Dokumentasjon for Kalenderprosjektet

Kalenderapplikasjonen er utviklet for å hjelpe brukere med å planlegge, organisere og holde oversikt over hendelser. Prosjektet bruker et backend-system bygget med Flask, en MariaDB-database for lagring, og en frontend i JavaScript. Formålet er å gi brukerne en enkel og effektiv måte å behandle daglige oppgaver og arrangementer.

Funksjonalitet (Per nå):

Logg inn: Brukere kan logge inn ved hjelp av en bestemt e-post og et passord.

Legg til hendelser: Brukeren kan opprette hendelser med dato, tid, sted og beskrivelse.

Vis kalender: Hendelser vises på en oversiktlig kalender som brukeren kan navigere gjennom.

Opprett ny bruker: Brukere kan registrere seg og opprette egne kontoer.

Planlagte funksjoner:

E-postpåminnelser: Varsler brukerne om kommende hendelser.

Redigering og sletting av hendelser: Gir brukeren mer kontroll over opprettede hendelser.

Hva blir brukt i prosjektet?

Backend:

Flask (Python): Håndterer API-forespørsler og databaselogikk.

Database:

«MariaDB» brukes til å lagre bruker- og hendelsesdata.

Frontend:

«JavaScript» for å gjennomføre kalendergrensesnittet.

«HTML/CSS» er det grunnleggende grensesnittdesign.

Verktøy:

«mysql-connector» for databasekommunikasjon.

«bcrypt» for passordhashing.

Databasestruktur

Tabell for bruker

brukerID (Primary Key): Identifikator for brukeren.

e_post: Brukerens e-postadresse.

passord: Passord for innlogging

fornavn og etternavn: Valgfritt for ekstra brukerdata.

Tabell for events

eventID (Primary Key): Identifikator for hver hendelse.

dato: Dato for hendelsen.

klokkeslett: Tidspunkt for hendelsen.

beskrivelse: En kort forklaring av hendelsen.

navn_prosjektet: Navn eller sted for hendelsen.

brukerID (Foreign Key): Knyttet til brukeren som opprettet hendelsen.

Utviklingslogg

Planlegging

Jeg bestemte meg for å jobbe videre med en timeplan-applikasjon som jeg hadde begynt å utvikle i sommer. Hovedideen var å lage en kalender som gir brukerne mulighet til å navigere mellom ulike måneder i året og som automatisk oppdaterer dagens dato daglig. Applikasjonen er spesielt designet for personer med travle hverdager som er avhengige av kalendere for å organisere seg. Målet var å lage et brukervennlig format som ikke bare gjør det enkelt å legge til arrangementer, men som også gir påminnelser om disse arrangementene og viser en oversikt over hvor mange arrangementer de har i løpet av dagen eller måneden.

MVP-videreutvikling

Jeg startet prosjektet med å planlegge strukturen og lage en grunnleggende oversikt over funksjonene. Til back-end valgte jeg Flask, MariaDB til databasen, og JavaScript til front-end. Jeg lagde et utkast til hvordan kalenderen skulle fungere, inkludert innlogging, legge til og vise hendelser.

For databasen opprettet jeg to tabeller:

- **bruker**: For å lagre informasjon om brukerne.
- **events**: For å lagre hendelser.

Jeg skrev SQL-koden for tabellene og la inn testdata for å kunne teste funksjonaliteten senere. Planen var å starte med back-end i Flask neste dag og bruke testdataene for å sikre at databasen fungerte som forventet. Underveis oppdaget jeg at currentDate i JavaScript ikke oppdaterte datoen riktig. For å finne feilen la jeg til en console.log for å sjekke om feil dato ble hentet.

Arbeid med back-end

Jeg begynte med å lage en Flask-applikasjon og koblet den til MariaDB-databasen. Først fokuserte jeg på å legge en funksjon for å legge til hendelser, inkludert en route (/add_event) som sender data fra skjemaet til databasen. Deretter lagde jeg en innloggingsfunksjon med en route (/login) som sjekker om e-post og passord stemmer med databasen.

For testing la jeg manuelt til en bruker i databasen, siden jeg ikke hadde en registreringsfunksjon ennå. Under testing opplevde jeg at Flask-applikasjonen krasjet, og jeg fikk feilmeldingen: *"Plugin could not be loaded"*. Etter undersøkelse fant jeg ut at dette skyldtes at databasen brukte autentiseringspluginen caching_sha2_password, som ikke støttes av klienten. Jeg endret autentiseringspluginen til mysql_native_password, men problemet ble ikke helt løst, så jeg planla å teste videre.

Kalender og kobling til back-end

Jeg begynte å jobbe med front-end for å forbedre kalenderens brukeropplevelse. Kalenderen viste datoer og lot brukeren bytte mellom måneder, men innloggingen fungerte fortsatt ikke, så jeg fikk ikke testet endringene i JavaScript-koden.

Jeg testet databasens funksjonalitet ved å bruke en telefonkatalog som eksempel, hvor jeg la til og viste kontakter. Dette fungerte som det skulle, så problemet lå mest sannsynlig i prosjektkoden. Til slutt oppdaget jeg at jeg hadde brukt feil bibliotek og trengte mysql-connector i stedet for mysql-connector-python.

Jeg oppdaget også at kalenderen ikke oppdaterte seg automatisk etter at nye hendelser ble lagt til. For å løse dette la jeg til en API-forespørsel som henter oppdatert data fra databasen etter lagring. Jeg debugget også JavaScript-logikken for navigasjon mellom måneder, men rakk ikke å fullføre.

Feilsøking og forbedringer

Under testing opplevde jeg at flere funksjoner prøvde å kjøre samtidig, noe som førte til at jeg ikke kunne logge inn. Jeg vurderte å bruke en "DOM contentload"-løsning, men fant en enklere løsning ved å dele opp JavaScript-koden i separate filer. Dette fjernet konfliktene mellom funksjonene.

Jeg begynte også å legge til sikkerhetsforbedringer, som å bruke bcrypt for å kryptere passordene. Målet var å sikre at innloggingen og kalenderens databasetilkobling fungerte på alle plattformer.

Oppsett av webserver og port forwarding

Jeg satte opp en webserver ved hjelp av Nginx og konfigurerte port forwarding for å gjøre serveren tilgjengelig utenfor hjemmenettverket. Jeg installerte Nginx og satte opp en "reverse proxy" for å håndtere både statiske filer (HTML, CSS, JavaScript) og dynamiske forespørsler fra back-end. Det var utfordrende å få Nginx til å sende riktig informasjon videre til Flask, fordi git hub av en eller annen grunn ikke klarte å klone prosjektet. Når jeg prøvde å klone prosjektet på min raspberry pi, klonet det veldig sakte, og deretter mislyktes tilkoblingen. Jeg prøvde å se om problemet var at det var på min raspberry pi og jeg prøvde å klone prosjektet fra git hub på datamaskinen min, men det klarte fortsatt ikke å klone. Så jeg lastet endelig ned zip-mappen til prosjektet og hadde ingen annen mulighet enn å lage et nytt repo og koble det til prosjektet mitt som jeg hadde lastet ned. Deretter resten av oppsetningen til webserveren gikk helt ok, og det fungerte etter noen justeringer.

Jeg satte opp portene 80 og 443 for HTTP og HTTPS for å gjøre applikasjonen tilgjengelig via internett, port 22 for SSH for å administrere serveren, og port 3306 for å gi tilgang til MariaDB-databasen. Dette oppsettet sørger for at både front-end, back-end og database kan kommunisere sømløst.

Det siste jeg har gjort er å få teste at andre maskiner kan åpne nettstedet mitt og at andre får teste den å få tilbakemeldinger om nettsiden. I tillegg så sørget jeg for at mitt virtuelle Python-miljø, myvenv, fungerte som det skulle for å holde applikasjonen isolert og stabil og legget til gitignore for at den virtuelle miljø skal ikke være med på den git hub repo.

Om jeg kunne få mer tid ville jeg jobbe mer med å forbedre ting på tilbakemeldinger jeg fikk som for eksempel lage en logg ut route, også gjøre synlig hvilkedtaoer er det som har en event på seg.