Tværfagligt projekt

Event finder – Procesrapport

Alexander Jensby Thomsen
16-01-2025

Indholdsfortegnelse

Indledning

Dette projekt har til formål at udvikle en Progressive Web App (PWA), der giver brugerne mulighed for at finde og få overblik over begivenheder i deres nærområde. Appen kombinerer fordelene fra både traditionelle webapplikationer og mobile apps, hvilket sikrer hurtig adgang, offline-funktionalitet og en brugervenlig oplevelse på tværs af enheder. Projektet fokuserer på at skabe en simpel og intuitiv platform, hvor brugerne kan søge efter begivenheder, se detaljer og filtrere efter geografisk afstand, så de kun ser relevante resultater tæt på deres placering.

Opgaven vil beskrive, hvordan applikationen er opbygget, hvordan den henter og viser data, samt hvordan den fungerer både online og offline. Derudover gennemgås centrale dele af serverdelen, herunder API-håndtering og databaseopsætning.

Jeg har valgt at afgrænse opgaven til at fokusere på at opfylde kravene for at kunne klassificeres som en PWA samt sikre god kode og en velstruktureret backend-arkitektur. Jeg vil ikke gå i dybden med teoretiske begreber som polymorfi eller klassediagrammer. I stedet forklares og vises den konkrete kode og arkitektur.

Desuden er appen målrettet danske brugere, da den tilhørende database udelukkende indeholder data fra Danmark. Der er derfor ikke taget højde for flersprogede versioner eller tilgængelighedsfeatures som oplæsning, forstørrelse eller tekstmarkering.

Tidsplan

Den oprindelige tidsplan blev fulgt i forbindelse med projektets indledende faser, men der blev ikke lagt tid nok af til at håndtere eventuelle nye elementer i forbindelse med udvikling af selve appen. Her inklusiv deployment og netværkshåndtering. Dette resulterede ud i løbende tilpasninger for at sikre at projektet overholdt målet om at udvikle en PWA. De løbende tilpasninger kom i forbindelse med nye problemstillinger som skulle være med til at opnå de bedste resultater.

Det vil sige at dem der står "ikke startet" er en reference til at disse blev ikke påbegyndt, grundet den manglede tid og afgørelse om at fokusere på andre vigtigere områder i projektet.

Tidsplan	Blokeret af	Blokering	Dato	Status	Tildel
Ikke-navngivet				Ikke startet	
Rapport skrivning			@16. januar 2025 → 8. marts 2025	I gang	
Begyndelse på H5			@13. januar 2025 → 17. januar 2025	Færdig	
Casebeskrivelse og problemformulering			@16. januar 2025 → 19. januar 2025	Færdig	
App II og valg af sprog			@20. januar 2025 → 22. januar 2025	Færdig	
Kravspec og tidsplan			@23. januar 2025 → 28. januar 2025	Færdig	
ER diagram for database			@28. januar 2025	Færdig	
Klasse diagram			@28. januar 2025 → 29. januar 2025	Færdig	
Lav databasen			@30. januar 2025 → 2. februar 2025	Færdig	
Lav route decorators			@30. januar 2025 → 10. februar 2025	Færdig	
Test API kald til databasen			@3. februar 2025 → 11. februar 2025	Ikke startet	
Udvikle frontend for liste af events		Ekstra tid til frontend	@10. februar 2025 → 14. februar 2025	Færdig	
Ekstra tid til frontend	Udvikle frontend for liste af events		@15. februar 2025 → 16. februar 2025	Ikke startet	
Intergrate API kaldene til UI'en			@8. februar 2025 → 28. februar 2025	Færdig	
Ekstra tid til tidligere opgaver			@1. marts 2025 → 4. marts 2025	Ikke startet	
Hosting			@26. februar 2025 → 6. marts 2025	Færdig	
Ydeligere tid, fokus på tests			@5. marts 2025 → 6. marts 2025	Ikke startet	
Fokus på rapport			@3. marts 2025 → 7. marts 2025	Ikke startet	
Start på udvikling af unit test			@24. februar 2025 → 28. februar 2025	Ikke startet	

Figur 1 Tidsplanen som der blev fuglt.

Teknologivalg og begrundelse

Dette afsnit er opdelt i fire dele. Første del begrunder valget af teknologi til serverudvikling og database. Anden del beskriver teknologivalget til appens frontend, herunder hvorfor PWA er valgt. Tredje del samler trådene og forklarer, hvordan de valgte teknologier i frontend og backend arbejder sammen for at skabe en velfungerende helhed. Fjerde del gennemgår de overvejelser, der blev gjort i forbindelse med valg af teknologi, samt hvilke alternativer der var overvejet undervejs

Afsnit 1 – Serverside

Serveren har til formål at sikre sikker kommunikation med databasen, hente de nyeste events fra eksterne kilder og håndtere API-kald fra klienterne. Den fungerer som et mellemled mellem frontend-applikationen og databasen, hvilket mulig gør effektiv datahåndtering og sikrer, at brugeren altid får de opdaterede og relevante oplysninger.

Serveren er kodet i Python, hvilket giver en minimalistisk tilgang til serverdrift, da alt er organiseret i relevante mapper. Python er også et fremragende valg til udvikling af API'er, især ved brug af Flask-biblioteket. Flask er en letvægts web framework, der gør det nemt at opbygge RESTful API'er. Flask har et stort og aktivt fællesskab, som gør konfiguration af ruter og integration med værktøjer som Swagger både nem og effektiv. Flask understøtter skalerbarhed og gør det lettere at håndtere API-kald, hvilket letter den videre udvikling og vedligeholdelse af applikationen.

Databasen er MSSQL, valgt på grund af min erfaring med at udvikle SQL-forespørgsler samt MSSQL's evne til at tilbyde en skalerbar og højtydende løsning. MSSQL understøtter brug af stored procedures og views, hvilket betyder, at jeg kan udvikle og skalere databasen uden at skulle skrive komplekse og tunge forespørgsler. Med værktøjet SQL Server Management Studio (SSMS) kan databasen nemt administreres og optimeres. Foe en oversigt over databasen, se kilde 1.

Afsnit 2 – APP'en

App'en er udviklet som en PWA, Progressive Web App. Dette skyldes disse fire følgende årsager:

1. Brugeroplevelse
    a. PWA tilbyder en brugeroplevelse som en normal app ville. Dette vil hjælpe brugeren der ønsker en app liggende løsning til mobile enheder uden at skulle hente den igennem ens app store.
2. Offline funktionalitet
    a. Med Service Workers tilbyder PWA en funktionalitet som gør det muligt at app'en er i brug når brugen ikke har adgang til internet. Dette er hovedsaligt vigtigt hvis brugeren ønsker at tilgå events som kan være gemt i enhedens cache.
3. Cross-platform
    a. Om brugeren bruger en iOS eller Android enhed, er lige meget, da PWA tilbyder at blive installeret på alle enheder. Dette er omkostningfrit, da mulige ændringer kun behøves at gøre et sted, i stedet for de enkelte filer, som hvis det var native.
4. Omkostningseffektiv
    a. Med en PWA behøver man ikke gøre udvikling mere kompleks end det det er, da man ikke behøver separate visioner af appen til iOS og Android.

Mit fokus ved valg af frontend-teknologi var at finde en løsning, der var modulær og fleksibel, så udviklingen kunne opdeles i mindre, håndterbare komponenter. Derfor faldt valget på Vue.js, som netop gør det muligt at bygge interaktive brugergrænseflader med en komponentbaseret arkitektur.

Vue.js er som udgangspunkt ikke en PWA, men ved hjælp af Vue CLI og relevante plugins kan projektet nemt udvides med PWA-funktionalitet. Når projektet bygges, bliver der automatisk genereret en manifest-fil og en service worker, som placeres i dist-mappen, hvilket sikrer, at appen lever op til PWA-kravene uden manuel opsætning.

For at sikre en fleksibel og skalerbar frontend-struktur er der valgt et framework, der understøtter simpel konfiguration af routing. Dette gør det muligt at styre, hvilke komponenter og sider der vises, baseret på forskellige kriterier som fx brugerens authentication-status eller validering af data.

Denne fleksibilitet gør det nemt at udvide appen med flere funktioner i fremtiden og sikrer, at brugeren altid præsenteres for de relevante sider, uden at skulle navigere manuelt mellem forskellige URL'er.

Et af kravene for at en applikation kan klassificeres som en PWA, er at den skal tilgås via en sikker HTTPS-forbindelse. For at opfylde dette krav er applikationen deployet på Googles Firebase-platform. Firebase er valgt som hosting-løsning, da platformen tilbyder gratis hosting til mindre projekter som dette, hvilket gør den ideel til udvikling og test. Derudover tilbyder Firebase en simpel og veldokumenteret CLI, som gør deployment-processen hurtig og nem. Dette gør det muligt at opdatere og udrulle nye versioner af applikationen med få kommandoer.

Afsnit 3 – Server og appen i helhed

Applikationen er opbygget i en klassisk klient-server-arkitektur, hvor frontend, server og database har tydeligt adskilte roller, men arbejder tæt sammen for at sikre en effektiv og brugervenlig oplevelse. Læs mere under afsnittet "Arkitektur og

Det betyder at når brugeren åbner appen, hentes de nyeste events ikke direkte fra databasen, men via serverens API som agerer mellemled. Denne tilgang sikrer flere ting:

- Sikekrhed
    - Frontend har ingen direkte adgang til serveren, hvilket minimerer risikoen for angreb imod databasen, og derved øger sikkerheden for data der opbevares.
- Validering og databehandling
    - Serveren validerer indkommende forespørgsler og sørger for, at kun relevante data sendes tilbage til klienten. Dette giver en kompleks databehandling og sker i serveren i stedet på klienten.

Frontend og backend er tæt integreret via RESTful API'er, der sikrer, at appen får de nyeste data fra serveren og viser dem korrekt i brugergrænsefladen. Flask API'et på serveren håndterer anmodninger fra frontend'en, mens databasen sørger for, at dataene altid er opdaterede og relevante. Desuden sikrer PWA-funktionaliteten, at appen fungerer optimalt både online og offline.

For at sikre en overskuelig og vedligeholdsesvenlig struktur i backend-applikationen er der valgt en lagdelt arkitektur kombineret med repositry pattern. Denne struktur opdeler backend i adskilte lag med hver deres ansvar, hvilket gør der lettere at vedligeholde, udvide og teste systemet.

Det vil betyde at for backend-applikationen er den del op i følgende:

- Router-lag
  - Håndterer de indhående API-kald og sørger for routing
- Service-Lag
  - Dette håndterer forretningslogikken.
- Repository-lag
  - Håndtere al kommunikation med databasen

Denne opdeling sikrer, at koden er organiseret, og at hver del af systemet har én tydelig rolle, hvilket understøtter en god arkitektur og gør fremtidig videreudvikling mere smidig. Se kilde 2, for en visualisering.

Overvejelser over andre teknologier

Som nævnt tidligere var Python valgt, grundet minimalistisk tilgang til serverdrift, da alt er organiseret i relevante mapper. En anden teknologi som også var overvejet var ASP.NET kodet i C#. ASP.NET og Entity Framework (EF), giver mulighed for at udvikle API'erne i et .NET miljø, men er ikke så minimalistisk, og kræver mere for at skalere ens API'er.

Man kan også med ASP.NET udvikle ens database med ORM (Object-Relational Mapping), men ville ikke passe ind i min arkitektur, da der er lagt meget vægt på hvor ansvarsområdet skal være. Og alt database udvikling skal være i databasen. Også i Python kan man lave ORM, med SQLAlchemy er det muligt at lave deklarative og dynamiske modeller, men også her passer det ikke ind med arkitekturen.

Til app'en var React Natvie også en vigtig kandidat, med et stort fællesskab og mange plugins. Men den manglende erfaring med JavaScript og React i helhed, scorede React Native ikke højt i point til teknologivalget.

Konklusion

Produktet

App'en kan udstille relevante events baseret på den ønskede afstand fra brugerens lokation. Dog er funktionaliteter som filtrering på interesser og dato ikke implementeret. Dette skyldes yderligere krav givet fra lærerne:

1. Brugere
    a. Man skulle kunne opretter brugere
    b. Man skal kunne logge ind med den brugere
    c. Brugeren skal kunne tilføje begivenheder til ens egen liste
    d. Der skal være en funktionalitet til at brugeren skal have en liste over begivenheder som er fra personen selv, har tilføjet til.

Denne simple, men alligevel store opgave har involveret at der skulle udvides på databasen, udstilles flere API'er og arbejde med cookies i form af authentication.

Min manglede erfaring med at udvikle frontend applikationer i Vue.js, og at det skal være PWA, har gjort at tiden som skulle have været brugt på filtrering af dato og interesser, ikke er blevet implementeret, da den tid var givet til en stabil funktionalitet om at kunne oprette og logge ind med en bruger.

Det kan konkluderes at selvom den oprindelige problemformulering ikke er blevet opfyldt helt korrekt, er der stadig hovedelementer opfyldt, og det er at man skal kunne få vist begivenheder indenfor en angivet afstand.

Rapporten

Jeg har tilladt mig selv at tilføje denne del til konklusionen, i det at der har været meget uenighed i udviklingsperioden, om hvordan de to rapporter skulle strukturens, skrives og hvilke kriterier der er. Rapporten startede med at blive skrevet efter egen erfaring med at skrive rapporter, men måtte indse her midt i forløbet, at underviseren ønskede noget andet. Jeg tillod mig at se på givet materiale for hvilke komponenter rapporterne skulle indeholde.

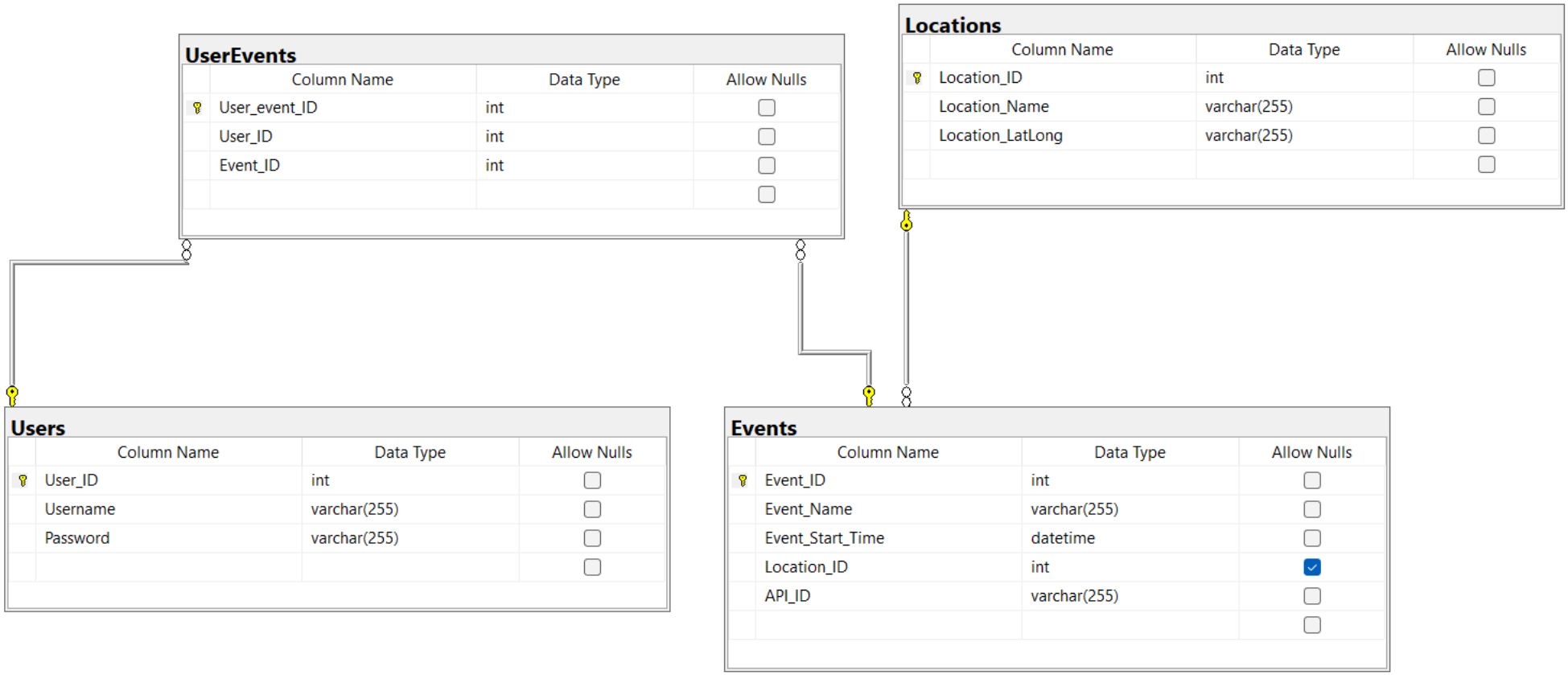
Dette resulterede i at vores afleveringsfrist for min problemformulering endte med at være 1/3 af en side, og indeholde også hvordan perspektivering skulle være. Jeg måtte erkende at både perspektivering, kun skulle være Maks 2 linjer, og der ingen perspektivering skulle være i rapporten.

De to rapporter er skrevet af en blanding af hvilke elementer jeg personligt finder relevant og hvilke elementer der ønskes. Dertil

Kilder

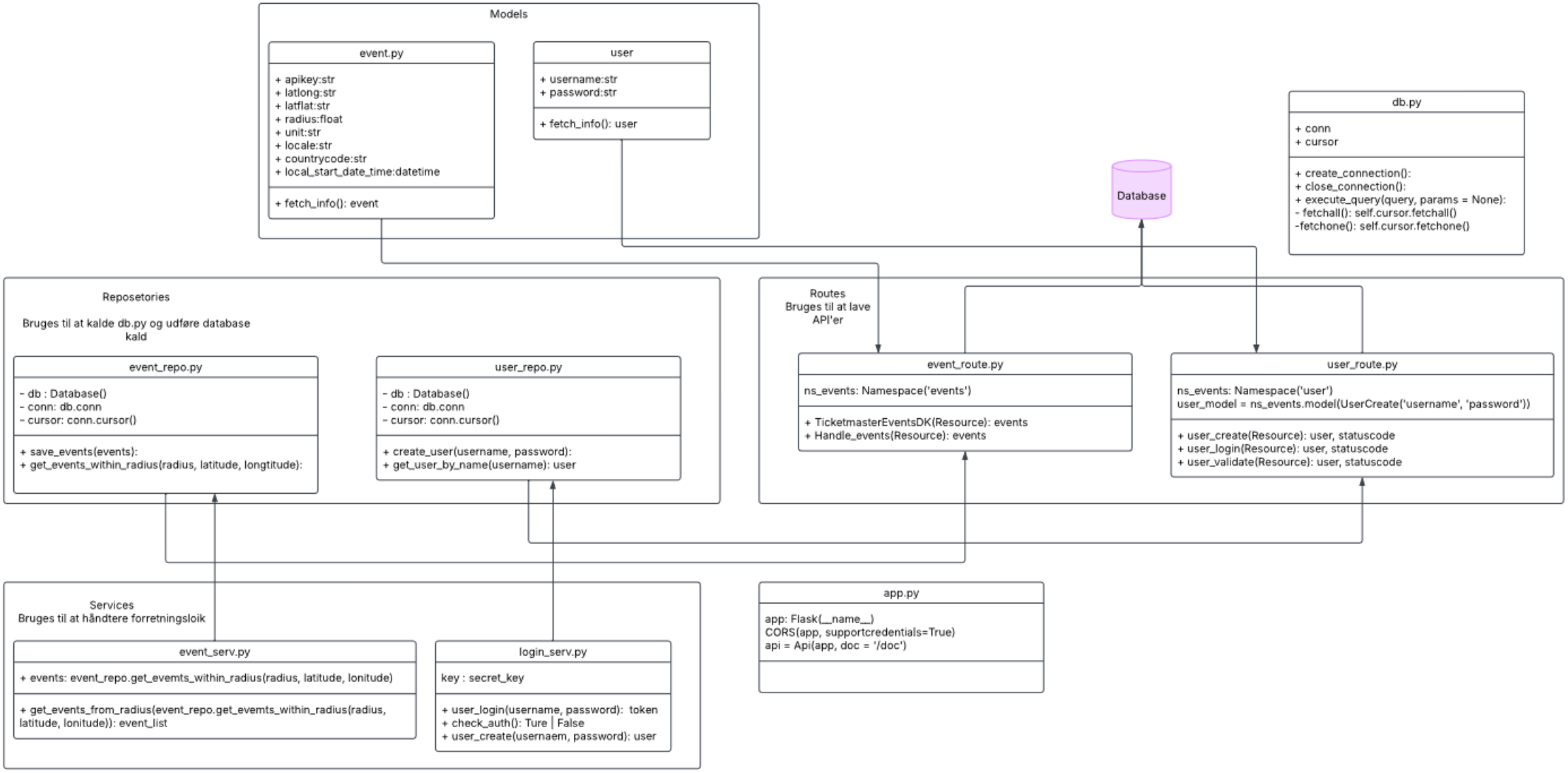
Databasediagram

Database diagram, oversigt over databasen



Kilde 2: Systemdiagram

Systemdiagram over hvordan backend kalder hinanden.



Bibliografi

Anbefalinger til identifikation og bedømmelse flows. (16. 10 2024). Hentet fra Microsoft: https://learn.microsoft.com/da-dk/power-platform/well-architected/reliability/identify-flows

Hjerrild, M. (2022). *laeremiddel.dk*. Hentet fra laeremiddel.dk: https://laeremiddel.dk/viden-og-vaerktoejer/datakodning-analyse-og-fortolkning-af-empiriske-data/analysetilgange-til-forskellige-empirityper/analyse-af-cases/

skriveopgave.kk.dk. (u.d.). Hentet fra skriveopgave: https://skrivopgave.kk.dk/begynd-her/emnevalg/problemformulering