огласи од повеќе извори и нивно пребарување, како и автоматско информирање на корисниците кога ќе се појави нов оглас што одговара на нивните претплати.

Во процесот на реализација беа користени неколку современи технологии и библиотеки. На серверската страна, Spring Boot овозможи стабилна и скалабилна архитектура со вградена поддршка за безбедност, обработка на податоци и REST комуникација. Со помош на JPA и Hibernate беше реализирано ефикасно управување со базата на податоци, додека Spring Security и JWT обезбедија сигурна автентикација и авторизација. Spring Boot Starter Mail овозможи автоматско испраќање на email известувања до корисниците, со што системот стекна практична вредност и реална применливост. На апликациската страната, Angular обезбеди брз, интуитивен и реактивен кориснички интерфејс, кој овозможува лесна интеракција со податоците преку REST API.

Дополнително, со помош на Python web crawler скрипта беше овозможено автоматско прибирање на огласи од надворешни извори, нивно обработување и зачувување во JSON формат, што овозможи висока флексибилност и независност од конкретна платформа. Со лесни можности за додавање на повеќе извори само со нивно додавање во crawler-от.

Резултатот е функционален, модуларен и лесно проширлив систем кој може да се користи во различни домени, како барање користени автомобили, недвижнини, услуги, итн.

Како можни насоки за понатамошен развој се издвојуваат: интеграција со уште повеќе извори на податоци, примена на машинско учење за препораки на огласи според интересите на корисникот, како и унапредување на системот за филтрирање и визуелизација на податоците.

Со ова, трудот успешно ја реализира зададената цел — да прикаже современ и практичен пристап за изградба на веб апликација базирана на реални потреби, со употреба на најактуелни технологии од полето на софтверскиот инженеринг.

РЕФЕРЕНЦИ

1. Angular - Wikipedia
2. Angular vs React vs Vue: Core Differences - Browserstack
3. SpringBoot - Wikipedia
4. 10 Reasons Why You Should Use Spring Boot - Sigmasolve
5. Why Is Python a Popular Programming Language for Web Crawling? - Scrapehero
6. How to implement automatic token insertion in requests using HTTP interceptor – Angular.love
7. Sending email through Java with SSL / TLS authentication - Geeks for geeks