**Redis Tema 1: "SnowTrack" - Praćenje kapaciteta na skijalištima**

**Opis funkcionalnosti:**

Web aplikacija omogućava korisnicima uvid u real-time podatke o stanju na skijalištima, uključujući:

* Broj slobodnih mesta na parkingu.
* Trenutno stanje staza (otvoreno/zatvoreno).

**Specifičnosti Redis-a:**

1. **Hash**:
   * Svako skijalište ima svoje ključeve sa informacijama o kapacitetu.
   * Primer:
   * "skijaliste:Kopaonik": {
   * "parking\_slobodna\_mesta": 25,
   * "staza\_otvorena": 15,
   * "staza\_zatvorena": 3
   * }
2. **Pub/Sub**:
   * Sistem šalje obaveštenja korisnicima kada se promeni stanje (npr. zatvaranje staze zbog lošeg vremena).
3. **Sorted Set**:
   * Rangiranje skijališta na osnovu poseta korisnika ili broja aktivnih staza.
   * Primer:
   * zadd "popularnost\_skijalista" 1200 "Kopaonik"
4. **TTL**:

**UI Funkcionalnosti:**

* Prikaz liste skijališta sa trenutnim podacima.
* Notifikacije o promenama (npr. obaveštenje o gužvama).
* Filter za pronalaženje skijališta sa najviše slobodnih parking mesta.

Naravno! Ako spojiš **SkiPlanner** sa Redis funkcionalnostima, na front-endu bi se trebale dodati nove sekcije i opcije kako bi korisnici mogli da iskoriste te funkcionalnosti. Evo kako bi to moglo izgledati i koje korake treba da preduzmeš:

**1. Nova funkcionalnost na front-endu**

Redis bi se koristio za real-time podatke i personalizovana iskustva, pa bi se na front-end dodale sledeće sekcije:

**a. Real-time kapacitet skijališta**

* **Nova stranica ili sekcija**: "Stanje na skijalištima".
* Prikaz liste svih skijališta sa informacijama u realnom vremenu:
  + Broj slobodnih parking mesta.
  + Status staza (otvorene/zatvorene).
  + Popularnost skijališta (broj posetilaca).

**Primer dizajna (UI):**

Skijalište | Parking mesta | Otvorene staze | Zatvorene staze | Popularnost

-------------------------------------------------------------------------------

Kopaonik | 25 | 15 | 3 | ⭐⭐⭐⭐

Tara | 10 | 10 | 5 | ⭐⭐⭐

Stara Planina | 50 | 20 | 0 | ⭐⭐⭐⭐⭐

* Koristi **real-time osvežavanje** pomoću WebSocket-a ili periodičnog fetch-ovanja Redis podataka sa servera.

**b. Notifikacije**

* Dodaj funkcionalnost za **obaveštenja u realnom vremenu**:
  + Korisnik dobija poruku kada:
    - Staza na skijalištu postane zatvorena.
    - Parking postane popunjen.
    - Upozorenje o gužvi.
  + UI prikazuje obaveštenja u vidu toast poruka ili banera.

**Primer:**

🔔 Staza "Jaram" na Kopaoniku je zatvorena zbog loših vremenskih uslova.

🔔 Parking na Stara Planina skijalištu je popunjen.

**c. Filter i rangiranje skijališta**

* Omogući korisnicima da **filtriraju skijališta**:
  + Sa najviše slobodnih parking mesta.
  + Sa najviše otvorenih staza.
  + Po popularnosti.

**Primer UI elementa za filter:**

Filtriraj skijališta po:

[ ] Najviše slobodnih parking mesta

[ ] Najviše otvorenih staza

[ ] Najpopularnija

[Primeni filter]

**2. Način integracije Redis funkcionalnosti u postojeći SkiPlanner front-end**

1. **Dodavanje API ruta**:
   * Backend bi trebao imati dodatne API endpoint-e za Redis funkcionalnosti, npr.:
     + GET /realTimeStatus: Vraća real-time podatke o skijalištima.
     + GET /notifications: Vraća obaveštenja za korisnika.
     + POST /subscribeNotifications: Omogućava korisniku da se pretplati na obaveštenja.
2. **Dodavanje novih sekcija na front-end**:
   * Na stranici za pregled skijališta dodaj **tabelu za real-time podatke**.
   * Na početnu stranicu dodaj sekciju za popularna skijališta (npr. "Top 5 skijališta trenutno").
3. **Implementacija real-time obaveštenja**:
   * Koristi WebSocket za prikaz notifikacija kada Redis Pub/Sub primi poruku.
   * Na primer, kada se status staze promeni, Redis Pub/Sub šalje signal backend-u, koji prosleđuje obaveštenje korisniku na front-end.

**3. Tehnički koraci za front-end integraciju**

**a. Real-time podaci**

* Na front-endu koristi framework kao što je **React Query** za povlačenje podataka sa servera.
* Primer fetch logike:

import { useQuery } from 'react-query';

const fetchRealTimeStatus = async () => {

const response = await fetch('/api/realTimeStatus');

return response.json();

};

const RealTimeStatus = () => {

const { data, isLoading } = useQuery('realTimeStatus', fetchRealTimeStatus);

if (isLoading) return <div>Loading...</div>;

return (

<table>

<thead>

<tr>

<th>Skijalište</th>

<th>Parking mesta</th>

<th>Otvorene staze</th>

<th>Popularnost</th>

</tr>

</thead>

<tbody>

{data.map((ski) => (

<tr key={ski.id}>

<td>{ski.name}</td>

<td>{ski.parking}</td>

<td>{ski.openTrails}</td>

<td>{ski.popularity}</td>

</tr>

))}

</tbody>

</table>

);

};

**b. Notifikacije**

* Koristi WebSocket ili **Server-Sent Events (SSE)** za notifikacije.
* Primer korišćenja WebSocket-a:

const NotificationSystem = () => {

useEffect(() => {

const socket = new WebSocket('ws://localhost:8080/notifications');

socket.onmessage = (event) => {

alert(`🔔 ${event.data}`);

};

return () => socket.close();

}, []);

return null;

};

**4. Predlog UI arhitekture**

1. **Početna stranica**:
   * Sekcija sa "Najpopularnijim skijalištima".
   * Prikaz real-time statusa parkinga i staza.
2. **Stranica skijališta**:
   * Detaljni podaci o parkingu i stazama za odabrano skijalište.
   * Mogućnost prijave za obaveštenja.
3. **Filter i pretraga**:
   * Korisnik može filtrirati skijališta na osnovu real-time podataka.

Da, podaci u **Neo4j** i **Redis** bazama mogu se potpuno odvojiti, i oba sistema mogu raditi nezavisno, ako to odgovara tvojoj arhitekturi i zahtevu aplikacije. Ovaj pristup je korisniji kada želiš da jasno razdvojiš odgovornosti i izbegneš zavisnost između dve baze podataka. Evo kako to može da funkcioniše:

**1. Potpuno odvajanje podataka između Neo4j i Redis**

* **Neo4j**:
  + Služi isključivo za **strukturalne i relacione podatke**.
  + Informacije o skijalištima, stazama, hotelima i restoranima ostaju u Neo4j-u.
  + Neo4j ne sadrži dinamičke podatke poput broja slobodnih mesta ili statusa staza.
* **Redis**:
  + Služi isključivo za **real-time podatke**.
  + Sadrži informacije kao što su:
    - Broj slobodnih parking mesta.
    - Trenutni broj otvorenih i zatvorenih staza.
    - Notifikacije i rangiranje skijališta (broj skijasa).

**2. Kako se upravlja nezavisnim podacima?**

**Neo4j podaci:**

* Kada korisnik dodaje novo skijalište, informacije kao što su naziv, lokacija, kapacitet skijališta itd. čuvaju se u Neo4j-u.
* Na primer:
* {
* "Naziv": "Kopaonik",
* "Lokacija": "Srbija",
* "KapacitetHotela": 1000,
* "UkupnoStaza": 25
* }

**Redis podaci:**

* Dinamički podaci za novo skijalište dodaju se u Redis nezavisno od Neo4j-a.
* Na primer:
* {
* "parking\_slobodna\_mesta": 50,
* "staza\_otvorena": 10,
* "staza\_zatvorena": 15
* }

**3. Prednosti potpunog odvajanja**

1. **Manja zavisnost između baza**:
   * Podaci u Redis-u i Neo4j-u se ažuriraju nezavisno, bez sinhronizacije između baza.
   * Ovo smanjuje kompleksnost sistema.
2. **Fleksibilnost**:
   * Neo4j se koristi samo za podatke koji zahtevaju grafe i relacije, dok Redis pokriva real-time potrebe.
   * Redis podaci se mogu resetovati ili izmeniti bez uticaja na Neo4j podatke.
3. **Jednostavnije skaliranje**:
   * Redis se može skalirati nezavisno za brze real-time zahteve.
   * Neo4j ostaje optimizovan za upite grafova i odnosa.

**4. Arhitektura nezavisnog rukovanja podacima**

**Dodavanje novog skijališta**

* Kada korisnik doda novo skijalište:
  1. Osnovni podaci idu u Neo4j:
  2. await \_neo4jService.DodajSkijaliste(new Skijaliste { Naziv = "Kopaonik", Lokacija = "Srbija", UkupnoStaza = 25 });
  3. Dinamički podaci se kreiraju u Redis-u:
  4. await \_redisService.InicijalizujKapacitet("Kopaonik", 50, 10, 15);

**Dohvatanje podataka za prikaz**

* Podaci se nezavisno dohvataju iz obe baze:
  1. **Osnovne informacije**:
     + Dohvataju se iz Neo4j-a.
  2. var skijaliste = await \_neo4jService.GetSkijaliste("Kopaonik");
  3. **Real-time podaci**:
     + Dohvataju se iz Redis-a.
  4. var kapacitet = await \_redisService.GetKapacitet("Kopaonik");

**Prikaz kombinovanih podataka na frontu**

Na frontendu, podaci iz obe baze mogu se kombinovati na nivou aplikacije:

* **Neo4j podaci**: Naziv, lokacija, kapacitet.
* **Redis podaci**: Broj slobodnih parking mesta, otvorene i zatvorene staze.

**6. Kada koristiti potpuno odvajanje?**

1. **Nema potrebe za direktnom integracijom**:
   * Ako podaci u Redis-u i Neo4j-u nisu direktno zavisni jedni od drugih.
   * Ako je Redis fokusiran na real-time podatke i Neo4j na relacione podatke.
2. **Jasna podela odgovornosti**:
   * Jedna baza se ne preopterećuje zahtevima koji nisu u njenoj specijalizaciji.
3. **Manji rizik od sinhronizacije**:
   * Nema potrebe za složenom logikom za ažuriranje između baza.

Da li ovakav pristup odgovara tvojoj aplikaciji? 😊

 **Redis Pub/Sub za real-time notifikacije**:

* Redis podržava mehanizam **Pub/Sub** (publish/subscribe) za obaveštenja u realnom vremenu. Potrebno je omogućiti:
  + Da se određene promene (npr. zatvaranje staza ili popunjen parking) šalju kao poruke Redis kanalu.
  + Klijenti mogu slušati te poruke kako bi prikazali obaveštenja korisnicima.

 **ASP.NET Core SignalR za notifikacije na frontendu**:

* **SignalR** je odličan izbor za slanje notifikacija sa servera ka klijentima u realnom vremenu.
* Kada Redis Pub/Sub primi poruku, server koristi **SignalR** da proslijedi obaveštenje korisnicima.

 **UI prilagodba**:

* Na frontendu treba dodati komponentu za prikaz **toast poruka** ili **banera** koji prikazuju obaveštenja.