**Treningsapp – Oppgave 1**

**Navn:** Sidney **Dato:** 15. mai 2025 **GitHub repository:** <https://github.com/07Siddi/Treningsapp>

Innholdsfortegnelse

[Oppgave 1: Begrepsforklaring 2](#_Toc198798628)

[Oppgave 2: Databasemodell 3](#_Toc198798629)

[Forutsetninger 3](#_Toc198798630)

[Tabeller og attributter 3](#_Toc198798631)

[Datamodellen 4](#_Toc198798632)

[Oppgave 3: Hente og vise data 5](#_Toc198798633)

[1. Visning av data i HTML-format 5](#_Toc198798634)

[SQL-spørring 5](#_Toc198798635)

[Forklaring av SQL-begreper 6](#_Toc198798636)

[Behandling og visning 7](#_Toc198798637)

[Bruk av Docker 8](#_Toc198798638)

[2. Rå data i JSON-format 8](#_Toc198798639)

[Eksempel på kode 8](#_Toc198798640)

[Oppsummering 9](#_Toc198798641)

[Oppgave 4: Versjonskontroll 10](#_Toc198798642)

# Oppgave 1: Begrepsforklaring

**Versjonskontrollsystem**

Et versjonskontrollsystem hjelper deg å lagre ulike versjoner av koden. Det gjør at du kan samarbeide med andre uten å slette hverandres arbeid. Eksempel: Git. (GitHub, GitLab)

**Rammeverk**

Et rammeverk er en slags "verktøykasse" for programmering. Det gir deg ferdige funksjoner og maler, så du slipper å starte helt fra bunnen. Eksempler: React, Django.

**Frontend**

Frontend er det brukeren ser og bruker, for eksempel knapper og menyer. Det lages med HTML, CSS og JavaScript.

**Backend**

Backend er den delen som kjører i bakgrunnen. Den snakker med databasen og gjør alt det tekniske som brukeren ikke ser.

**Datatype**

En datatype viser hvilken type informasjon det er snakk om. Eksempler:

* Int (tall)
* string (tekst)
* date (dato)

**Debugging**

Debugging betyr å finne og rette feil i koden. Man tester og undersøker hva som går galt, og fikser det.

**Relasjonsdatabase**

En relasjonsdatabase lagrer data i tabeller som henger sammen. Eksempel: Én tabell for brukere og én for treningsøkter.

**Primærnøkkel**

En primærnøkkel er et unikt ID-nummer for hver rad i en tabell. Eksempel: bruker\_id.

**Fremmednøkkel**

En fremmednøkkel peker til en primærnøkkel i en annen tabell. Den brukes for å koble sammen informasjon. Eksempel: gruppe\_id i treningsøkter peker på gruppen den hører til.

# Oppgave 2: Databasemodell

## Forutsetninger

* Brukere kan være medlem i flere grupper.
* Én gruppe kan ha flere brukere.
* Én treningsøkt loggføres av én bruker i én gruppe.
* Bruker må være over 16 år (valideres i frontend/backend).

## Tabeller og attributter

1. **Bruker**
   * bruker\_id (int, primærnøkkel)
   * fornavn (varchar)
   * etternavn (varchar)
   * alder (int)
   * epost (varchar)
   * passord\_hash (varchar)
2. **Gruppe**
   * gruppe\_id (int, primærnøkkel)
   * navn (varchar)
   * beskrivelse (text)
   * gruppenøkkel (varchar)
3. **BrukerGruppe** (mellomtabell for mange-til-mange)
   * bruker\_id (int, fremmednøkkel)
   * gruppe\_id (int, fremmednøkkel)
4. **Treningsøkt**
   * økt\_id (int, primærnøkkel)
   * gruppe\_id (int, fremmednøkkel)
   * bruker\_id (int, fremmednøkkel)
   * type (varchar)
   * varighet\_minutter (int)
   * dato (date)
   * beskrivelse (text)

## Datamodellen

A screenshot of a computer

AI-generated content may be incorrect.

# Oppgave 3: Hente og vise data

For å hente og vise data fra databasen i treningsappen har jeg laget to forskjellige endepunkt (ruter) i backend. Disse er utviklet i Node.js med rammeverket Express, og bruker PostgreSQL som database.

* GET /treningsapp – viser brukere, grupper og treningsøkter i HTML-format
* GET /database-json – returnerer rå databaseinnhold som JSON

## 1. Visning av data i HTML-format

Endepunktet /treningsapp henter og viser informasjon om brukere, hvilke grupper de er med i, og eventuelle treningsøkter de har registrert i disse gruppene.

### SQL-spørring

Følgende SQL-spørring brukes for å hente ut nødvendige data fra flere tabeller i databasen:

|  |
| --- |
| SELECT  b.bruker\_id,  b.fornavn,  b.etternavn,  b.alder,  b.epost,  g.navn AS gruppenavn,  t.type AS okt\_type,  t.varighet\_minutter,  t.dato  FROM bruker b  LEFT JOIN brukergruppe bg ON b.bruker\_id = bg.bruker\_id  LEFT JOIN gruppe g ON bg.gruppe\_id = g.gruppe\_id  LEFT JOIN treningsokt t ON b.bruker\_id = t.bruker\_id AND g.gruppe\_id = t.gruppe\_id  ORDER BY b.bruker\_id, t.dato DESC; |

### Forklaring av SQL-begreper

| **Del** | **Forklaring** |
| --- | --- |
| AS | Brukes for å gi en kolonne et nytt navn midlertidig. For eksempel: g.navn AS gruppenavn betyr at kolonnen navn kalles gruppenavn i resultatet. |
| LEFT JOIN | Kobler sammen to tabeller, og viser **alle** rader fra tabellen til venstre – selv om det ikke finnes match i tabellen til høyre. Brukes når man vil vise f.eks. alle brukere, også de som ikke er med i en gruppe eller ikke har registrert trening. |
| ON | Forteller hvordan tabellene skal kobles. F.eks. ON b.bruker\_id = bg.bruker\_id betyr at vi kobler brukeren med brukergruppe der ID-ene er like. |
| ORDER BY | Sorterer resultatene. Her sorteres det først etter bruker\_id, og så etter dato for treningsøkten. |
| DESC | Sorterer i **synkende** rekkefølge – f.eks. nyeste dato først eller høyest tall først. Brukes ofte for å vise det nyeste først. |

### Behandling og visning

Resultatet fra databasen struktureres i et JavaScript-objekt. Deretter genereres HTML-dokumentet dynamisk, slik at det vises pent i nettleseren. For hver bruker vises:

* Navn, alder og e-post
* Hvilke grupper brukeren er med i
* Eventuelle treningsøkter (type, varighet og dato)

Eksempel på HTML-utskrift i koden:

|  |
| --- |
| <h2>${bruker.navn}</h2>  <p><strong>Alder:</strong> ${bruker.alder}</p>  <p><strong>E-post:</strong> ${bruker.epost}</p> |

A screenshot of a computer

AI-generated content may be incorrect.

### Bruk av Docker

For å starte databasen på en enkel måte brukte jeg **Docker**. Det gjør at man slipper å installere PostgreSQL direkte på PC-en – i stedet kjører databasen i en "container" som er lett å starte og stoppe.

Jeg brukte denne kommandoen:

|  |
| --- |
| docker run --rm --name my-postgres -p 5432:5432 -e POSTGRES\_PASSWORD=hei123 postgres |

Dette starter en midlertidig PostgreSQL-database som kjører på port 5432. Passordet for databasen settes til hei123, og databasen slettes automatisk når jeg stopper den (på grunn av --rm).

Dette gjorde det mye enklere å teste appen, og jeg slapp å bekymre meg for installasjon eller konfigurasjon på maskinen min

## 2. Rå data i JSON-format

Endepunktet /database-json returnerer all data fra fire tabeller som ren JSON. Dette brukes primært til testing, utvikling og feilsøking.

### Eksempel på kode

|  |
| --- |
| const brukere = await pool.query('SELECT \* FROM bruker');  const grupper = await pool.query('SELECT \* FROM gruppe');  const brukergrupper = await pool.query('SELECT \* FROM brukergruppe');  const treningsokter = await pool.query('SELECT \* FROM treningsokt');  res.json({  brukere: brukere.rows,  grupper: grupper.rows,  brukergrupper: brukergrupper.rows,  treningsokter: treningsokter.rows  }); |

A computer screen with white text

AI-generated content may be incorrect.**A screen shot of a computer code

AI-generated content may be incorrect.A computer screen shot of white text

AI-generated content may be incorrect.**Når man besøker http://localhost:3000/database-json, vil man få hele databasen presentert i strukturert JSON-format.

A computer screen with white text

AI-generated content may be incorrect.

## Oppsummering

Ved hjelp av Express og PostgreSQL har jeg laget to metoder for å hente og vise data:

* HTML-visning for sluttbrukere
* JSON-visning for utvikling og testing

# Oppgave 4: Versjonskontroll

Når flere jobber sammen på treningsappen, er det viktig at alle kan jobbe trygt uten å ødelegge andres arbeid. Et versjonskontrollsystem, som Git, hjelper med dette. Det gjør at flere kan jobbe samtidig på koden uten problemer.

Systemet lagrer alle endringer, så man kan alltid se hvem som har gjort hva. Hvis noe blir feil, kan man enkelt gå tilbake til en tidligere versjon som fungerte. For eksempel, hvis en ny funksjon gjør at appen slutter å virke, kan man rette det ved å bruke en tidligere versjon.

En viktig del av versjonskontroll er bruk av grener (branches). Grenene gjør det mulig for utviklere å lage egne kopier av koden for å prøve ut nye ideer eller funksjoner uten å påvirke hovedkoden (som ofte kalles “main” eller “master”). Når funksjonen er ferdig og testet, kan endringene slås sammen (merges) tilbake til hovedgrenen. Dette gjør at arbeidet kan skje parallelt og uten konflikt.

Versjonskontroll gjør også at arbeidet alltid er lagret trygt, så man mister ingenting. På denne måten blir det lettere å samarbeide, utvikle og fikse appen.