



Fagprøve – David Hasselø

Faglig dokumentasjon

Sammendrag

Dette dokumentet beskriver planlegging, gjennomføring og faglig vurdering av mitt fagprøveprosjekt: en webapplikasjon for interne turkonkurranser i Omega 365. Prosjektet dekker hele prosessen fra tolkning av oppgaven, gjennom valg av teknologier og utvikling, til testing og ferdigstillelse.

David Hasselø

david.hasselo@omega365.com

Innhold

Faglig dokumentasjon	2
Systemoversikt	2
Arkitektur	2
Database	4
Nøkkelfunksjoner	5
API-dokumentasjon	8
Teknologivalg og begrunnelse	9
Sikkerhet	9
Testing	9
Oppsett for utvikling/produksjon	10

Fagprøve – David Hasselø

Faglig dokumentasjon

Systemoversikt

StikkUt er en webapplikasjon for turregistrering, konkurranser og lotteritrekning.

Løsningen er utviklet som en fullstack-applikasjon med Node.js/Express på backend,

PostgreSQL som database og et responsivt frontend-grensesnitt bygget med HTML, CSS (Bootstrap + egendefinerte stiler) og JavaScript.

Arkitektur

Hvordan henger backend, database og frontend sammen?

Legg gjerne ved et arkitekturdiagram (eller bruk drawSQL-skjermbildet)

1. Klient (Frontend)

Frontend er bygget med HTML, CSS og JavaScript, og bruker Bootstrap for styling. Hver hovedfunksjon har sin egen undermappe og egne filer, for eksempel:

- /public/tours/ for turregistrering
- /public/competitions/ for konkurranser
- /public/lottery/ for loddtrekning
- /public/leaderboard/ for resultattabell
- /public/login/ og /public/register/ for autentisering

Frontend kommuniserer med backend via REST API-kall, hovedsakelig gjennom asynkrone fetch-kall. For å sikre gjenbruk og konsistens benyttes felles hjelpefunksjoner

fra `common.js`, som blant annet håndterer autentisering, feilhåndtering, visning av meldinger, modaler og tabell-rendering.

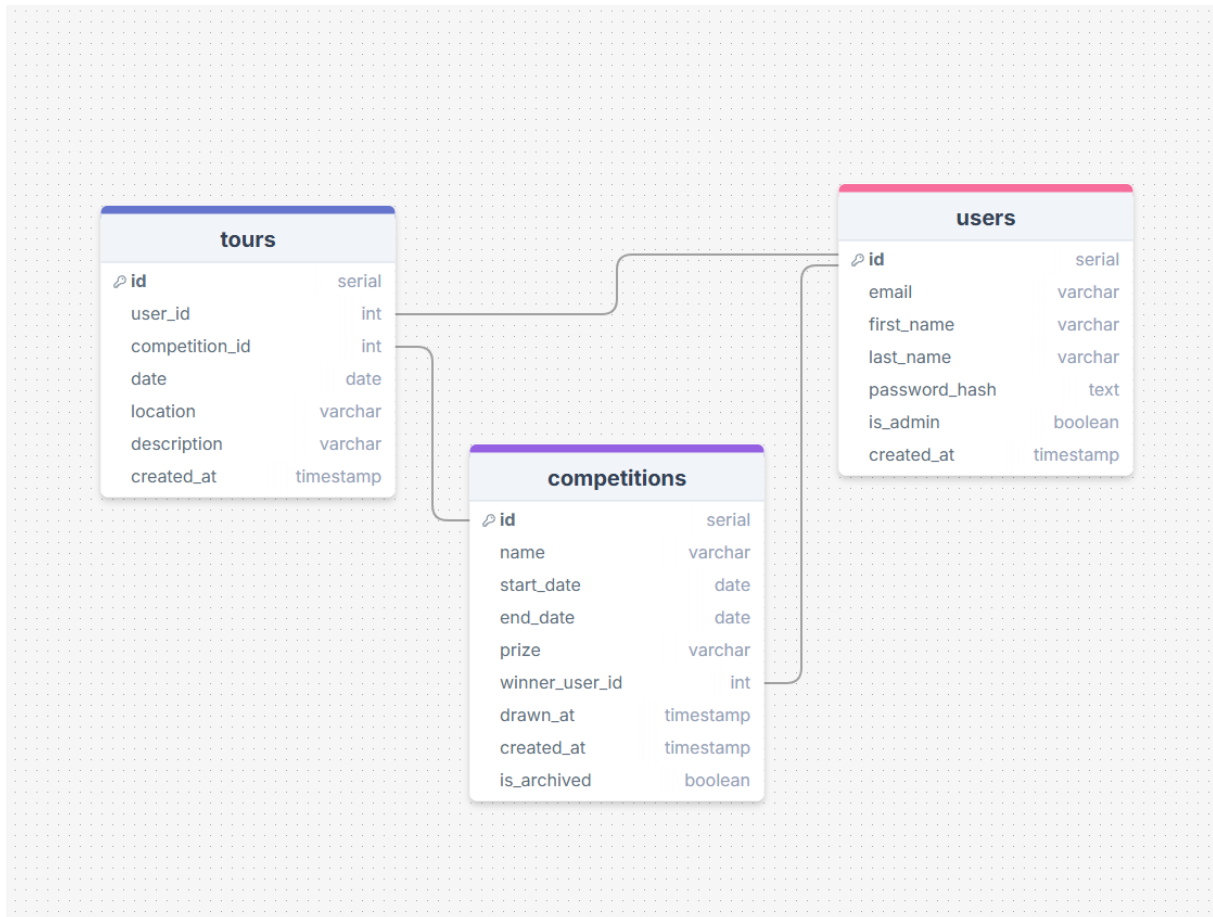
2. Server (Backend)

Backend er en Node.js-applikasjon med Express, strukturert med egne ruter og kontrollere for hver funksjonell del. Alle API-endepunkter ligger under `/api/`, og det er egne ruter for autentisering, turer, konkurranser, leaderboard og lotteri.

Backend håndterer også all logikk for autorisasjon, validering og kommunikasjon med databasen. Tilkoblingen til PostgreSQL-databasen håndteres av en egen modul, og alle sensitive operasjoner (som å opprette konkurranser eller trekke vinner) krever at brukeren har admin-rettigheter, noe som sjekkes på serversiden.

Database

Databasen er en relasjonsdatabase (PostgreSQL) som består av tre tabeller: users, competitions og tours. Disse tabellene er koblet sammen med foreign keys for å sikre dataintegritet og støtte applikasjonens funksjonalitet. For eksempel er hver tur knyttet til en bruker, og kan valgfritt knyttes til en konkurranse. Når en konkurranse avsluttes, lagres vinneren i konkurransetabellen.



(Diagrammet er designet i DrawSQL)

Nøkkelfunksjoner

Brukerregistrering og innlogging

Registrering:

Brukere kan registrere seg med e-post, navn og passord via skjema i register.html. Data sendes til /api/auth/register (se auth.controller.js) og passord hashes med bcrypt før lagring i tabellen for brukere.

Innlogging:

Brukere kan logge seg inn med e-post og passord via skjema i login.html. Data sendes til /api/auth/login og sjekker e-post og passord mot databasen. Ved suksess returneres en JWT-token som lagres i localStorage, og blir brukt for autentisering mot alle API-kall.

Navigasjon og autorisasjon

Dynamisk navigasjon:

Navigasjonen for appen er lik for alle html sider utenom index.html. Funksjonen updateAuthNav() i common.js bygger menyen basert på om bruker er innlogget og evt. Admin. Den vil vise «Hei (navn)!» og brukermeny hvis innlogget, ellers «Logg inn» / «Register».

Autorisasjon:

Alle API-kall sender JWT-token i header, og backend sjekker token og rettigheter (f.eks. admin for konkurranse-endepunkter).

Turregistrering (CRUD)

Visning:

loadTours() i tours.js henter brukerens turer fra /api/tours, og bruker renderTable() fra common.js for å vise turene i tabellform.

Legge til/redigere tur:

Opprettelse og redigering av turer skjer gjennom en modal i tours.html. Funksjonen saveTour() i tours.js sender POST/PUT til /api/tours eller /api/tours/:id. Skjemaet nullstilles og ID-feltet fjernes for ny tur.

Slette tur:

Funksjonen deleteTour() bruker showDeleteConfirmation() (SweetAlert) for bekreftelse, og skjer via DELETE til /api/tours/:id.

Konkurranser (CRUD, admin)

Visning:

loadCompetitions() i competitions-admin.js henter active konkurranser og viser kun «Legg til konkurranse» for admin.

Legge til/redigere konkurranse:

Opprettelse og redigering av konkurranse skjer gjennom en modal i competitions.html. saveCompetition() sender POST/PUT til /api/competitions eller /api/competitions/:id.

Slette konkurranse:

deleteCompetition() bruker SweetAlert for bekreftelse og sender DELETE til /api/competitions/:id.

Konkurranseshistorikk:

loadHistoryCompetitions() i competitions-history.js henter tidligere konkurranser og viser dem i dropdown.

Rangering (Leaderboard)

loadLeaderboard() i leaderboard.js henter data fra /api/leaderboard og viser antall turer per bruker i aktiv konkurranse. Bruker renderTable() fra common.js for visning.

Lotteri og trekning

Visning:

loadLotteryData() i lottery.js henter deltakere, antall lodd og sjanser fra /api/lottery og viser statistikk og deltakerliste.

Trekning:

Kun admin ser «Trekking vinner»-knappen og performDraw() sender POST til /api/lottery/draw. Vinner vises og lagres i databasen.

Historikk:

loadLotteryHistory() henter tidligere trekninger og viser dem i tabell.

Universelle hjelpefunksjoner

Universelle funksjoner som er felles for mange av funksjonene i appen finner man i `common.js`, og hjelper med å optimalisere appen ved å minske duplikat kode og gjøre koden mer lesbar.

`fetchJSON()`: Wrapper for `fetch` med automatisk token-håndtering og feilmeldinger.

`showMessage()`: Viser meldinger til bruker (suksess, feil, advarsel).

`handleModal()`: Åpner og fyller ut modaler for skjemaer.

`apiCall()`: Wrapper for API-kall med loading state og feilhåndtering.

`renderTable()`: Dynamisk tabell-rendering for alle lister.

`showDeleteConfirmation()`: SweetAlert-bekreftelse før sletting.

Sikkerhet og sesjon

Token valideres på alle sider (`DOMContentLoaded` i `common.js`). Ved utløpt token logges bruker ut og sendes til login. Kun egne data kan hentes, redigeres og slettes.

API-dokumentasjon

API-et følger REST-prinsipper og tilbyr endepunkter for alle sentrale operasjoner. Her er noen eksempler:

- `POST /api/auth/register`: Registrerer ny bruker (krever fornavn, etternavn, e-post og passord).
- `POST /api/auth/login`: Logger inn bruker og returnerer JWT-token.
- `GET /api/tours`: Henter innlogget brukers turer (krever JWT).
- `POST /api/tours`: Oppretter ny tur.
- `GET /api/competitions`: Henter aktive konkurranser.

- POST /api/competitions: Oppretter konkurranse (kun admin).
- DELETE /api/competitions/:id: Sletter konkurranse (kun admin).
- POST /api/lottery/draw: Trekker vinner i aktiv konkurranse (kun admin).

Lenke til postman api dokumentasjon:

<https://documenter.getpostman.com/view/44720323/2sB2qi9dh3>

Teknologivalg og begrunnelse

Node.js og Express er valgt for backend fordi det gir et moderne, effektivt og populært miljø for å bygge REST API-er, med god støtte for asynkrone operasjoner og mange tilgjengelige pakker. PostgreSQL er valgt som database fordi det er robust, åpen kildekode og godt egnet for relasjonsdata. JWT brukes for sikker autentisering og enkel håndtering av brukerøker på tvers av frontend og backend. Docker og Docker Compose benyttes for å forenkle oppsett og distribusjon, og sikrer at utviklings- og produksjonsmiljøet er likt.

Sikkerhet

Brukerdata og passord er sikret ved at alle passord hashes med bcrypt før lagring i databasen. JWT-token brukes for autentisering, og må sendes med alle API-kall som krever innlogging. Token valideres på backend før sensitive operasjoner utføres. Admin-rettigheter og tilgangskontroll håndteres i backend, slik at kun brukere med is_admin = 1 kan opprette eller slette konkurranser og utføre trekning.

Testing

Appen er testet manuelt i nettleser av en utvikler, hvor alle hovedfunksjoner er gjennomgått. Det er lagt vekt på å teste både vanlige brukerhistorier og ulike

feilsituasjoner, som ugyldig input, utløpt sesjon og manglende rettigheter. API-et er også testet med verktøy som Postman for å sikre at alle endepunkter fungerer som forventet. Det er utarbeidet et testskjema som dokumenterer hvilke funksjoner som er testet og eventuelle funn.

Oppsett for utvikling/produksjon

Nødvendige verktøy:

1. Docker og Docker Compose

For enkel oppstart av både database (PostgreSQL) og appen i utviklingsmiljø.

Last ned fra: <https://www.docker.com/products/docker-desktop>

2. Git

For å klonere repoet.

Last ned fra: <https://git-scm.com/>

3. pgAdmin (valgfritt)

For å administrere og inspisere PostgreSQL-databasen kan man benytte seg av nettleser versjonen som vises på <http://localhost:8080> når man kjører appen.

ellers kan man laste ned skrivebords programmet fra: <https://www.pgadmin.org/>

Oppsett og kjøring

1. Klon repoet:

```
«git clone https://github.com/DavidHasselhoe/FagproveDavidHasseloe.git»
```

```
«cd StikkUt-app»
```

2. Start app og database med Docker Compose (Husk å kjøre Docker Desktop i bakgrunnen):

```
“docker compose up –build”
```

Dette starter:

- Node.js-appen på port **3000**

- PostgreSQL-databasen på port **5432**
- pgAdmin på port **8080** og importerer pgadmin-servers.json for å opprette database konfigurasjonen helt automatisk slik at du slipper å «logge inn»

NB: Du trenger ikke å lage en .env-fil – alle nødvendige miljøvariabler er allerede satt i docker-compose.yml. (Vanligvis ville jeg ha laget en .env.example der brukeren setter sine egne miljøvariabler, men for å holde det enkelt har jeg valgt å eksponere de i docker-compose.yml for å sikre at alt fungerer på enklest mulig måte, i tillegg er alt dette en lokal instans og er kun for ment som et utviklingsmiljø for andre utviklere.)

3. Åpne appen og databasen i nettleseren

Gå til <http://localhost:3000> for å bruke StikkUt-appen.


Gå til <http://localhost:8080> for å administrere databasen. Skriv inn passord (

4. Stopp alle tjenester

Dersom man ønsker kan man stoppe docker tjenesten ved å kjøre:

«docker compose down»

For å gi en bruker admin tilgang kan du kjøre scriptet under:

1. Åpne pgAdmin på <http://localhost:8080>
2. Logg inn og koble til databasen ShtikkUtApp
3. Høyreklikk på databasen → Query Tool
4. Lim inn scriptet, bytt ut e-posten med din egen, og trykk "Execute" ()

```
UPDATE users
```

```
SET is_admin = TRUE
```

```
WHERE email = 'test@example.com';
```