Oppgave 2.1

Skal dokumentere hvilke API-endepunkter (ressurser) som skal brukes.

Oppgaven består av ulike API-endepunkter, vi har først brukt den tildelte API'en fra Marius, https://webapp-api.vercel.app/api/users denne har vi benyttet i seed filen.

Neste vi brukte er localhost:3000/users – Der vil brukeren ha muligheten til å opprette en bruker med følgende detaljer. ID, gender og type sport. Ved opprettelsen av bruker vil den bli lagt til i vår database, under localhost:3000/api

Da brukeren skal opprette en økt så vil de følge api endepunktet localhost:3000/users/newActivity. Her skal brukeren legge til dato for økten, økt navnet, tags som beskriver økten, hvilke type sport, hvor lenge den varer og intensiteten på økten.

For å opprette en konkurranse så blir brukeren sendt til localhost:3000/users/competition. Der skal brukeren legge til konkurranse navn, dato, sted, mål, type konkurranse, prioritet og til slutt legge til kommentar. Vi har lagt til en alert som skal vises når den samme brukeren forsøker å opprette mer enn 3 konkurranser i samme år, det samme gjelder for treningsmål.

Til slutt har vi det siste API'et hvor brukeren har muligheten til å opprette treningsmål localhost:3000/users/session

Skal dokumentere hvilke HTTP-verb som er tilgjengelige for ressursen, og hva slags forespørsler de skal håndtere.

De http-verbene som er tilgjengelig på denne oppgaven er POST, GET i route filen i API, og PATCH og GET i route.ts under mappen «[id]» under API mappen.

I route filen for startsiden bruker vi GET og POST og route filen for ID bruker vi GET og PATCH.

GET forespørselen bruker findmany funksjon fra prisma for å hente data fra databasen. Den dataen inkluderer aktiviteter og hver aktivitet inkluderer intervaller. Når dataen blir hentet så sender den en JSON response som inneholder den hentede dataen med statuskode 200, som indikerer at det ble gjennomført.

POST funksjonen forventer en JSON-payload som representerer en bruker (som angitt av User-type). Den henter JSON-kroppen fra den kommende forespørsel ved at vi bruker request.json(). Deretter benytter vi .create() som er en prisma metode for å opprette en ny bruker i databasen ved å bruke den mottatte dataen.

Ved å ha opprettet brukeren, vil den sende et JSON svar som inneholder den nye brukerinformasjonen med en statuskode på 200 som indikerer at forespørselen ble godkjent.

I route.ts som er under [id] mappen. Har vi en GET-funksjon sin tar imot en forespørsel og et objekt som inneholder parametere med en ID som streng. Funksjonen søker i databasen ved hjelp av Prisma sin findUnique() metode for å finne en unik bruker basert på den spesifikke ID-en som er gitt i parametrene.

Den inkluderer brukerens aktiviteter, hvor hver aktivitet inneholder intervaller. Det samme skjer her, når dataen er hentet vellykket, sender den et JSON svar som inneholder informasjonen om den spesifikke brukeren, samt en statuskode på 200.

PATCH-funksjonen håndterer forespørsler og parametere som inneholder ID som string.

Først forventer funksjonen en JSON-payload som representerer en aktivitet. Deretter henter den JSON-kroppen fra den innkommende forespørselen ved bruk av request.json(). Deretter forsøker den å oppdatere en bruker i databasen basert på den spesifikke ID-en som er gitt i parametrene. Dersom brukeren ikke eksisterer, så vil den returnere en melding om at brukeren ikke finnes med en statuskode på 404..

Dersom brukeren eksisterer, blir aktiviteten opprettet og knyttet til brukeren ved hjelp av Prisma sin update() metode. Det opprettes også tilhørende intervaller for denne aktiviteten.

Ved oppdatering så sender den et JSON svar som inneholder den oppdaterte brukerinformasjonen med statuskode på 200.

Som catch på errorene så har vi inkludert en console.error som skal gi feil i consolen!

Skal dokumentere hvilke sider (URL-er) som skal benytte de ulike API-ene og gi en overordnet beskrivelse av hva som kan utføres på hver side. Dette inkluderer opprettelse av sider i appen og en grov beskrivelse av hvilke funksjoner som skal være tilgjengelige på disse sidene.

- /users bruker api/route.ts [GET funksjonen som henter alle verdiene, samt POST til å lage en utøver/bruker]
- /users/competition bruker api/competition
- /users/newActivity bruker api/newActivity
- /users/session bruker api/session

I tabellen med utøvere så er det en knapp der man legger til økter for hver utøver. URL-en til denne er /users/{user.id} (user.id er ikke det same som user id). På denne siden kan man se hvilke aktiviteter som utøveren har. Er en knapp på denne siden som tar oss til enn annen side der man kan legge til aktiveter til utøveren.

Skal dokumentere datamodellen og bakgrunnen for denne modellen. Samt typene som skal benyttes for de ulike feltene (eks. Data, String, Boolean, Int m.fl)

Bildene under viser hvordan vi har satt opp vår datamodell. Måten vi har satt det opp på har vært i forhold til hvordan den tildelte API'en så ut. Vi har noen felter inkludert som inneholder spørsmålstegn som indikerer at feltet er valgfritt. Fra User modellen kan vi se at id har en unik id som blir autogenerert ved bruk av uuid. Samt har vi «aktiviteter» den kan inneholde en liste med aktiviteter.

```
model User {
  id         String     @id @unique @default(uuid())
  userId     String     @unique
  gender     String
  sport      String
  meta       Meta?
  activities Activity[]
}
```

I meta modellen har vi samme datatype for id, vi har inkludert User typen som skal lage en relasjon ved bruk av fields som sier det er userId, og hvordan de to tabellene er koblet sammen, som er id, hentet fra user tabellen. Det blir som fremmednøkkel.

```
model Meta {
  id        String @id @default(uuid())
  heartrate Int?
  watt      Int?
  speed     Int?
  User      User   @relation(fields: [userId], references: [id])
  userId    String @unique
}
```

Det samme har vi gjort med de andre modellene vi viser til under. Både datatypen og relasjonen mellom disse.

```
model Activity {
  id        String     @id @unique @default(uuid())
  date      DateTime?
  name      String?
  tags      String?
  sport     String?
  User      User?      @relation(fields: [userId], references: [id])
  userId    String?
  intervals Interval[]
  createdAt DateTime   @default(now())
  updatedAt DateTime   @updatedAt
}
```

```
model Interval {
  id         String    @id @unique @default(uuid())
  duration   Int
  intensity  Int
  Activity   Activity? @relation(fields: [activityId], references: [id])
  activityId String?
}
```