UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Danilo Novaković, PR136-2016

Razvoj veb aplikacije za rezervaciju apartmana primenom CQRS šablona

Projekat

- Primenjeno softversko inženjerstvo (OAS) -

Novi Sad, 2020.

Sadržaj

1 UVOD

Istraživanja kažu da čovek u proseku 10 do 100 puta više čita sadržaj web stranica naspram njihovog modifikovanja (u slučaju Instragram-a za jednu dodatu sliku poznate ličnosti servis može imati preko milion čitanja). Iz ovog razloga, kao i zbog velikog broja potencijalnih korisnika (kao uzrok popularnosti web-a), javlja se potreba optimizacije procesa čitanja.

U radu će biti predstavljen CQRS šablon koji omogućava rešavanje ovog problema, primenjen na Web aplikaciji za rezervaciju apartmana, sa smernicama ka daljem razvoju i mogućim alternativama.

2 KORIŠĆENE TEHNIKE I TEHNOLOGIJE

REST - *Representational State Transfer*, je definisan 2000. godine u doktorskoj disertaciji koju je napisao *Roy Fielding*. On definiše principe softverske arhitekture, ignorišući detalje implementacije i sintakse i daje fokus ulogama komponenti. U vebu se *REST*, odnosno *RESTful API* oslanja na *HTTP* protokol, pa se samim tim operacije nad resursima svode na *HTTP* metode (*GET*, *POST*, *PUT*, *DELETE* su samo neke od njih). [1]

Entity Framework (EF) Core - lightweight, proširiva, open-source i cross-platform verzija popularne Entity Framework tehnologije. EF Core služi kao object-relational mapper (O/RM), omogućavajući .NET programerima da rade entitetima iz baza podataka kao sa .NET objektima, time uklanjajući potrebu za većinom koda za pristup podacima koji obično trebaju pisati. [2]

ASP.NET Core – cross-platform, visoko performansni, open-source framework za kreiranje modernih web aplikacija (servisa) sa cloud podrškom. [3]

React - (poznat i kao **React.js** ili **ReactJS**) je Javaskrpit okrućenje otvorenog koda koja obezbeđuje pregled podataka zapisanih preko HTMLa. React pregledi su obično obezbeđeni korišćenjem komponenti koje sadrže dodatne komponente definisane kao prilagođene HTML oznake. React obezbeđuje programeru model u kojem podkomponente ne mogu direktno da utiču na spoljašnje komponente, efikasno ažuriranje HTML dokumenta pri promeni podataka i jasno razdvajanje komponenti na današnjim jednostraničnim aplikacijama [4]

Redux – predvidiv state container za JavaScript baziranje aplikacije. Uvodi par termina kao što su **store** (model), **actions** (zahtev za promenu modela) i **reducers** (komponente koje menjaju model na osnovu zahteva – *action*-a)**.** Najpopularnije rešenje za state-management u React aplikacijama [5]

MediatR biblioteka predstavlja prostu implementaciju mediator šablona u kojem objekti ne komuniciraju direktno već preko *mediator* objekta. Na ovaj način se zavisnost između objekata smanjuje, samim tim povećava skalablinost [6]

MongoDB.Driver – biblioteka koja omogućava komunikaciju između MongoDB baze i .NET aplikacija [7]

3 SPECIFIKACIJA

Potrebno je realizovati veb aplikaciju za sistem koji podržava rezervacije apartmana (kao Airbnb aplikacija). Aplikaciju koriste tri grupe (uloge) **korisnika**: Gost, Domaćin i Administrator.

Aplikacija je osmišljena tako da podržava tri tipa korisnika (isključujući slučaj kada korisnik nije registrovan):

- Korisnik administrator koji upravlja podacima
- Korisnik domaćin koji izdaje apartmane
- Korisnik gost koji rezerviše apartmane

Uloga **administratora** se svodi na to da može da menja postojeće, briše ili dodaje nove entitete (apartmane, sadržaje apartmana, domaćine,…). On je zadužen za to da svi podaci u sistemu budu ispravni i tačni u svakom momentu. Administratori se učitavaju na početku rada programa i ne mogu se naknadno dodavati.

Dostupne aktivnosti za **domaćina** (*Host*) su sledeće:

- Dodavanje, brisanje i modifikacija apartmana
- Mogućnost odbijanja, prihvaćenja i markriranje rezervacija kao “završene“ nakon završnog noćenja gosta
- Prikaz svih rezervacija nad svojim apartmanima
- Prikaz svojih apartmana
- Odluku koji komentari da se prikazuju na njegovom apartmanu

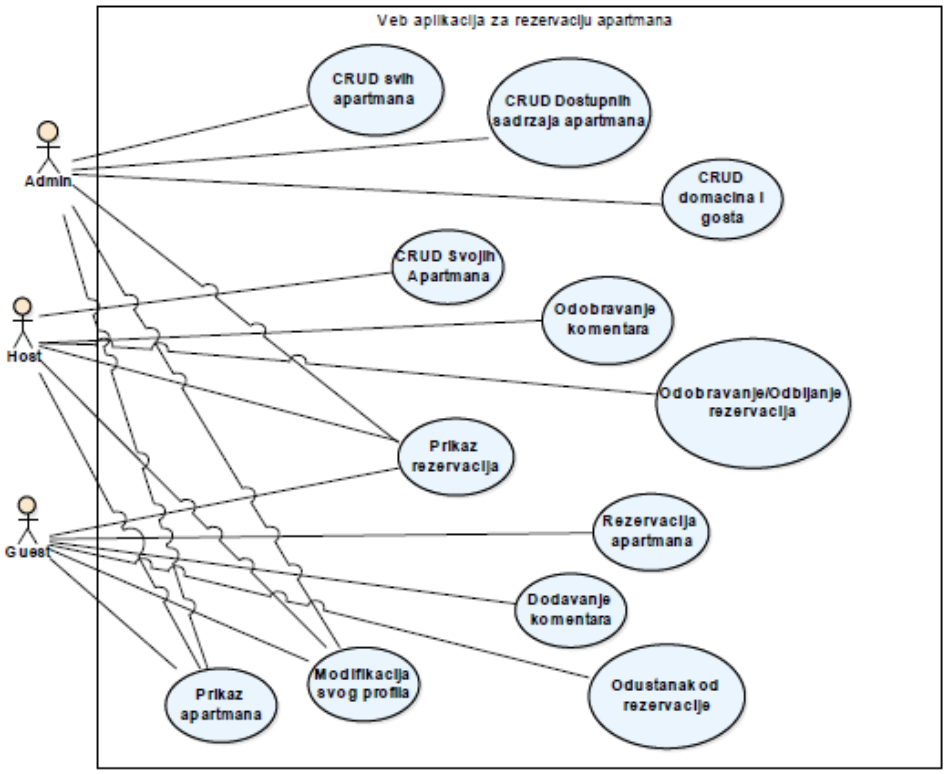
Gostima (*Guest*) su dostupne sledeće mogućnosti:

- Pregled svih apartmana (pretraga, filtriranje, sortiranje)
- Pregled svojih rezervacija i odustanak od istih (ukoliko je to moguće – više o ovome u nastavku teksta)
- Rezervacija apartmana
- Mogućnost ostavljanja komentara za apartman nakon završnog noćenja

Ne registrovani korisnici imaju mogućnost prikaza svih apartmana.

Svi korisnici mogu modifikovati-brisati svoje profile.

Dijagrami slučajeva korišćenja (*use-case* dijagrami) se koriste da predstave niz akcija koje sistem može ili treba da izvrši u toku komunikacije sa korisnikom tog sistema. Oni u sebi sadrže učesnika (aktera) i funkcije koje sistem obezbeđuje učesniku.



SLIKA 3.1 - DIJAGRAM SLUČAJA KORIŠĆENJA

4 Serverska strana rešenja

Serverska strana je implementirana putem ASP.NET Core framework-a, sa MongoDB.Driverom za NoSQL bazu i EF Core ORM tehnologijom za SQL bazu. U ovoj sekciji će biti objašnjen CQRS šablon, prikazani modeli podataka, načini sinhronizacije (tj. replikacije ili projekcije) podataka sa strane za pisanje u stranu za čitanje kao i prikaz strukture foldera (slojevite arhitekture).

4.1 CQRS

Osnovna ideja **CQRS** (Command Query Responsibility Segregation) jeste razbijanje zajedničkog modela na dva – jedan za pisanje i jedan za čitanje.

U svojoj srži proširuje ideju **CQS** (svaka metoda treba biti komanda (proizvodi bočne efekte) ili query (bez bočnih efekata), ali ne oba, drugim rečima "*postavljanje pitanja nebi trebalo da menja odgovor*") na arhitekturalni nivo (ako je CQS na nivou metoda, CQRS bi bio na nivou klasa).

4.1.1 Prednosti

Skalabilnost – u prosečnoj enterprise aplikaciji su najviše korišćene operacije čitanja, CQRS nam omogućava da nezavisno skaliramo ova dva modela (npr. Cluster od 10 servera za query stranu). Moguće je takođe implementirati više od jednog read modela – npr. jedan optimizovan za potrebe desktop korisnika, drugi za potrebe mobilnih aplikacija, dok u domenu mikroservisa read baza može služiti kao "cache" pomoću koje bi se izbegli interni pozivi ka drugim servisima.

Performanse – čak i ako se odlučimo da nam read i write strane budu na jednom serveru (bazi) opet možemo primeniti tehnike optimizacije koje nebi bile moguće u slučaju jednog zajedničkog modela – npr. samo postojanje odvojenog API-a za Queries (upite) nam dozvoljava da podesimo Cache za taj deo aplikacije, kao i pisanje visoko optimizovanih SQL upita dok bi na Command strani koristili neki ORM.

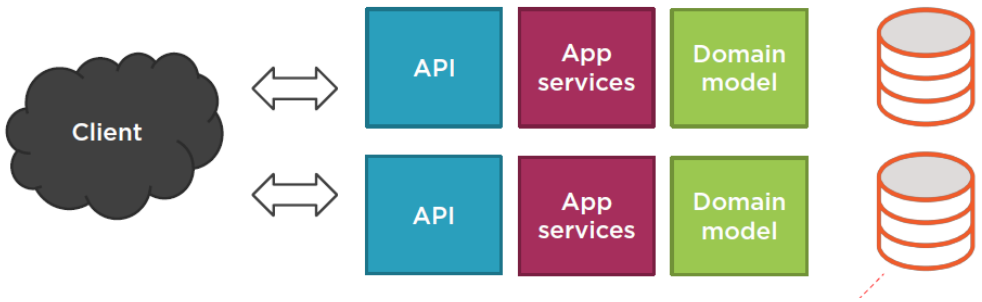
Jednostavnost – Query i Command strane uglavnom imaju drugačije zahteve što čini da zajedniči model često završi kao kompleksno rešenje koje nijedan zahtev ne ispunjava optimalno. Samo razbijanje modela na dva dela nam nekad čak može dati novi pogled na aplikaciju i time probuditi neke nove ideje kao i jednostavnija (razumljivija - "čistija") rešenja.

Praćenjem CQRS-a mi tehnički poštujemo Single Responsibility princip (S iz SOLID-a) na arhitekturalnom nivou

4.1.2 Nivoi CQRS-a

Kao što je već navedeno, CQRS ne zahteva obavezno separaciju na nivou baza podataka (npr. korišćenjem Entity Framework-a za pisanje i ADO.NET (ili dapper-a) za čitanje vi tehnički poštujete CQRS princip) te se u literaturi izdvajaju 3 tipa (nivoa) CQRS-a:

1. **Separacija na nivou softvera** (zajednička baza za pisanje i čitanje) – ovo je najjednostavniji oblik CQRS-a koji ima najmanje troškove. Separacija čitanja i pisanja na nivou API-a, metoda i klasa više služi kao dobra praksa (tj. "clean design")
2. **Separacija na nivou baza podataka** (odvojene baze za pisanje i čitanje) - ova verzija se smatra kao "potpuna" iz razloga što su Read i Write strane potpuno odvojene (nezavisne) jedne od druge, u toj meri da ih možete pokrenuti na odvojenim instancama / mikroservisima. Međutim, nasprem 1. verzije ovde počinje da raste kompleksnost naše aplikacije jer:
    a. Moramo razmišljati o konzistentnosti podataka – podaci postaju eventualno konzistentni
    b. Javlja se potreba za replikacijom (sinhronizacijom/projekcijom) podataka iz strane sa čitanje u stranu za pisanje (*o ovome više u nastavku teksta)*
3. **Kombinacija sa Event Sourcing-om** – ova verzija, kao i 2. podrazumeva odvojene baze za pisanje i čitanje, sa tim da se u bazi za pisanje, umesto relacionog modela podataka, zapisuju događaji (events). Ova verzija CQRS-a nosi sa sobom najveću kompleksnost i zapravo ima najviše smisla u sistemima u kojima nam je bitan pregled istorije izmena, kao i u domenu mikroservisa gde bi pisanje replikatora u vidu State-driven projekcije bila previše skupa/kompleksna (ili u nekim slučajevima čak i nemoguća) aktivnost



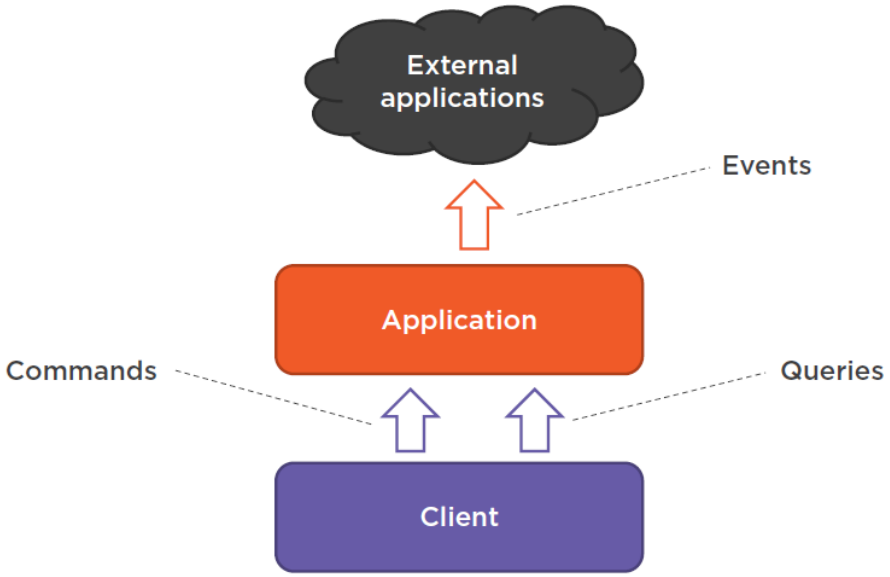
SLIKA 4.1 – PRIMER NEKIH OD MESTA GDE SE MOŽE PRIMENITI SEPARACIJA

4.1.3 Razlike između komandi, upita i događaja

Radi izbegavanja potencijalne zabune potrebno je napraviti jasnu razliku između komandi, upita i događaja u kontekstu CQRS-a.

Komande (commands), kao i upiti (queries) mogu biti viđeni kao “push” model u kome klijent zapravo inicira akciju ka našem servisu. One su nešto što treba da se izvrši i kao takve uglavnom imaju naziv u sadašnjem vremenu (npr. “RegisterUser“, „GetAvailableApartments“).

Događaji (events) bi zapravo predstavljali “pull” model u kome bi aplikacije bile obaveštene da se nešto desilo, zbog ovoga one uglavnom imaju naziv u prošlom vremenu (npr. “UserRegistered”). Događaji u kontekstu CQRS-a se uglavnom koriste za Event-Driven sinhronizaciju baza podataka u kombinaciji sa Event-Sourcing-om.



SLIKA 4.2 – ILUSTRACIJA ULOGA UPITA, KOMANDI I DOGAĐAJA U SISTEMU

4.1.4 MediatR

MediatR je .NET biblioteka koja se bavi rešavanjem jednog prostog problema – odvajanje “in-process“ slanja poruka od obrađivanja istih.

Ona pruža razne interfejse, međutim one od interesa koje su korišćene u primeru rezervacije apartmana su Request/Response

Request objekat predstavlja ulazni parametar od metode RequestHandler-a. Način na koji MediatR zna da poveže Request sa RequestHandlerom je putem refleksije.

CQRS i MediatR možemo da kombinujemo tako što ćemo commands & queries implementirati kao Request objekte (sa njihovim RequestHandler-ima). Upravo je to i urađeno.

```
5 references | 0 changes | 0 authors, 0 changes
public class DeleteAmenityCommand : IRequest
{
    4 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public long Id { get; set; }
}
```

SLIKA 4.3 – PRIMER REQUEST OBJEKTA

```
3 references | 0 changes | 0 authors, 0 changes
public class DeleteAmenityCommandHandler : IRequestHandler<DeleteAmenityCommand>
{
    private readonly IApartmentReservationDbContext context;

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public DeleteAmenityCommandHandler(IApartmentReservationDbContext context)
    {
        this.context = context;
    }

    99 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public async Task<Unit> Handle(DeleteAmenityCommand request, CancellationToken cancellationToken)
    {
        var dbAmenity = await this.context.Amenities
            .SingleOrDefaultAsync(a => a.Id == request.Id && !a.IsDeleted, cancellationToken)
            .ConfigureAwait(false);

        if (dbAmenity is null)
        {
            throw new NotFoundException();
        }

        dbAmenity.IsDeleted = true;
        await this.context.SaveChangesAsync(cancellationToken).ConfigureAwait(false);

        return Unit.Value;
    }
}
```

SLIKA 4.4 – PRIMER REQUESTHANDLER OBJEKTA

Ono što nam ovo omogućuje jeste da skoro svu logiku izbacimo iz kontrolera na prezentacijonom sloju. Ono što on postaje jeste samo jako tanka komponenta koja šalje Request objekte mediatoru i vraća rezultat nazad (uz eventualne minimalne modifikacije).

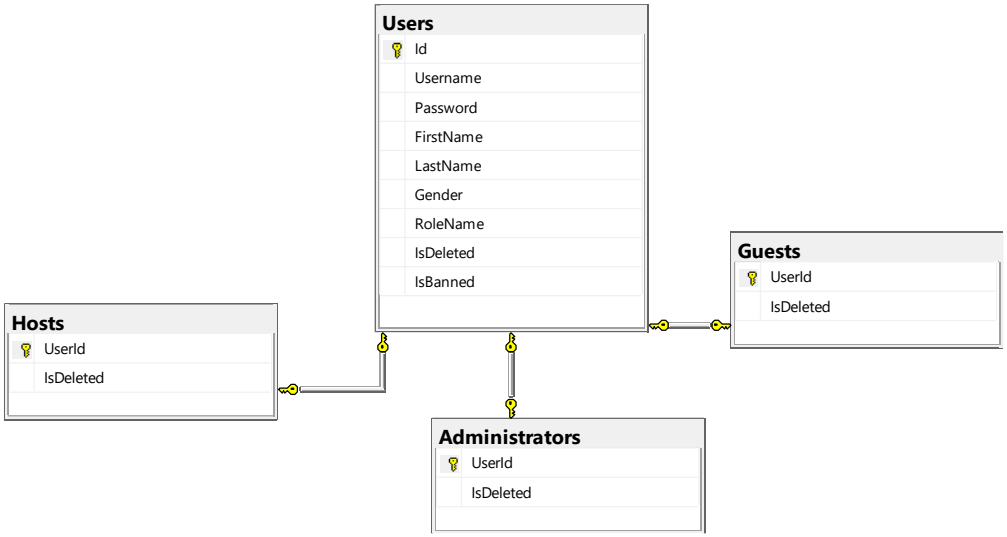
```
// DELETE: api/Amenities/5
[HttpDelete("{id}")]
[Authorize(Policy = Policies.AdministratorOnly)]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public async Task<IActionResult> Delete(long id)
{
    await this.mediator.Send(new DeleteAmenityCommand() { Id = id }).ConfigureAwait(false);

    return this.NoContent();
}
```

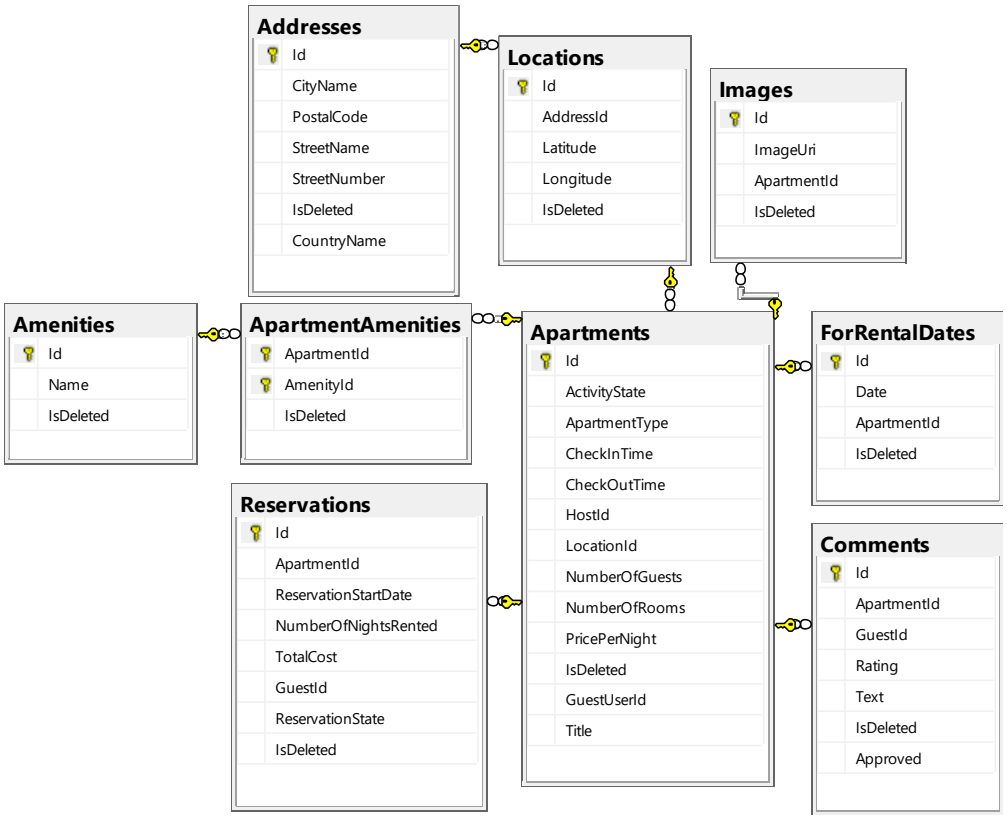
SLIKA 4.5 – PRIMER METODE API KONTROLERA KOJA KORISTI MEDIATOR

4.2 Command Model

Za implementaciju baze podataka za komande (pisanje) je korišćen relacioni SQL model podataka. U nastavku će biti prikazan diagram modela podataka generisan od strane Microsoft SQL Studio alata.



SLIKA 4.6 – MODEL KORISNIČKIH ULOGA U SISTEMU



SLIKA 4.7 – MODEL ENTITETA VEZANIH ZA APARTMANE

4.3 Query Model

Za implementaciju baze podataka za čitanje (upite) je korišćen nerelacioni MongoDB model podataka. Sem toga što nam MongoDB-a omogućava izbegavanje spajanja tabela prilikom upita, sam model podataka za čitanje je normalizovan i optimizovan tako da u potpunosti odgovara potrebama klijentske strane – u toj meri da se ovaj model skoro u potpunosti podudara sa DTO objektima.

Primer optimizacije modela za čitanje možemo primetiti već kod entiteta za Rezervaciju. Ukoliko bi hteli iz prethodno navedenog SQL modela (*Slike 4.6 i 4.*7) da pošaljemo klijentu nazad objekat u formatu prikazanom na Slici 4.8 morali bi smo da uradimo spoj tabele Apartmana pa onda spoj Host-a koji se spaja dalje sa User tabelom da bi recimo izvukli username od Host-a (slično i za ostala polja). Dok u slučaju odvojene baze (read modela) možemo da vratimo objekat malte ne 1-1 iz baze. Ovim dobijamo kako na performansama tako i na jednostavnosti našeg softverskog rešenja.

```
_id: 1
GuestId: 3
GuestUsername: "guest"
HostId: 2
HostUsername: "host"
ApartmentId: 1
ApartmentTitle: "Great apartment, affordable"
ReservationStartDate: 2020-09-07T12:38:04.944+00:00
NumberOfNightsRented: 3
TotalCost: 249
ReservationState: "Accepted"
```


SLIKA 4.8 – PRIMER RESERVATION DOKUMENTA


```
_id: 1
FirstName: "Jotaro"
Gender: "Male"
LastName: "Kujo"
Password: "admin"
RoleName: "Administrator"
Username: "admin"
Banned: false
```


SLIKA 4.9 - PRIMER USERS DOKUMENTA


```
_id: 1
HostId: 2
IsHostBanned: false
ActivityState: "Active"
ApartmentType: "Full"
CheckInTime: "14:00:00"
CheckOutTime: "10:00:00"
› Location: Object
NumberOfGuests: 3
NumberOfRooms: 4
PricePerNight: 83
Title: "Great apartment, affordable"
Rating: 2.5
› Images: Array
› Amenities: Array
› ForRentalDates: Array
› AvailableDates: Array
⌄ Comments: Array
  ⌄ 0: Object
      _id: 1
      GuestId: 3
      GuestUsername: "guest"
      Rating: 4
      Text: "Had fun, enjoyable experience. Neighbours were kinda anoying tho."
      Approved: true
  › 1: Object
```


SLIKA 4.10 – PRIMER JEDNOG APARTMANA U MONGODB KOLEKCIJI

4.4 Replikatori

Kao što je to prethodno napomenuto, uvođenje nove baze podataka uz svoje prednosti donosi i mane. Jedna od tih mana jeste potreba za replikacijom, tj. sinhronizacijom podataka između baza (u slučaju CQRS-a se radi o jednostranoj replikaciji iz Command u Query stranu)

4.4.1 Vrste replikacija u CQRS-u

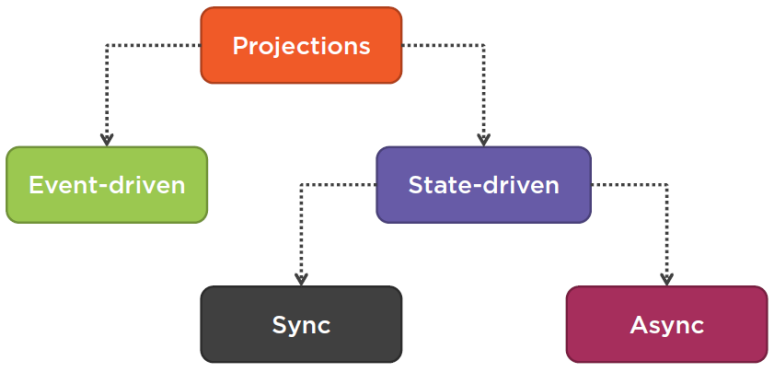
U zavisnosti od pokretača (trigger-a) mogu se izdvojiti dva ključna načina replikacije (sinhronizacije, projekcije) podataka: **Event-driven** i **State-driven**

Event-Driven – podrazumeva korišćenje Event-Bus-a ili slične tehnologije gde bi Command strana objavljivala događaje prilikom uspešnog izvršenja. Read strana bi se "slušala" ove događaje i prilikom njihovog pojavljivanja bi ažurirala svoje stanje.

Problem kod ovog rešenja zapravo nastaje ako se taj Event zagubi. U slučaju da je naša Command strana relacioni model, ne postoji način da se ti izgubljeni eventi obnove. Ovo znači da se javlja zahtev skladištenja Eventa u zasebnu bazu što čini Event-Driven projekciju usko vezanu sa napomenutim Event-Sourcing šablonom.

State-Driven – kako ime kaže Replikacija je inicirana izmenom stanja modela. Ovde se mogu napraviti dve pod-podele:

a) Sinhrona – nakon svake izmene se pokreće proces replikacije
b) Asinhrona – model se proširuje sa dodatnim flag poljem (npr. IsSyncNeeded) koji bi nam govorio da li je objekat izmenjen, dok bi neki pozadinski proces (Replikator) periodično vršio proveru ovog polja i po potrebi ažurirao bazu za čitanje. Treba napomenuti da nije potrebno dodavati IsSyncNeeded u svaki entitet već samo u one "glavne" (tj. Aggregate u kontekstu DDD), kao da je takođe moguće uvesti nove tabele za ovu namenu (umesto izmena entiteta)



SLIKA 4.11 - VRSTE SINHRONIZACIJA

4.4.2 Implementacija

U primeru rezervacije apartmana je odabrana asinhrona State-Driven projekcija sa replikatorima implentiranim putem .NET background service-a

Odabrano je rešenje gde se proširuje određeni podskup tabela sa IsSyncNeeded flagom. Međutim, za odabir kako će se ovaj flag postaviti postoji više načina:

a) Putem database triggera
b) Eksplicitno

Generalno se preporučuje eksplicitna opcija jer je eksplicitno rešenje uglavnom lakše za održavanje. Pristup putem database trigger-a se preporučuje kada nemamo dobru enkapsulaciju ili kontrolu nad izvornim kodom.

U projektu rezervacije apartmana je iskorišćena Change tracker funkcionalnost Entity Frameworka za veoma jednostavno podešavanje. Svaki entitet koji poseduje ovaj flag je proširen tako da nasleđuje *ISyncable* interfejs, što nam omogućuje da iz Change trackera unutar SaveChanges metode pristupimo tim entitetima i modifikujemo njihove vrednosti na jedan lak i brz način (Slika 4.12)

```
33 references | Danilo Novakovic, 4 days ago | 1 author, 2 changes | 0 exceptions
public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)
{
    var entities = ChangeTracker
        .Entries()
        .Where(e => e.Entity is ISyncable
                    && (e.State == EntityState.Added || e.State == EntityState.Modified));

    foreach (var entityEntry in entities)
    {
        var prevSyncNeeded = entityEntry.OriginalValues.GetValue<bool>(nameof(ISyncable.IsSyncNeeded));
        var currSyncNeeded = entityEntry.CurrentValues.GetValue<bool>(nameof(ISyncable.IsSyncNeeded));

        if (prevSyncNeeded && !currSyncNeeded)
            continue; // IsSyncNeeded was set to false by replicator

        ((ISyncable)entityEntry.Entity).IsSyncNeeded = true;
    }

    return base.SaveChangesAsync(cancellationToken);
}
```

SLIKA 4.12 – IMPLEMENTACIJA SAVECHANGES METODE DBCONTEXT KLASE

Treba napomenuti da prethodno pomenuta metoda funkcioniše samo ako je u aplikaciji korišćeno logičko brisanje. U slučaju da nije, identifikatore entiteta za brisanje je potrebno dodatno negde sačuvati (kompleksnost je veća te se preporučuje logičko brisanje za State-driven projekcije ovog tipa)

Što se tiče samih replikatora ASP.NET Core nam pruža podršku za registraciju pozadinskih servisa koje možemo registrovati putem *.AddHostedService<T>()* extension metode unutar *Startup.cs* fajla.

```
namespace Microsoft.Extensions.Hosting
{
    public abstract class BackgroundService : IHostedService, IDisposable
    {
        protected BackgroundService();

        public virtual void Dispose();
        public virtual Task StartAsync(CancellationToken cancellationToken);
        public virtual Task StopAsync(CancellationToken cancellationToken);
        protected abstract Task ExecuteAsync(CancellationToken stoppingToken);
    }
}
```

SLIKA 4.13 – PODRŽANE METODE APSTRAKTNE BACKGROUNDSERVICE KLASE

```
5 references | Danilo Novakovic, 3 days ago | 1 author, 1 change
public class ReservationReplicationService : BackgroundService
{
    private readonly ILogger<ReservationReplicationService> _logger;
    private readonly DbReplicationSettings _settings;
    private readonly IServiceScopeFactory _serviceScopeFactory;

    0 references | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
    public ReservationReplicationService(
        IOptions<DbReplicationSettings> settings,
        ILogger<ReservationReplicationService> logger,
        IServiceScopeFactory serviceScopeFactory)...

    3 references | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)...

    /// <summary>
    /// Synchronizes Reservations table from SQL (Command Model) to noSQL (Query Model)
    /// </summary>
    1 reference | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
    private async Task SyncReservationsAsync(CancellationToken stoppingToken)...

    1 reference | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
    private Task SyncReservationAsync(IQueryDbContext queryDb,
                                      Domain.Entities.Reservation dbReservation,
                                      CancellationToken stoppingToken)...

    1 reference | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
    private ReservationModel MapToQueryModel(Domain.Entities.Reservation dbReservation)...
}
```

SLIKA 4.14 – PRIMER INTERFEJSA JEDNOG REPLIKATORA

```
1 reference | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
private async Task SyncReservationsAsync(CancellationToken stoppingToken)
{
    using(var scope = _serviceScopeFactory.CreateScope())
    {
        var db = scope.ServiceProvider.GetRequiredService<IApartmentReservationDbContext>();
        var queryDb = scope.ServiceProvider.GetRequiredService<IQueryDbContext>();

        var dbReservations = await db.Reservations
            .Include(r => r.Guest).ThenInclude(g => g.User)
            .Include(r => r.Apartment).ThenInclude(a => a.Host).ThenInclude(h => h.User)
            .Where(r => r.IsSyncNeeded)
            .Take(_settings.MaxNumberOfEntitiesPerReplication)
            .ToListAsync(stoppingToken);

        var tasksToAwait = new List<Task>();

        foreach (var dbReservation in dbReservations)
        {
            Task task = SyncReservationAsync(queryDb, dbReservation, stoppingToken);

            tasksToAwait.Add(task);

            dbReservation.IsSyncNeeded = false;
        }

        tasksToAwait.Add(db.SaveChangesAsync(stoppingToken));

        await Task.WhenAll(tasksToAwait).ConfigureAwait(false);
    }
}
```

SLIKA 4.15 – PRIMER ALGORITMA ZA REPLIKACIJU VIŠE ENTITETA

```
1 reference | Danilo Novakovic, 3 days ago | 1 author, 1 change | 0 exceptions
private Task SyncReservationAsync(IQueryDbContext queryDb,
                                  Domain.Entities.Reservation dbReservation,
                                  CancellationToken stoppingToken)
{
    var replacement = MapToQueryModel(dbReservation);
    var filter = Builders<ReservationModel>.Filter.Eq(u => u.Id, replacement.Id);

    if (dbReservation.IsDeleted)
    {
        return queryDb.Reservations.DeleteOneAsync(filter, stoppingToken);
    }

    var options = new ReplaceOptions() { IsUpsert = true };
    return queryDb.Reservations.ReplaceOneAsync(filter, replacement, options, stoppingToken);
}
```

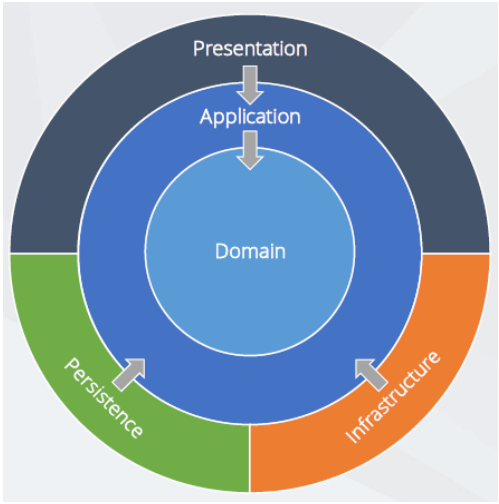
SLIKA 4.16 – PRIMER ALGORITMA ZA REPLIKACIJU JEDNOG ENTITETA

4.5 Višeslojna arhitektura

Serverska aplikacija je organizovana u sledećim slojevima (“Onion“ arhitektura):

- Domain (Query + Write)
- Application
- Persistence (Query + Write)
- Infrastructure
- Presentation

U nastavku teksta će svaki od ovih slojeva biti detaljno objašnjeni.

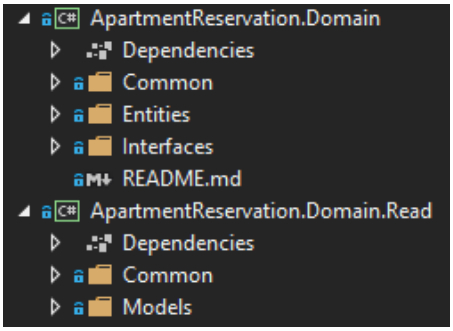


SLIKA 4.17 - SLOJEVI NA SERVERSKOJ STRANI

4.5.1 Domain layer

Domain sloj predstavlja centralni deo aplikacije. U njemu se nalaze entiteti, enumeracije, konstante, logika vezano za domenske entitete i slično (npr. domain exceptions). Da bi se dobila veća nezavisnost od tehnologije ovaj sloj bi trebalo da sadrži minimalan broj “framework dependent features“-a kao što su data anotacije i slično.

Iz potrebe CQRS-a za dva odvojena modela (komande i upite) ovaj sloj je izdvojen-a na dva dela - Domain (Commands) i Domain.Read (Queries)

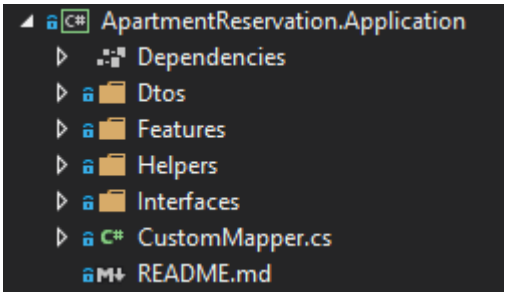


SLIKA 4.18 - PRIMER DOMAIN SLOJEVA

4.5.2 Application layer

Application, zajedno sa Domain slojem, čini ***Core*** sloj. Tj. svi ostali zavise od njega dok on ne zavisi od drugih. Kao ovakav Core sloj mora da poštuje D iz SOLID principa, tj. da se oslanja na apstrakciju a ne na konkretizaciju.

Primeri entiteta koji spadaju u ovaj sloj su Interfejsi, DTO, Komande, Custom Exceptions, Validators..



SLIKA 4.19 – PRIMER APPLICATION SLOJA

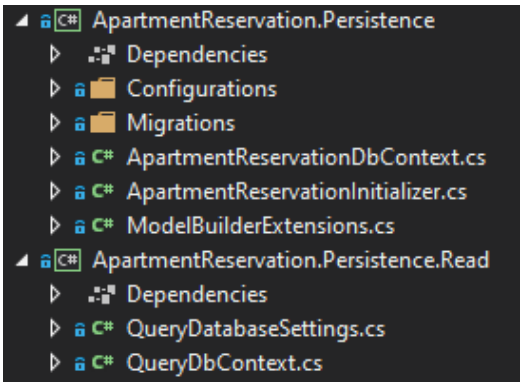
Glavna, ujedno I najsloženija komponenta ovog sloja predstavlja *Features* folder u kojem se nalaze Commands & Queries (sa svojim handleri-ma implementirani uz pomoć *MediatR* biblioteke)

4.5.3 Persistence layer

Persistence sloj sadrži implementacije interfejsa iz *Core* sloja vezano za čuvanje podataka sa bazom.

Kao što je bio slučaj sa Domain slojem, radi veće preglednosti, ovaj sloj je izdvojen na dva dela – Persistance (Commands) i Persitence.Read (Queries).

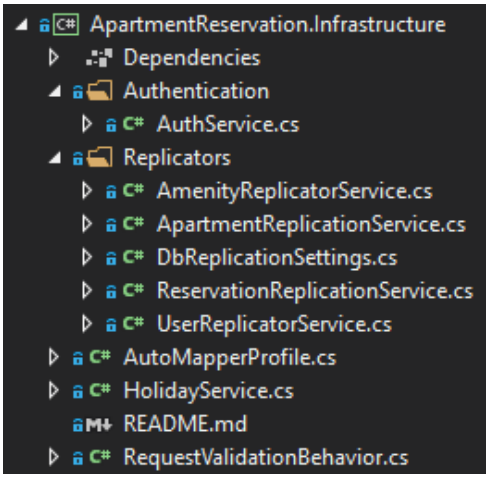
On u slučaju komandi sadrži klase vezane za Entity Framework, tj. DbContext, Migracije, Fluent API Konfiguracije, Initialize (Seed) metodu, Extenzije (za npr. ModelBuilder), dok je u slučaju upita reč o “wrapper“ klasama vezane za komunikaciju sa MongoDB (NoSQL) bazom.



SLIKA 4.20 - PRIMER PERSISTANCE SLOJEVA

4.5.4 Infrastructure layer

U ovom sloju se nalaze implementacije interfejsa iz *Core* dela koje se odnose na external api pozive (npr. File System, Email, … anything external) i ostale potrebe kao što su npr. autentifiakcija, replikacija...



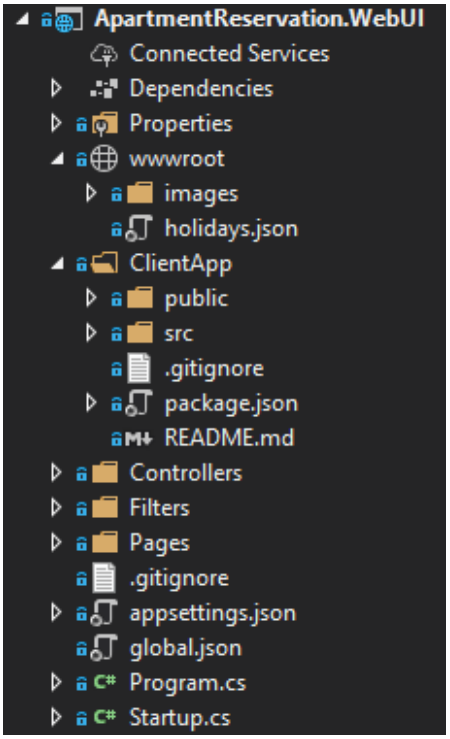
SLIKA 4.21 - PRIMER INFRASTRUCTURE SLOJA

4.5.5 Presentation layer

Presentation sloj predstavlja “najvišeg” člana u slojevitoj arhitekturi. On zavisi od svih ostalih slojeva, ali nijedan drugi ne zavisi od njega. Neke komponente

koje se mogu naći u ovom sloju su “SPA“ (tj. Klijentski) folder, API kontroleri, Exception filteri, wwwroot folder itd.

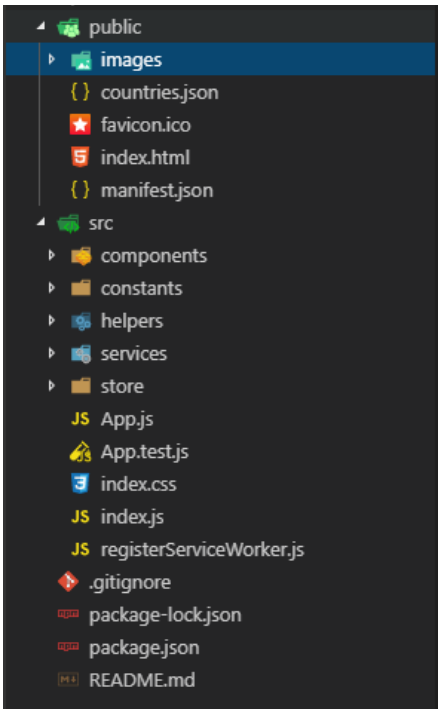
Za razliku od tradicijalnog pristupa, **kontroleri u ovom sloju su “tanki”** što znači da oni samo generišu komande (zahteve) koje obrađuju handleri iz aplikacijonog sloja sa eventualno dodatnom logikom vezanu za autorizaciju.



SLIKA 4.22 - IZGLED PRESENTATION SLOJA

5 Klijentska strana rešenja

Klijentska aplikacija je razvijena korišćenjem React-a. Unutar korenskog foldera možemo primetiti u suštini dva foldera od interesa, *Public* koji sadrži statičke fajlove (slično kao *wwwroot* na serverskoj strani) i *Src* u kojem se nalaze servisi, komponente itd. (kod sa logikom).



SLIKA 5.1 – STRUKTURA KLIJENTSKOG FOLDERA

Components folder sadrži komponente, tj. React klase i funkcije čija je uloga vizualni prikaz elemenata, formi i slično. One su te koje koriste servise, helpere i reducere (o njima reč u nastavku teksta) radi promene stanja.

Glavna (korenska) komponenta je u suštini App.js koja pretstavlja layout i mesto za registrovanje ruta za navigaciju. Dok je index.js mesto gde se taj isti App “obmota“ sa provajderom iz Redux biblioteke.

```jsx
export const ApartmentCarousel = ({ images, className }) => {
  return (
    <Carousel className={`apartment-carousel ${className || ""}`}>
      {images &&
        images.map((img, index) => {
          return (
            <Carousel.Item key={`apartment-carousel-item-${index}`}>
              <img className="carousel-image" src={img.uri} alt="apartment" />
            </Carousel.Item>
          );
        })}
    </Carousel>
  );
};
```

SLIKA 5.2 – PRIMER REACT KOMPONENTE

Constants – predstavlja lokaciju, tj. skup svih konstanti korišćenih u aplikaciji. Ovo je urađeno da bi se izbegli magični stringovi I brojevi u aplikaciji.

```jsx
export const Guest = "Guest";
export const Host = "Host";
export const Admin = "Administrator";

export const roleNames = {
  Guest: Guest,
  Host: Host,
  Admin: Admin
};
```

SLIKA 5.3 – PRIMER JEDNOG FAJLA SA KONSTANTAMA

Helpers - sadrži skup pomoćnih funkcija kao što su fje. za rad sa vremenom, parsiranjem, javascript objektima i slično.

```
export const makeCancelable = promise => {
  let rejectFn;

  const wrappedPromise = new Promise((resolve, reject) => {
    rejectFn = reject;

    Promise.resolve(promise)
      .then(resolve)
      .catch(reject);
  });

  wrappedPromise.cancel = () => {
    rejectFn({ canceled: true });
  };

  return wrappedPromise;
};
```

SLIKA 5.4 – PRIMER JEDNE POMOĆNE FUNKCIJE

Services – sadrži servise vezane za api pozive gde bi amenitiesService sadržao skup funkcija vezane za sadržaje apartmana, commentService za komentare itd.

```
export const handleResponse = createResponseHandler(logout);

export const userService = {
  login,
  logout,
  register,
  getAll,
  getById,
  update,
  updateCurrentUser,
  ban,
  unban,
  delete: _delete
};

export function login(username, password) {
  const requestOptions = {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ username, password })
  };

  return fetch(`api/Account/Login`, requestOptions)
    .then(handleResponse)
    .then(user => {
      // store user details in local storage to keep user logged in between page refreshes
      localStorage.setItem("user", JSON.stringify(user));

      return user;
    });
}
```

SLIKA 5.5 – PRIMER SERVISA ZA RAD SA KORISNIKOM

Poslednji ***Store*** folder obuhvata fajlove vezane za *Redux* biblioteku te sledi kratak pregled-opis iste.

5.1 Redux

Osnovna namena Redux biblioteke jeste da pomogne u radu sa stanjem. Ona uvodi par termina kao što su **store, actions, reducers**.

Store u suštini predstavlja model tj. objekat bez set-era (ovo je urađeno da bi se izbegli bugovi, tj. problemi koje uvode globalne promenljive). Da bi smo promenili stanje uveden je pojam **akcija**.

```
export const alertActions = {
  success,
  error,
  clear
};

function success(message) {
  return { type: alertConstants.SUCCESS, message };
}

function error(message) {
  return { type: alertConstants.ERROR, message };
}

function clear() {
  return { type: alertConstants.CLEAR };
}
```

SLIKA 5.6 – PRIMER AKCIJA

Akcija predstavlja objekat koji obavezno sadrži polje ***type*** (opisuje tip akcije kao npr. „ADD_USER”) i opciono dodatne parametre.

Dispatch-eri bi bile funkcije koje ovaj objekat dostavljaju komponentama koje bi ove akcije obradili. Te komponente se nazivaju **Reduceri**.

```
const alertReducer = (state = {}, action) => {
  switch (action.type) {
    case alertConstants.SUCCESS:
      return {
        type: "success",
        message: action.message
      };
    case alertConstants.ERROR:
      return {
        type: "danger",
        message: action.message
      };
    case alertConstants.CLEAR:
      return {};
    default:
      return state;
  }
};

export default alertReducer;
```

SLIKA 5.7 – PRIMER REDUCERA

Radi veće preglednosti i skalabilnosti takođe je moguće napraviti više *Reducera* u kojem slučaju bi trebalo napraviti jedan korenski reducer koji bi bio spoj ostalih. Funkcija koja ovo omogućava je ***combineReducers.***

```
import { combineReducers } from "redux";

const rootReducer = combineReducers({
  auth: authReducer,
  project: projectReducer,
  alert: alertReducer,
  apartment: apartmentReducer
});

export default rootReducer;
```

SLIKA 5.8 – PRIMER KOMBINOVANJA REDUCERA

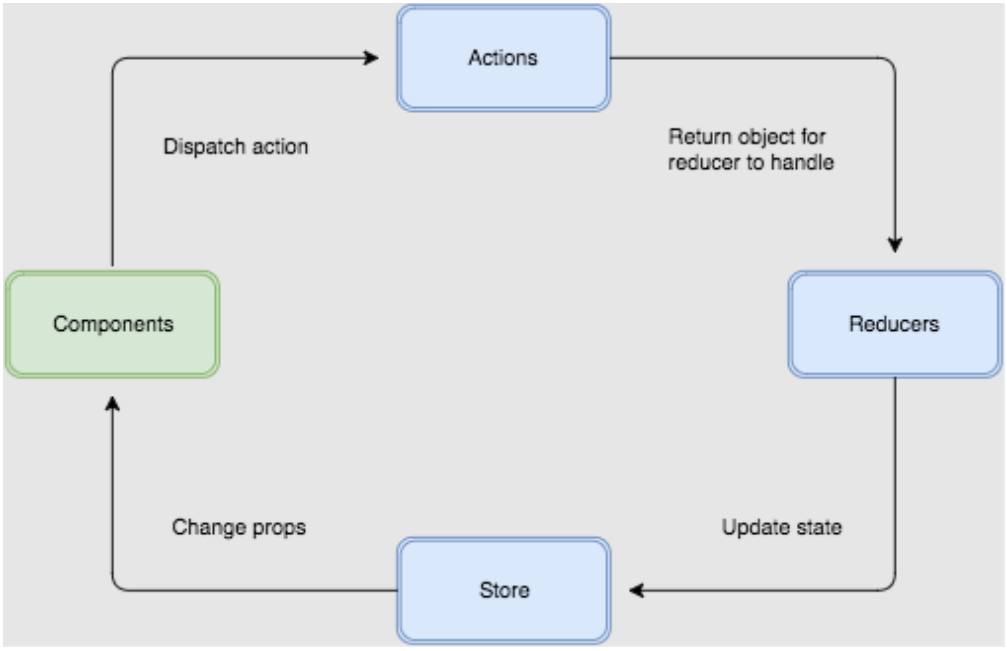
Sa obzirom da Redux nije vezan za React (može se koristiti i u običnom javascriptu bez ikakvog frameworka), uveden je **Provider**, tj. veza između Reduxa ili Reacta. Ova komponenta pruža mogućnost ubacivanja “store” promenljive kao parametre u konstruktoru (*dependency injection*) komponentama konektovanih putem **connect** funkcije iz React-Redux paketa.

```
const logger = createLogger();
const store = createStore(rootReducer, applyMiddleware(thunk, logger));

const app = () => {
  return (
    <Provider store={store}>
      <App />
    </Provider>
  );      You, a month ago • added redux to client app
};

ReactDOM.render(app(), rootElement);
```

SLIKA 5.9 – POVEZIVANJE REDUX BIBLIOTEKE SA REACT-OM



SLIKA 5.10 – TOK REACT-REDUX APLIKACIJE

6 Prikaz implementiranog rešenja

U ovom poglavlju aplikacija će biti opisana iz korisnikovog ugla.

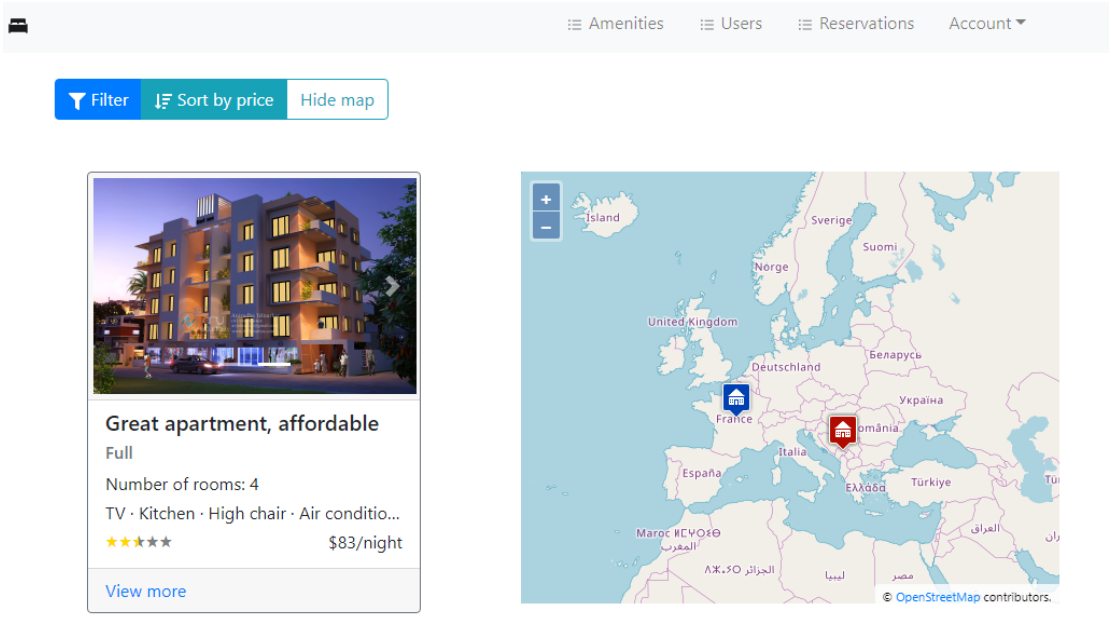
Kao što je već rečeno, postoje tri tipa korisnika, administrator, domaćin i gost, pa će tako biti objašnjeno kako oni koriste aplikaciju, respektivno.

Početni ekran aplikacije sadrži prikaz dostupnih aktivnih apartmana, korisnik koji nije ulogovan može da vrši pregled i sortiranje istih ali ne može da modifikuje apartmane niti da pravi rezervacije i slično.

U gornjem desnom uglu se nalaze dve opcije Sign up I Log In za prijavu korisnika. Korisnik putem registrovanja postaje Gost, dok domaćina može jedino administrator da doda, a samog administratora nije moguće programski dodati.

Nakon uspešne prijave se korisniku prikazuje ekran modifikovan u skladu sa njegovom ulogom.

6.1 Administrator



SLIKA 6.1 – POČETNA STRANA ZA ADMINISTRATORA

Za razliku od korisnika koji nije ulogovan, administrator ima takođe prikaz I neaktivnih apartmana (kao mogućnost modifikacije istih klikom na nekih od njih).

Navigacioni meni sa sobom takođe unosi nove opcije (tabove):

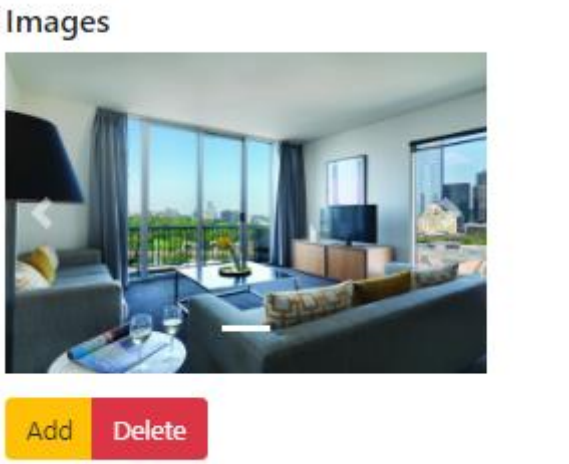
- **Amenities** - omogućava modifikaciju I dodavanje dostupnih sadržaja apartmana. Administratoru je dostupan tabelaran prikaz sa intuitivnim dugmićima Add, Edit i Delete putem kojih vrši modifikaciju
- **Users** - tabelaran prikaz korisnika sa dugmićima koji omogućavaju modifikaciju sadržaja. Međutim ono što ovaj prozor takođe omogućava jeste Filtriranje sadržaja putem *Filter* dugmeta
- **Reservations** - pregled svih rezervacija u sistemu, međutim nije mu dozvoljena modifikacija istih (samo domaćini I gost imaju mogućnost modifikovanja stanja rezervacije)

6.1.1 Apartman

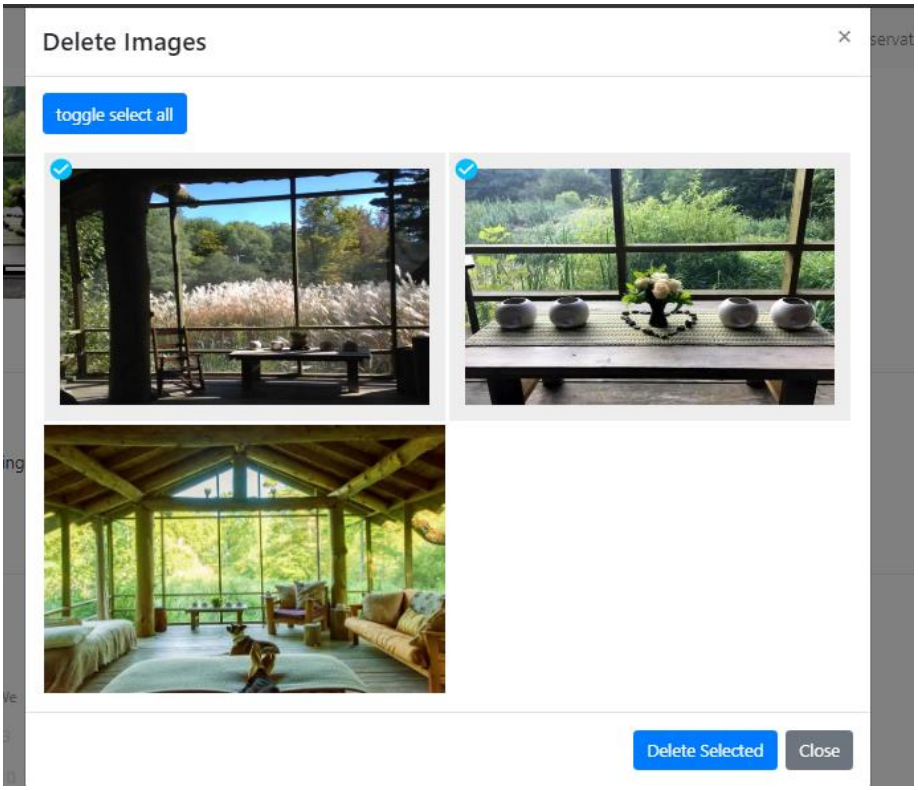
Klikom nekog apartmana sa početne stranice administrator ima prikaz detalja apartmana u sekcijama (zbog obima podataka koje entitet Apartman nosi sa sobom – bolji *user experience*) gde administrator ima mogućnost modifikaciju svake putem **Edit** dugmeta.

Sekcije:

1. kratak opis apartmana sa nekim ključnim informacijama kao što su Tip, broj soba, broj dozvoljenih gostiju, cena, vreme prijave I odjave, kao I lokacija (grad)
2. podršku za rad sa slikama. Aministratoru je dat prikaz trenutnih slika putem Carousel komponente (*slika 6.2*). Mogućnost dodavanja novih mu pruža **Add** dugme, dok mogućnost brisanja **Delete** (*Slika 6.3*)
3. sadržaji apartmana. Prikaz sadržaja je izvršen u vidu liste, dok je modifikacija implementirana putem auto-complete multi select liste (*slika 6.4*) u kojem administrator može selektovanjem komponente da dobije dropdown svih mogućih amenitiesa ili da kuca ima pa da dobije sugestiju (slično kao u pretraživaču). Brisanje nekog od sadržaja apartmana je moguće obaviti klikom na “x” dugme pored naziva istog.
4. Od preostalih sekcija su sekcija za **Komentare** (gde administrator ima samo pregled svih), za **Dostupne datume** noćenja, sekcija za **Lokaciju** koja pruža izmenu putem forme kao i putem klika na mapu (kada klikne na mapu automacki se ažurira forma).



SLIKA 6.2 – SEKCIJA ZA SLIKE APARTMANA



SLIKA 6.3 – FORMA ZA BRISANJE SLIKA

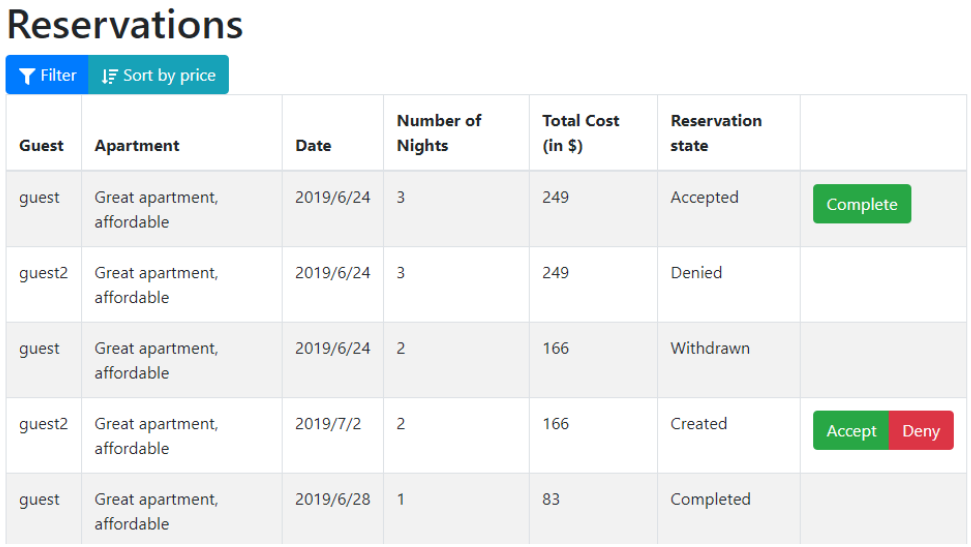
SLIKA 6.4 – FORMA ZA IZMENU SADRŽAJA APARTMANA

6.2 Domaćin

Za razliku od administratora koji ima prikaz svih aktivnih I neaktivnih apartmana, domaćin ima prikaz samo svojih apartmana.

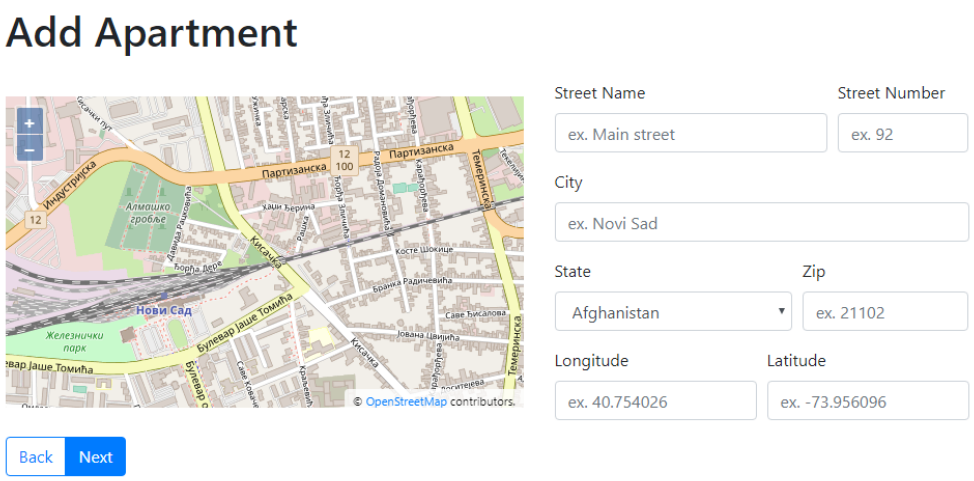
Klikom na nekih od apartmana domaćin ima identičan pregled kao administrator (isto ima sekcije koje može da modifikuje). Takođe mu je dozvojena modifikacija **svog** profila (ovo je dozvoljeno svim ulogovanim neblokiranim korisnicima).

Domaćin takođe ima pristup rezervacijama, međutim on može da vidi samo rezervacije nad njegovim apartmanima, takođe može da izvršava akcije nad rezervacijama u skladu sa svojom ulogom (objašnjeno u ranijim poglavljima).



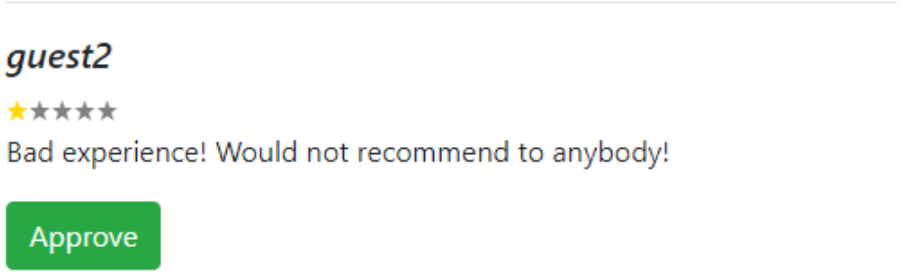
SLIKA 6.5 – PREGLED REZERVACIJA ZA APARTMANE DOMAĆINA

Još jedna od funkcionalnosti koje domaćin poseduje jeste dodavanje apartmana. Klikom na *Add Apartment* iz navigacione sekcije se domaćinu prikazuje forma sa više stranica, gde svaka strana odgovara sekciji apartmana (ovo je učinjeno da bi se poboljšao user experience). Korisnik ide na sledeću sekciju pritiskom na dugme **Next** dok mu je takođe pružena opcija da se vrati na prethodnu putem dugmeta **Back.**



SLIKA 6.6 – PRIMER JEDNE OD SEKCIJA U FORMI ZA DODAVANJE APARTMANA

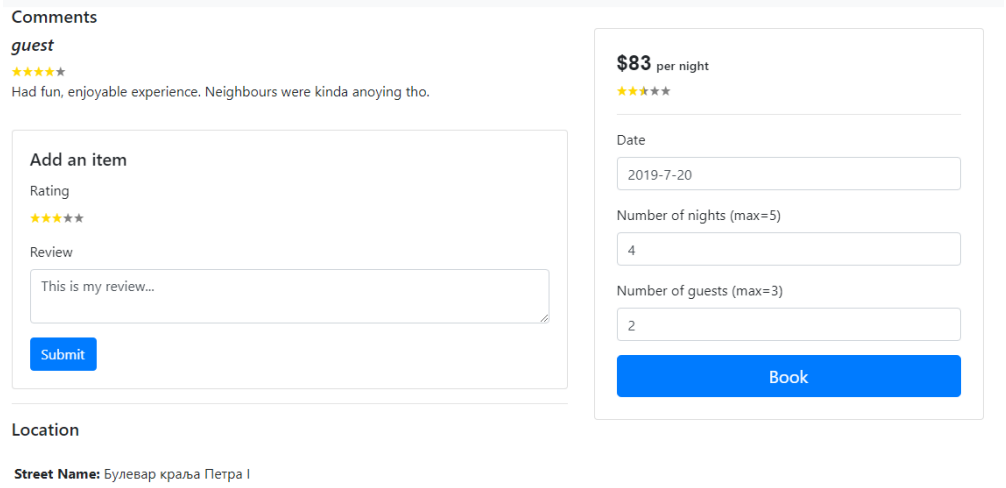
Dodatna funkcionalnost koju Domaćin takođe poseduje jeste kontrola da li hoće da mu se prikaže komentar apartmana na stranici ili ne. Kada Gost napiše komentar Domaćin ima opciju **Approve** (prihvati) posle koje komentar postaje vidljiv ostalim korisnicima.



SLIKA 6.7 – PRIMER KOMENTARA

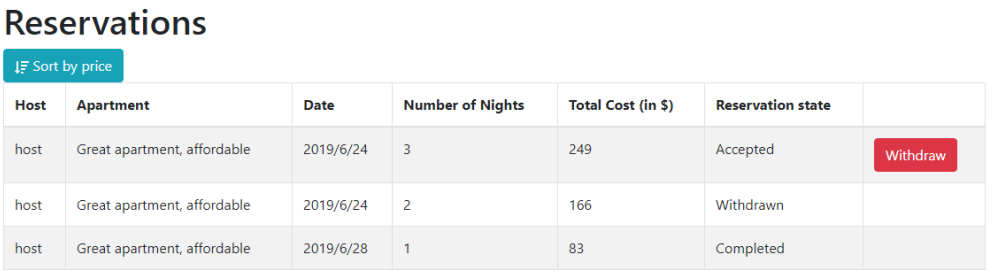
6.3 Gost

Početna strana domaćina se ne razlikuje skoro uopšte od ne ulogovanog korisnika. On vidi isto listu samo aktivnih apartmana, međutim klikom na nekih od njih gost ima sa strane “sticky“ formu putem koje može da rezerviše apartman, kao i mogućnost da doda komentar ukoliko je barem jedna od njegovih rezervacija za dati apartman u statusu *završena* (Completed).



SLIKA 6.8 – PRIKAZ APARTMANA ZA GOSTA

Gostu je takođe omogućen prikaz svih rezervacija koje je on rezervisao, kao i mogućnost izvršenja akcija nad tim rezervacijama u skladu sa svojom ulogom.



SLIKA 6.9 – PRIKAZ REZERVACIJA ZA GOSTA

7 ZAKLJUČAK

U ovom radu je objašnjen razvoj veb aplikacije za rezervaciju apartmana. Aplikacija se sastoji od dve manje aplikacije: klijentske strane, i serverske strane. Klijentska strana je rađena u *React*-u, serverska u *AspNetCore*-u.

U radu je data specifikacija i implementacija aplikacije. Objašnjena je višeslojna arhitektura i CQRS pristup implementiran putem MediatR biblioteke.

Budući pravci razvoja projekta bi obuhvatali optimizaciju replikatora, uvođenje paginacije, uvećanje apstrakcije na serverskoj strani (u vidu repository-a za svaku bazu pojedinačno ladi lakšeg proširivanja i testiranja), kao i bolje korišćenje "loggera", tj. zapisivanje kritičnih akcija, grešaka i sličnih aktivnosti što bi kao rezultat prouzrokovalo u efikasniji oporavak potencialnih problem koje bi korisnik (ili neko drugo lice) prijavio.

Moguće je takođe upariti CQRS sa Event Sourcing šablonom (u bazi se upisuju eventi umesto podataka – kada se servis digne pravi se model, nakon određenog broja eventa se pravi snapshot modela radi optimizacije) uz moguć prelazak na mikroservisnu arhitekturu gde bi čak Command i Query mogli da budu na odvojenim servisima što bi omogućilo npr. da imamo više pokrenutih instanci za čitanje naspram upisa.

Na klijentskoj strani bi takođe mogla biti uvedena neka vrsta "customer support-a" u kojoj bi klijent mogao da stupi u kontakt na neki način sa nadležnim licima.

8 LITERATURA

[1] https://en.wikipedia.org/wiki/Representational_state_transfer, ***REST***
[2] https://docs.microsoft.com/en-us/ef/core/, ***EF Core***
[3] https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core, ***ASP.NET Core***
[4] https://en.wikipedia.org/wiki/React_(web_framework), ***React***
[5] https://redux.js.org/, ***Redux***
[6] https://github.com/jbogard/MediatR/wiki, ***MediatR***
[7] https://docs.mongodb.com/drivers/csharp/, ***MongoDB.Driver C#***