

Dnevnik stručne prakse

VEGA IT Sourcing d.o.o.

Novosadkog sajma 2, 21000 Novi Sad, Srbija



Mentor prakse: [REDACTED]

E-mail adresa mentora: [REDACTED]

Opis kompanije

Vega IT je u potpunosti domaća kompanija, koju su osnovali bivši studenti računarskih nauka sa FTN-a. Kompanija već osam godina uspešno posluje na tržištima Velike Britanije, Nemačke, Belgije i Holandije.

Kompanija ima preko 800 uspešno realizovanih projekata širom Evrope, Velike Britanije, SAD-a i UAE. Ključne oblasti ove kompanije su finansiska tehnologija, farmaceutska tehnologija, transport i logistica, prehrambena tehnologija, lanci preduzeća i drugi.

Projekti:

Viantro je platforma za pronalaženje partnera koja pomaže i lekarima da pronađu nove mogućnosti za posao i medicinskim ustanovama da skrate proces zapošljavanja i pristupe pravim medicinskim kandidatima u Nemačkoj. Korištenjem platforme za podudaranje klijenta, medicinske ustanove mogu skratiti postupak zapošljavanja sa 136 dana na samo nekoliko dana. Tehnologije koje su se koristile su: HTML, CSS, Angular, NgRx biblioteka i Angular material za front-end stranu. Za testiranje je korišteno Karma i Jasmin

Paperchase je mobilna aplikacija koja treba da poveća prodaju kancelarijske radnje „Paperchase“. Paperchase je vodeća kancelarijska radnja u Velikoj Britaniji koja ima više od 50 godina u industriji. Ovo preduzeće je pružilo čist dizajn i detaljne specifikacije za mobilnu aplikaciju. Tehnologije koje su se koristile za razvoj ove aplikacije su: React Native, dok je back-end strana razvijena korišćenjem i prilagođavanjem nopCommerce koja se zasniva na jezgri programskog jezika .NET. Za obradu plaćanja se koristila platforma Stripe.

Heartcount je alat za praćenje rezultata i odnosa (lično ispunjenje, napredak, odnosi sa kolegama i menadžerima) koji utiču na sreću na poslu. Alat pruža menadžmentu trenutne povratne informacije i operativne informacije za sve u ekosistemu. Heartcount okuplja čitav ekosistem, od zaposlenih, preko HP službe do menadžera. Namena ovog softvera je da pomogne kompanijama da bolje razumeju svoje zaposlene. Tehnologije koje su se koristile su: JavaScript, TypeScript, Angular 5+, HTML, CSS, na strani servera JavaScript, NodeJS, ExpressJS, Sequelize ORM, MySQL.

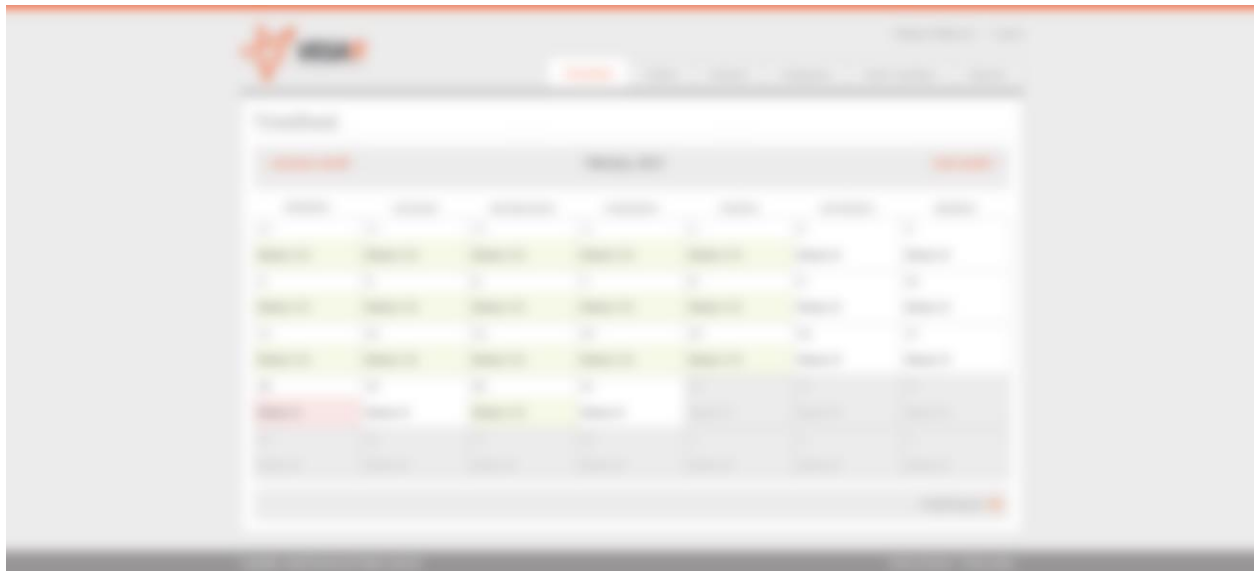
Global retail shop ovo je platforma koja omogućava svojim korisnicima da otvore svoje profile, postavе sliku i opis proizvoda koji žele, a aplikacija će filtrirati najbolje podudaranje proizvoda. Kada korisnik primi ponudu, on ima priliku da izabere najbolju opciju i dovrši kupovinu. Cilj je bio da se poveća broj kupaca pomoću jedinstvene mobilne aplikacije. Tehnologija koja se koristila je React Native.

Dnevnik rada u kompaniji

Moja praksa se malo drugačije sprovodila u odnosu na to kako se ona do sada radila. Ja sam svoju praksu radio od kuće, online. Sa mentorom sam komunicirao preko Skype-a. Sa mentorom sam imao dva sastanka dnevno.

1. dan prakse

Prvog dana prakse se upoznajem sa svojim mentorom. Mentor mi je ispričao da ću tokom prakse razvijati aplikaciju koja se koristi za vođenje evidencije radnih sati za svakog zaposlenog u kompaniji. Zatim mi je mentor poslao prototip aplikacije. Moj prvi zadatak je bio da proučim taj prototip i da na sledećem sastanku sa mentorom, ispričam šta treba da se napravi na osnovu prototipa i da pitam ako mi je potrebno neko objašnjenje. Izgled prototipa možete pogledati na slici 1.



Slika 1

Na početnoj strani je prikazan tab *TimeSheet* sa danima i ukupnim radnim satima po danu. Pored ovog tab-a, nalaze se još sledeći tab-ovi: *Clients* sa listom klijanata, *Projects* sa listom projekata, *Categories* sa listom kategorija, *Team members* lista sa timovima i *Reports* lista izveštaja za određenog korisnika. Na sledećem sastanku popričao sam sa mentorom o prototipu i mom sledećem zadatku. Moj sledeći zadatak je bio da napravim React aplikaciju i da HTML strane iz prototipa prebacim u React aplikaciju. Ovaj zadatak sam uspešno rešio bez većih problema.

2. dan prakse

Na sastanku sa mentorom prezentovao sam to što sam uradio. Na prototipu se menjanjem taba-a, menjao i prikaz sadržaja stranice, kao i sam izgled tog tab-a. Trenutno aktivni tab je izgledao drugačije od drugih tab-ova. Moj sledeći zadatak je bio da obezbedim da se klikom na određeni tab promeni izgleda tab-a i sam sadržaj stranice. Prikaz sadržaja određenog tab-a sam napravio pomoću React-ove komponente *Router*. Korišćenje ove komponente možete pogledati na slici 2.



slika 2

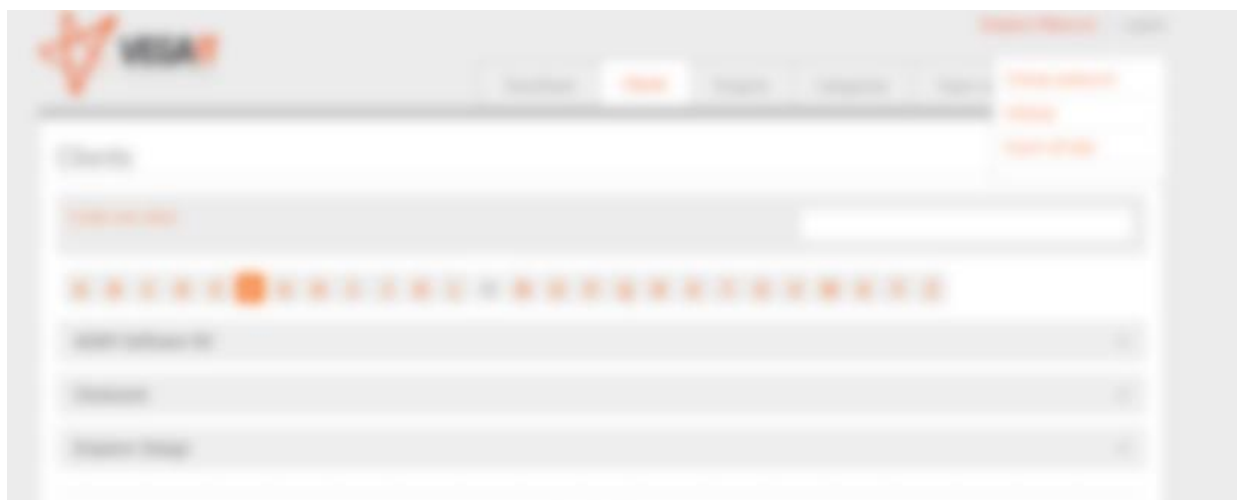
Unutar *Router* komponente se nalaze moje komponente i *Switch* komponenta koja sadrži *Route* komponentu u kojoj sa *path* atributom definišemo url do stranice sa *exact* atributom definišemo koja stranica je početna i sa atributom *component* definišemo koja će se komponenta pozvati da bi prikazala sadržaj stranice. U svaku od komponenti trebalo je dodati sledeći kod koji možete pogledati na slici 3.



slika 3

Koristio sam *NavLink* komponentu, u njoj sam takođe pomoću *exact* atributa naveo koji od tab-ova je na početku aktivan, pomoću atributa *to* definišemo url stranice koju želimo da prikažemo, *className* atribut definiše css klasu koja se primenjuje na tab, a *activeClassName* definiše css klasu koja će biti primenjena kada se klikne na dati tab. Moj sledeći zadatak je bio da pomoću css napravim da se postavljanjem

kursora na ime ulogovanog korisnika prikazuju opcije (Slika 4), trebao sam napraviti da se kao i u prototipu klikom na određenog korisnika prikazuju njegove informacije (Slika 5) i trebao sam još napraviti da se klikom na Create new client prikaže dijalog za unos novog korisnika (Slika 6).



slika 4



slika 5



slika 6

3. dan prakse

Na *li* tag sam dodao *id* sa nazivom *loggedIn* (Slika 7), prilikom prelaska preko tog tag-a, aktivira se css stil i za *display* postavlja vrednost *block* (Slika 8). Podrazumevana vrednost za *display* za klase *user-menu* i *invisible* je *none*.



Slika 7



Slika 8

Prikaz dodatnih informacija za korisnika sam napravio tako što sam dodao sledeći kod (Slika 9).

Kada se klikne na određeni element lokalna promenljiva *isOpen* za tu komponentu dobija vrednost *true*, zatim funkcija *componentDidUpdate* automatski reaguje na menjanje stanja promenljive *isOpen*,

```
componentDidUpdate(){  
  this.$el = $(this.state.element)  
  if (this.state.isOpen) {  
    this.$el.slideDown('normal');  
  } else {  
    this.$el.slideUp('normal');  
  }  
}
```

provreva da li je vrednost promenljive *true*, ako jeste, prikazuju se dodatne informacije za korisnika. Kada se klikne na korisnika, čije su dodatne informacije prikazane, *isOpen* promenljiva dobija vrednost *false*, poziva se *componentDidUpdate*, koja proverava vrednost *isOpen* promenljive i ako je ona *false*, dodatne informacije za korisnika se zatvaraju.

Slika 9

4. i 5. dan prakse

Za kreiranje dijaloga za novog korisnika sam koristio komponentu *Modal* (Slika 10). Unutar *Modal* komponente, pomoću *visible* atributa sam definisao da prikaz dijaloga zavisi od lokalne promenljive *newClient*, pomoću *onClickAway* atributa sam definisao koja će se funkcija pozvati ako se klikne van dijaloga, pored ovih atributa postoje još *width* ili širina dijaloga, *height* ili visina dijaloga i atribut *effect* atribut kojim možemo definisati na koji način će se prikazati dijalog. Unutar tag-ova se navodi komponenta koja se poziva, to jest koja će nam prikazati formu za unos novog korisnika.

```
<Modal  
  visible={this.state.newClient}  
  width="470px"  
  height="150px"  
  effect="fadeinup"  
  onClickAway={this.handleClick}  
>  
  {newClient}  
</Modal>
```

Slika 10

Sledeći zadatak je bio da za *TimeSheet* tab omogućim promenu meseca i da se za dati mesec prikazuju njegovi dani. Prilikom prvog prikaza ove stranice prikazuje se mesec sa današnjim datum, automatski se izračunava i prethodni kao i naredni mesec. Klikom na *previous month* ili *next month* (Slika 1) možemo promeniti mesec koji se prikazuje. Promenom meseca, ponovo se izračunava prethodni i naredni mesec. Funkcije koje nam ovo omogućavaju su: *calcNumDays*, računa koliko će se dana prikazati iz prethodnog meseca (Slika 11), izračunava koji je prethodni i naredni mesec u odnosu na trenutni mesec, *showCels* koja prikazuje polja, to jest dane meseca (Slika 12).


```

void printArray(int arr[]) {
    int n = sizeof(arr)/sizeof(arr[0]);
    for(int i=0; i<n; i++)
        cout<< arr[i]<<" ";
    cout<<endl;
}

// Driver code
int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printArray(arr);
    return 0;
}

```

Slika 11

Sledeći zadatak je bio sličan prethodnom, klikom na neki dan u mesecu potrebno je bilo da se prikažu detalji za taj dan.

```

def solve(n, dp, product, p, prime, visited):
    if n == 1:
        return 0
    if dp[n] != -1:
        return dp[n]
    if n == 2:
        return 1
    if n == 3:
        return 2
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            dp[n] = max(dp[n], 1 + solve(n // i, dp, product, p, prime, visited))
    for i in range(2, n):
        if n % i == 0:
            dp[n] = max(dp[n], 1 + solve(i, dp, product, p, prime, visited))
    if n == 4:
        dp[n] = dp[n] + 1
    if n == 6:
        dp[n] = dp[n] + 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            if i not in prime:
                continue
            if n // i not in prime:
                continue
            dp[n] = max(dp[n], 1 + solve(n // i, dp, product, p, prime, visited))
            dp[n] = max(dp[n], 1 + solve(i, dp, product, p, prime, visited))
    return dp[n]

n = int(input())
dp = [-1] * (n + 1)
product = 1
prime = []
visited = []
solve(n, dp, product, 1, prime, visited)
print(dp[n])

```

Slika 12

6. i 7. dan prakse

Napravio sam da se klikom na određeni dan meseca prelazi na detaljniji prikaz za taj dan. Klikom se šalje datum na koji se kliknulo i na osnovu datuma se izračunava trenutna nedelja u godini i ona se zatim prikazuje (Slika 13).

Slika 13

Dani za datu nedelju se prikazuju pomoću funkcije *calcDate* (Slika 14).

[illegible]

Slika 14

Klikom na *previous week* ili *next week* (Slika 15) možemo da promenimo nedelju koja pregledamo kao i klikom na neki od dana možemo da promenimo dan koji se prikazuje.



Slika 15

Promenom nedelje ili dana u nedelji pozivaju se ponovo funkcije *getWeekNumber* i *calcDate*.

8. i 9. dan prakse

Podatke koje prikazuju komponente trebalo je premestiti u drugu komponentu. U toj komponenti su se nalazili svi podaci koji su se prikazivali. Ovu komponentu smo nazvali kontejner i ona je prosleđivala podkomponentama podatke. Podkomponente nisu bile svesne porekla podataka, već je njihov zadatak bio samo da prikažu podatke, dok se kontejner komponeta brinula kako da dođe do podataka. Na početku u kontejner komponenti sam imao listu sa podacima. Sledeći zadatak je bio da kreiram backend. Potrebno je bilo da kreiram kontroler (slika 16) koji će mi poslati podatke i potrebno je bilo da povežem frontend i backend. Podatke koje dobija kontejner na frontend-u dobavlja servis (Slika 17) koji sam izdvojio kao posebnu klasu u slučaju da bude bilo potrebe da se kasnije menja izvor podataka. U servisnoj klasi postoji funkcija za svaku od REST metoda, to jest svaka funkcija gađa određeni URL kontrolera i tako obezbeđuje podatke za prikaz ili obezbeđuje da se dodaju, ažuriraju ili brišu podaci. Funkcije servisne klase se pozivaju u kontejneru (Slika 30).

Slika 30

Slika 16

```

@GetMapping("/{id}")
public ResponseEntity<Response> findById(@PathVariable Long id) {
    Response response = responseRepository.findById(id)
        .orElseThrow(() -> new ResponseNotFoundException(id));
    return ResponseEntity.ok(response);
}

@PostMapping("/{id}")
public ResponseEntity<Response> update(@PathVariable Long id, @RequestBody Response response) {
    Response existingResponse = responseRepository.findById(id)
        .orElseThrow(() -> new ResponseNotFoundException(id));
    existingResponse.setName(response.getName());
    existingResponse.setDescription(response.getDescription());
    existingResponse.setPrice(response.getPrice());
    existingResponse.setCategory(response.getCategory());
    existingResponse.setCreated(response.getCreated());
    existingResponse.setUpdated(response.getUpdated());
    Response updatedResponse = responseRepository.save(existingResponse);
    return ResponseEntity.ok(updatedResponse);
}

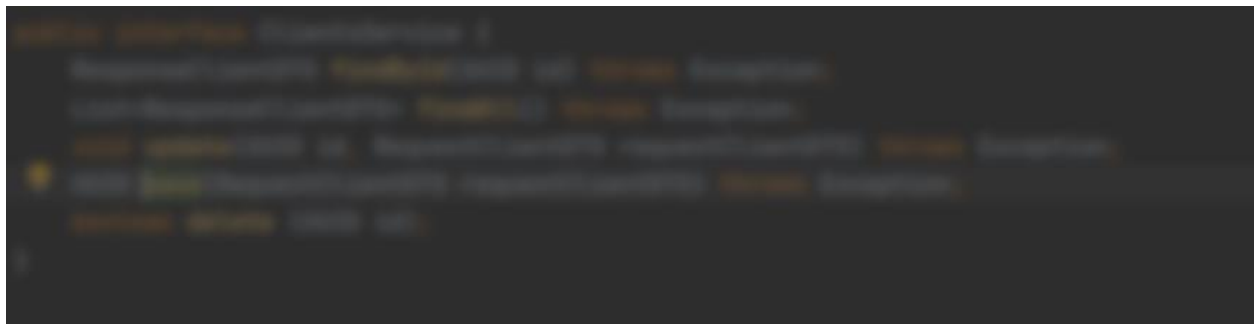
@DeleteMapping("/{id}")
public ResponseEntity<Response> delete(@PathVariable Long id) {
    Response response = responseRepository.findById(id)
        .orElseThrow(() -> new ResponseNotFoundException(id));
    responseRepository.delete(response);
    return ResponseEntity.ok(response);
}

```

Slika 17

10. i 11. dan prakse

Sledeći korak je bio da napišem kod za servis. Kontroler je bio zadužen da pošalje podatke frontend-u, ali logiku backend-a je radio servis. Za svaku metodu iz kontrolera, napisao sam funkciju u servisu koja je dobavljala podatke, kao i skladištila, ažurirala i brisala. Za servis sam prvo kreirao interfejs (Slika 18), a zatim sam kreirao sam servis koji je implementirao moj interfejs (Slika 19). U kontroleru sam autowired-ovo interfejs, a ne implementaciju servisa.



Slika 18



Slika 19

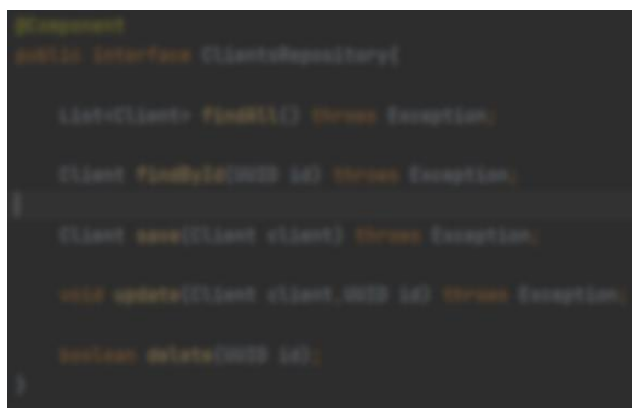
U servisu koristim *MapService* (Slika 20) koji mapira java objekte u java objekteDTO, ovo radim radi serijalizacije podataka ka frontend-u. Za *mapService* takođe postoji interfejs.



Slika 20

12. dan prakse

Sledeći zadatak je bio pisanje repozitorijuma. Kao i za servise i za njega sam morao prvo napisati interfejs (Slika 21), a zatim sam pisao implementaciju repozitorijuma (Slika 22) koji implementira prethodno napisani interfejs. U repozitorijumu se nalaze funkcije za dobavljanje, kreiranje, ažuriranje i brisanje podataka. Repozitorijum komunicira sa bazom, a zatim podatke šalje servisu koji podatke pomoću *mapService*-a prepakuje u DTO-ove i šalje ih kontroleru, a kontroler šalje frontend-u.



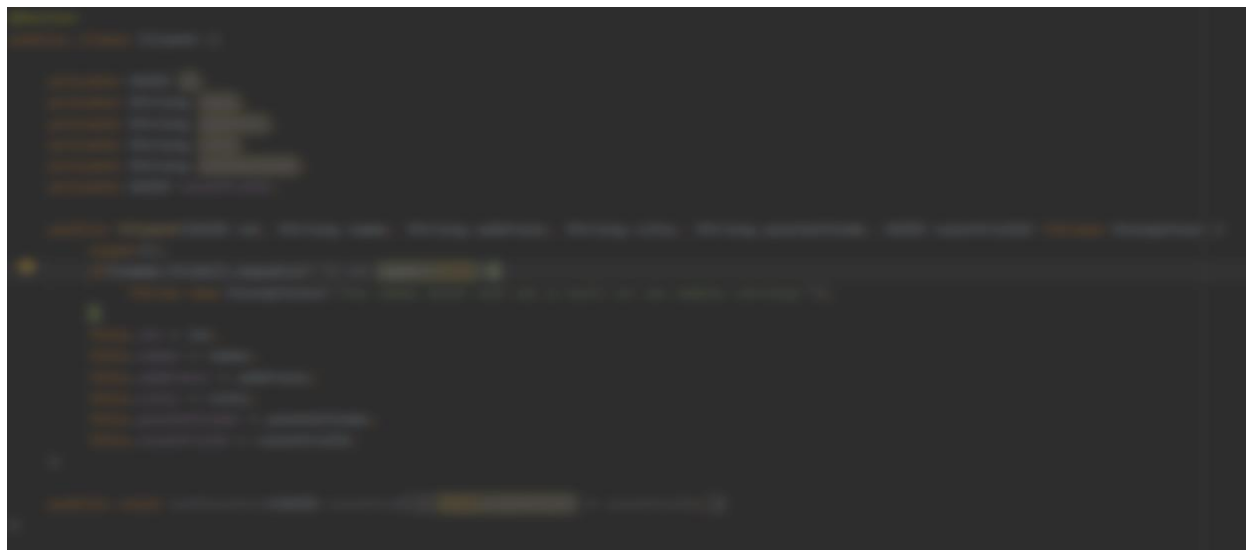
Slika 21



Slika 22

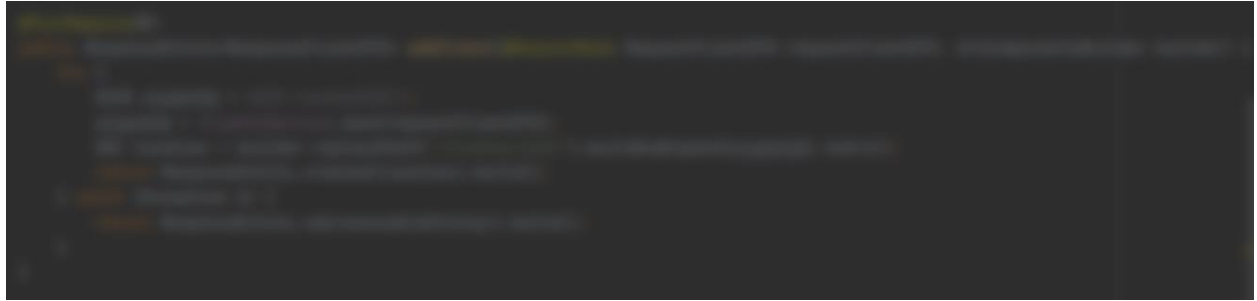
13. dan prakse

Potrebno je postaviti zaštitu u klasu *Client* kako se ne bi mogao napraviti klijent bez imena i brišu se seteri kako se ime kasnije ne bi moglo menjati (Slika 23).



Slika 23

U slučaju da se pokuša napraviti objekat tipa *Client* bez imena tada će se baciti izuzetak koji će se propagirati do samog kontrolera koji će frontend-u kao odgovor vratiti *unprocessableEntity* (Slika 24).



Slika 24

Pored validacije za ime klijenta bilo je potrebno odraditi i validaciju prilikom ažuriranja i brisanja. U slučaju da se želi ažurirati klijent koji ne postoji tada se baca izuzetak koji se propagira do kontrolera koji kao odgovor prosleđuje *notFound* (slika 25). Kod brisanja ako ne postoji određeni klijent ne izaziva se izuzetak, već samo vraća boolean vrednost na osnovu koje kontroler kao odgovor vraća *noContent* ili *notFound* u zavisnosti da li je klijent postojao ili ne.

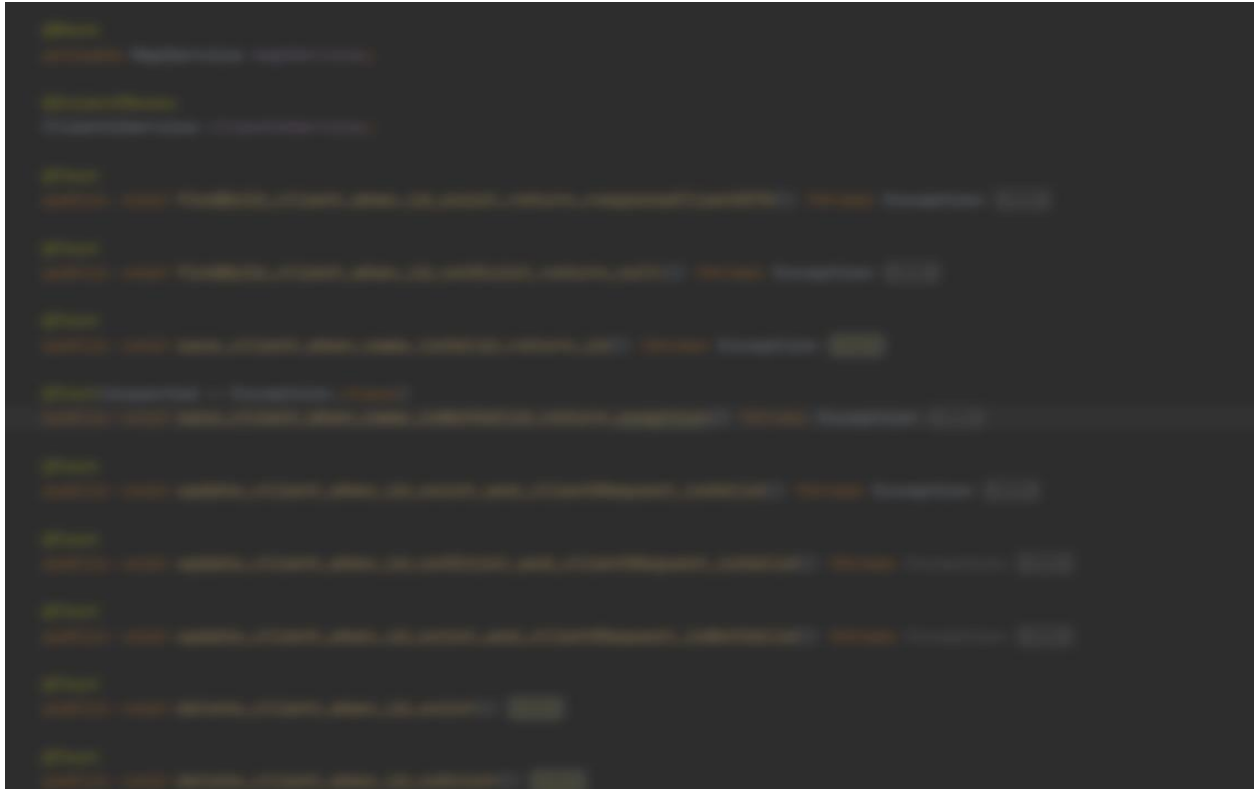
14. dan prakse

Potrebno je bilo kreirati testove za *Client* klasu (Slika 25) i za servis (Slika 26).



Slika 25

Za testiranje *Client* klase naveo sam parametre koje kasnije koristim. Pomoću *dataWithName* funkcije kreiram listu objekata tipa *Object* od kojih se kasnije kreiraju objekti tipa *Client*. Svaki *Client* je drugačiji od prethodnog i tako iteriranjem kroz listu kreiramo nove objekte tipa *Client* koje odmah i testiramo. Prvi test testira kada je ime klijenta postoji, a drugi test testira kada ime klijenta ne postoji.

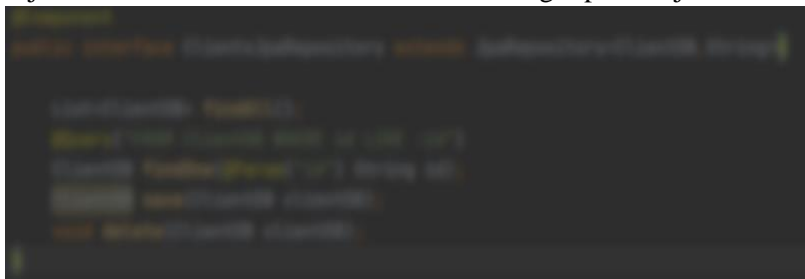
A screenshot of a code editor showing Java code for testing the Client class. The code includes imports for Mockito and JUnit, and defines a test class with two test methods. The first method, testClientExists, uses dataWithName to create a list of Client objects and then iterates through them to verify that the client exists. The second method, testClientDoesNotExist, uses dataWithName to create a list of Client objects and then iterates through them to verify that the client does not exist.

Slika 26

Za testiranje servisa sam koristio *mockito* biblioteku. Sa *@Mock* anotacijom kažemo koje objekte želimo da nam *mockito* obezbedi za testiranje, a sa *@InjectMocks* mockito-u kažemo šta testiramo i da želimo da nam napravi instancu tog objekta. U ovom testu sam testirao sve metode *clientsService*-a. Imamo slučaj kada su podaci validni i kada podaci nisu validni.

15. dan prakse

Za komunikaciju sa bazom podataka koristim *JpaRepository*. Napravio sam repozitorijum koji nasleđuje *JpaRepository* (Slika 27), a zatim sam taj repozitorijum autowired-ovo u moj repozitorijum (slika 22) i u njemu sam koristio metode autowired-ovanog repozitorijuma.

A screenshot of a code editor showing Java code for a custom repository. The code defines a class that inherits from JpaRepository and implements its methods. The code is written in a dark-themed editor with syntax highlighting.

Slika 27

Dodao sam još da funkcije u *mapService* (Slika 28) koje mapiraju klasu *Client* na klasu *ClientDB* (Slika 29).

```
@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}
```

Slika 28

```
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}

@Override
public void setUp() throws Exception {
    // ...
}
```

Slika 29