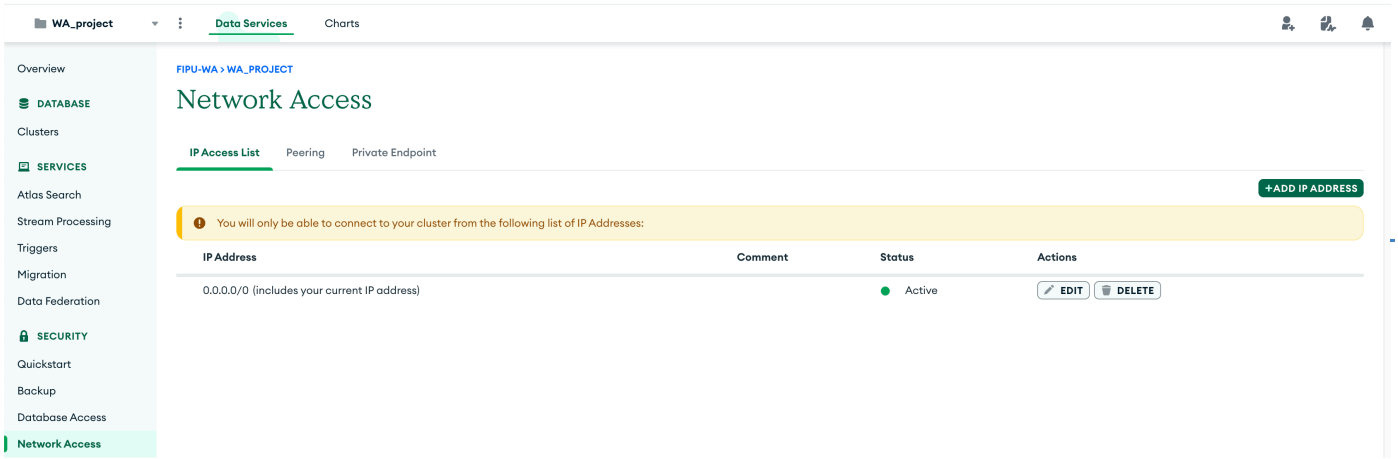
Primjer 2. kolokvija - Web aplikacije

Prije početka pisanja kolokvija studenti su **dužni provjeriti** jesu li na računalu instalirani **Node.js, VS Code i Postman**. Bez ovih alata, nemoguće je uspješno pripremiti se za kolokvij. Umjesto Postmana, moguće je koristiti i VS Code ekstenziju Thunder Client ili neki treći HTTP klijent.

VAŽNO! Potrebno je pripremiti MongoDB Atlas cluster te generirati osobni **Connection String** i **lozinku** za povezivanje preko **Node.js drivera**.



> **PREPORUKA**: Odaberite Security -> Network Access -> Add IP Address -> omogućite pristup Atlasu sa svih IP adresa (`0.0.0.0/0`)

Jednom kad ste pripremili sve potrebne alate, možete klonirati repozitorij `lukablaskovic/wa-final-template` koji ćete koristiti i na kolokviju. Predložak ima sve instalirane pakete koji vam trebaju, uključujući i `db.js` datoteku kojom se povezujete na vaš MongoDB Atlas.

Potrebno je dodati sljedeće varijable u `.env` datoteku:

```
MONGO_URI=vaš_mongo_connection_string_s_lozinkom
DB_NAME=wa_final
```

Drugi kolokvij obuhvaća **ukupno gradivo iz vježbi**, a studenti mogu ostvariti maksimalno **60 bodova iz kontinuiranog praćenja**.

Primjer kako će izgledati 2. kolokvij

1. Definiranje skupine endpointa `drinks.js` (10 bodova)

Unutar `routes/drinks.js` definirajte ruter na kojeg ćete implementirati skupinu endpointa resursa `drinks`.

- **1.1 Implementirajte rutu GET** `/` koja **vraća sve napitke** iz Mongo kolekcije `drinks` u obliku JSON polja, uz odgovarajući statusni kod i poruku.
  - Obradite slučaj ako je kolekcija prazna, vratite odgovarajući statusni kod i poruku.
- **1.2 Implementirajte rutu POST** `/` koja **dodaje novi napitak (1)** u Mongo kolekciju `drinks` i vraća novi napitak u obliku JSON polja, uz odgovarajući statusni kod i poruku.
  - Ruta mora korisniku vratiti u tijelu odgovora novi napitak koji je dodan u kolekciju s dodanim `_id` poljem.
  - Ako je proslijeđeni JSON objekt prazan, vratite odgovarajući statusni kod i poruku. U ovom zadatku ne morate još validirati ulazne podatke.

Primjer JSON objekta za **dodavanje novog napitka**:

```
{
  "naziv": "Coca-Cola",
  "zapremina": 0.2,
  "cijena": 4.5,
  "kolicina": 100
}
```

2. Validacija ulaznih podataka prilikom (10 bodova)

- **2.1 Implementirajte ručno validaciju ulaznih podataka** prilikom dodavanja novog napitka u kolekciju `drinks`. Validirajte sljedeće podatke:
  - `naziv` - obavezan, tipa `string`, minimalne duljine `3` znaka, a maksimalne duljine `50` znakova.
  - `zapremina` - obavezna, tipa `number`, minimalne vrijednosti `0.1`.
  - `cijena` - obavezna, tipa `number`, minimalne vrijednosti `0.5`.
  - `kolicina` - obavezna, tipa `number`, minimalne vrijednosti `50`.
    Ako korisnik proslijedi neispravne podatke, vratite odgovarajući statusni kod i poruku.
- **2.2 Implementirajte rutu GET** `/:id` koja **vraća jedan napitak** iz kolekcije `drinks` prema identifikatoru `id` u obliku JSON polja, uz odgovarajući statusni kod i poruku.
  - Ako napitak s proslijeđenim ispravnim identifikatorom ne postoji, vratite odgovarajući statusni kod i poruku.
  - Ako korisnik proslijedi neispravan identifikator (duljina stringa različita od `24`) vratite odgovarajući statusni kod i poruku.

3. Implementacija rute za dodavanje više napitaka i *middlewarea* (15 bodova)

U postojećem endpointu za dodavanje, implementirajte sljedeće funkcionalnosti:

- **3.1 Dodajte mogućnost dodavanja više napitaka** odjednom u kolekciju `drinks` na način da korisnik prosljeđuje polje objekata u JSON formatu.

Primjer JSON polja objekata za **dodavanje više napitaka**:

```
[
  {
    "naziv": "Coca-Cola",
    "zapremina": 0.2,
    "cijena": 4.5,
    "kolicina": 100
  },
  {
    "naziv": "Fanta",
    "zapremina": 0.2,
    "cijena": 4.0,
    "kolicina": 150
  }
]
```

- **3.2 Implementirajte middleware** naziva `validateDrink` koji će se koristiti prilikom dodavanja novih napitaka (ili jednog napitka) u kolekciju `drinks`.
  Middleware treba raditi sljedeće:
  - U dijeljeni objekt `req` dodajte atribut `type` koji će sadržavati informaciju radi li se o **jednom objektu** ili **polju objekata**.
  - Ako je proslijeđen **objekt**, validirajte ga prema pravilima iz **Zadatka 2**.
    - Ako je **objekt** prazan ili validacija nije uspješna, vratite odgovarajući statusni kod i poruku.
  - Ako je proslijeđeno **polje objekata**, validirajte svaki objekt prema pravilima iz **Zadatka 2**.
    - Ako je **polje objekata** prazno ili validacija nije uspješna za barem jedan objekt, vratite odgovarajući statusni kod i poruku.
  - Ako je sve uspješno, nastavite dalje izvođenje, odnosno izađite iz *middlewarea*

Uključite *middleware* `validateDrink` u postojeću rutu za dodavanje novih napitaka te dodajte jedan ili više napitaka ovisno o vrijednosti atributa `req.type`.

Napomena: sva validacija podataka mora biti implementirana u middlewareu `validateDrink`, dok u implementaciji rute mora biti samo implementacija dodavanja u bazu ovisno o tipu ulaznih podataka.

4. Implementacija ruta za dodavanje korisnika (10 bodova)

Dodajte novi router u `users.js` u direktorij `routes`:

- **4.1 Implementirajte rutu POST** `/` koja dodaje novog korisnika u kolekciju `users`.

    - Korisnik mora sadržavati ključeve `username`, `password` i `email`.
    - Implementirajte funkciju `hashPassword(plain_text, salt_rounds)` koja će hashirati lozinku korisnika korisnika koristeći `bcrypt` biblioteku za dani broj `salt_rounds`.
    - Koristeći funkciju `hashPassword`, hashirajte lozinku korisnika **prije** spremanja u MongoDB kolekciju.
    - Koristeći biblioteku `express-validator` **implementirajte validaciju korisničkih podataka**:
        - `username` - obavezan, tipa `string`, minimalne duljine 3 znaka, a maksimalne duljine 20 znakova.
        - `password` - obavezan, tipa `string`, minimalne duljine 8 znakova te isključivo alfanumeričkih znakova.
        - `email` - obavezan, tipa `string`, mora sadržavati `@` i `.` znakove.
        - Pogreške obradite koristeći `validationResult(req)`.
- **4.2 Implementirajte funkciju** `comparePassword(plain_text, hashed_password)` koja će uspoređivati plain text lozinku s hashiranom lozinkom iz baze i vratiti `true` ako su lozinke jednake, inače `false`.

Prebacite implementacije funkcija `hashPassword` i `comparePassword` u zasebnu datoteku `auth.js`.

5. Implementacija rute za prijavu korisnika (15 bodova)

- **5.1 Unutar postojećeg routera** `users.js` implementirajte rutu **POST** `/login` koja će omogućiti korisniku **prijavu koristeći korisničko ime i lozinku**.
    - Korisnik mora proslijediti korisničko ime (`username`) i lozinku (`password`) u tijelu zahtjeva.
    - Validirajte ulazne podatke koristeći biblioteku `express-validator` te obradite pogreške koristeći `validationResult(req)` na isti način kao u **4.1 Zadatku**.
    - Provjerite prvo postoji li korisnik u bazi pretraživanjem korisničkog imena, ako ne postoji vratite odgovarajući statusni kod i poruku.
    - Ako korisnik postoji, koristeći funkciju `comparePassword` iz **4. Zadatka** usporedite lozinke; **ako se lozinke ne podudaraju** vratite odgovarajući statusni kod i poruku.
    - **Ako se lozinke podudaraju**, generirajte JWT token koristeći `jsonwebtoken` biblioteku i **potpišite novi JWT koristeći tajni ključ** iz `.env` datoteke koji je oblika vašeg datuma rođenja, npr. `2002-01-01`. Token mora biti potpisan na **1 sat**.
    - U tijelu odgovora **vratite generirani JWT token**.
- **5.2 Implementirajte *middleware*** `authoriseUser` **unutar nove datoteke** `middleware.js` koja će nastaviti izvođenje samo ako korisnik u zaglavljima HTTP zahtjeva ima ispravan (validan) JWT token.
    - Potrebno je provjeriti zaglavlje `Authorization` te izvući `Bearer` token iz njega. Klijent šalje JWT toke u obliku: `Bearer <JWT_token>`
    - Ispravnost tokena provjerite odgovarajućom metodom iz `jsonwebtoken` biblioteke i privatnim ključem. Ako token nije ispravan, vratite odgovarajući statusni kod i poruku.
    - Ako je token ispravan, u dijeljeni objekt `req` dodajte atribut `user` koji će sadržavati korisničko ime korisnika **dekodirano iz JWT-a**.