Forside

Titelblad

# Indholdfortegnelse

# Læsevejledning

# Indledning?

# Kravspecifikation

Selve produktet omfatter et API med tilhørende database og en PWA.

## Funktionalitet

Denne tabel viser kravene til systemets forskellige dele.

|  |  |  |  |
| --- | --- | --- | --- |
| **Krav ID** | **Kategori** | **Type** | **Krav** |
| SO1 | Server | Opsætning | Database skal være opsat korrekt med tabeller til at indeholde brugere, ture, og punkter. |
| SF1 | Server | Funktionalitet | API skal kunne modtage data fra PWA og lagre denne i databasen. |
| SF2 | Server | Funktionalitet | API skal kunne sende data fra databasen til PWA. |
| SF3 | Server | Funktionalitet | API skal udregne relevante statistikker når brugeren beder om data. |
| SS1 | Server | Sikkerhed | Databasen skal opbevare sensitive brugeroplysninger sikkert. |
| PO1 | PWA | Opsætning | PWA skal kunne tilgås og downloades eksternt. |
| PF1 | PWA | Funktionalitet | PWA skal kunne registrere og logge brugere ind. |
| PF2 | PWA | Funktionalitet | PWA skal kunne sende brugeres lokation til API efter ønske. |
| PG1 | PWA | Gui | PWA skal kunne præsentere tidligere løbeture for brugere. |
| PG2 | PWA | Gui | PWA skal kunne præsentere brugbar information om gennemsnitshastigheder, m.m for brugere. |

## Testkonditioner

Denne tabel viser de testkonditioner der bestemmer om kravene er opfyldt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Opfyldt** | **Til dels opfyldt** | **Ikke opfyldt** | **Test Case** |
| SO1 |  |  |  | Database skal inspiceres grundigt i Microsoft SQL Server Management Studio efter opsætning. |
| SF1-A |  |  |  | Kontrollér via swagger og Microsoft SQL Server Management Studio at API’ets endpoints modtager og lagre data korrekt. |
| SF1-B |  |  |  | Kontrollér i mssql studio at data fra PWA forekommer i databasen som forventet. |
| SF2-A |  |  |  | Kontrollér via swagger at API’ets endpoints returnere data som forventet. |
| SF2-B |  |  |  | Kontrollér at data fra database forekommer i PWA som forventet. |
| SS1 |  |  |  | Kontrollér at brugeres passwords bliver hashet i databasen. |
| PO1 |  |  |  | Kontrollér at en uafhængig enhed kan downloade og benytte PWA’en. XXXLIGHTHOUSETESTXXX |
| PF1 |  |  |  | Kontrollér at brugere kan oprettes og logge ind og ud uden problemer. |
| PF2 |  |  |  | Kontrollér at oplysninger fra PWA om lokation kommer ordentligt igennem til databasen og med den rette frekvens. |
| PG1 |  |  |  | Kontrollér at PWA kan fremvise tidligere ture korrekt og hensigtsmæssigt. |
| PG2 |  |  |  | Kontrollér at PWA kan give nyttig information og statistik til brugeren. |

## Krav

Dette afsnit beskriver hvordan faglige krav stillet til projektet er opfyldt.

### Obligatoriske elementer

Projektet har fem faglige krav som er beskrevet nedenfor.

#### Konfigurationsstyring

Git bliver benyttet til konfigurationsstyring af projektet.

Projektets GitHub repository indeholder både begge større dele af projektet, API og PWA, samt rapporter og tidsplaner m.m.

Derudover dette er GitHub også konfigureret med firebase actions til at build og deploy PWA til firebase ved hvert push, som er et forsøg på at have en grad af continous integration inde over.

#### Sikkerhed

Der er blevet taget højde for sikkerhed i projektet ved at brugeres passwords bliver hashet i databasen ved brug af en langsom algoritme, og aldrig opbevaret i plain text.

Når en bruger er logget ind bliver de tildelt en JSON Web Token, som senere kan bruges til at krydstjekke at de er hvem de giver sig ud for, når de beder om information. Det er ikke muligt for en bruger at ændre på denne token uden at gøre den ugyldig.

Yderligere sker al udvæksling af information mellem API og PWA via https.

#### Test

Projektets API har en række tests skrevet til testing af endpoints. Dette er ikke unit testing, men nærmere black-box testing, da API’ets afhængighed af tokens gjorde dette vanskeligt, hvis ikke umuligt. Testene kalder i stedet endpoints og kontrollere at de får de korrekte responses.

API’ets tests kan ses i filen: <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/UnitTests/UnitTest1.cs>

Til PWA er en række brugertestcases vedlagt som bilag. Brugertests er brugt i stedet for autotests i dette projekt, da medvirkende udviklere har ekstensiv erfaring med autotest, og autotestscases er utroligt smarte og effektive, men tager lang tid at opbygge korrekt.

X brugertestcases er vedlagt som bilag XX

#### Database

Projektets API har en tilhørende database, sat op via Entity Framework, og code-first princippet. Denne database omfatter data i form af brugere, løbeture, og punkter.

#### Server

### Valgfri elementer

Udover de fem obligatoriske faglige områder, skal projektet omfatte et af en række valgfri områder.

#### APP Udvikling

# Information om teknologier

Dette afsnit vil omhandle info om de forskellige teknologier der er brugt til dette produkt.

## API

Projektet benytter et API der er skrevet i ASP.net. API står for Application Programming Interface, og er i al sin enkelthed et bindeled der tillader flere forskellige teknologier at kommunikere[[1]](#footnote-1).

Dette projekts API er bindeled mellem databasen og PWA’en. Dette tillader de to services at køre på separate enheder med forbindelse til internettet, frem for ellers at skulle køre på samme enhed.

### Entity Framework

Projektets API benytter Entity Framework til at kommunikere med databasen.

Entity Framwork er et såkaldt ORM framework, som står for Object Relational Mapping. Det betyder at i stedet for at arbejde med sin data i tabeller, med kolonner og rækker, kan man tilføje og tilgå data som objekter, hvilket gør dataene meget nemmere at arbejde med i koden[[2]](#footnote-2).

### JSON

Når data skal sendes mellem PWA og API sker dette i formatet JSON.

JSON står for JavaScript Object Notation, og er et dataudvekslingssprog der er nemt for mennesker at læse og forstå, hvilket gør det nemt at arbejde med. JSON er sproguafhængigt, men bruger konventioner som ligner C-Sprogfamilien, deriblandt C# som API’et er skrevet i[[3]](#footnote-3).

Det at JSON udveksler data i objekter gør det ideelt til udveksling af data med et API som er objektbaseret.

### bcrypt

Til beskyttelse af data, i form a hashing af passwords, bruger API’et bcrypt.

bcrypt er udviklet af Niels Provos and David Mazières, og har to store fordele over andre algoritmer når det kommer til password hashing. For det første er bcrypt, modsat mange andre, ikke programmeret for hastighed og ydeevne, men med sikkerhed i højsædet. Når man hasher et password, kan man vælge at hashe resultatet igen for at fordoble tiden et brute-force angreb ville skulle investere i at knække det. Dette vil en bruger i teorien ikke mærke noget til, da man kan hashe et password mange tusind gange på et enkelt sekund, men en brute-force svindler vil være tvunget til at hashe et større antal gange for hvert forsøg. Mange algoritmer gør dette, men bcrypt har som egenskab at den kan skalere sit antal af iterationer med computerens kraft, så man ikke ender med en masse gamle, sårbare passwords, og en masse nye, sikrere passwords. For det andet tager bcrypt et såkaldt salt, som bliver hashet sammen med passwordet. Dette sikrer at folk med samme password ikke vil have samme hash i databasen, så en brute-force svindler vil skulle knække hvert password separat, frem for at kunne tage en masse på samme tid, hvis han vel og mærke har været i stand til at aflæse på forhånd at en mængde brugeres hashede passwords er ens[[4]](#footnote-4).

### Azure

## Database

Projektet benytter sig af en MS-SQL database sat op via code-first princippet gennem Entity Framwork. Dette betyder at man har bedt API’et om at generere en database ud fra en beskrivelse, i modsætning til at sætte en database op først og kode den sammen med API’et manuelt. Dette springer over en mængde fejl som en bruger kunne komme til at lave i den sammenhæng, når databasen virker lige godt uanset.

MS-SQL er et relationelt databasesystem, hvilket vil sige at man kan have data der har relationer til anden data. Disse relationer kommer i to typer; en-til-mange relationer, og mange-til-mange relationer. For eksempel, i dette projekt ser vi brugere, ture, og punkter. En bruger består af information vigtig for en bruger, for eksempel brugernavn og password. En tur består af information om en given tur, for eksempel hvornår den foregik. Et punkt består af koordinator på hvor på kloden punktet befinder sig. Alle tre objekttyper har også Id’er som man kan bruge til at lave relationer mellem dem. En tur skal, for eksempel, indeholde hvilken bruger der har være ude på turen, så hver tur refererer også til et brugerId. Og punkter er ikke meget værd hvis de ikke fortæller hvilken tur de hører til, så hvert punkt refererer til et turId. Disse relationer er begge eksempler på en-til-mange relationer. En mange til mange relation kunne for eksempel være hvis man byggede systemet så flere brugere kunne deltage i en given tur, i hvilket tilfælde man ville have en mellemtabel til at indeholde relationsdataene, som begge tabeller så ville have en-til-mange-relationer til. I den forstand kan en mange-til-mange relation siges at være en snedig kombination af to en-til-mange relationer.

Den primære kommunikationsvej til en MS-SQL database er T-SQL.

### T-SQL

T-SQL bygger på SQL, Structured Query Language, som er standarten for at kommunikere med relationelle databasesystemer[[5]](#footnote-5).

SQL er deklarativt, modsat det meste andet programmering som er imperativt. Det vil sige at hvor man normalt fortæller et stykke software præcis hvad det skal udføre, kommando for kommando, vil man her i stedet beskrive det resultat man er ude efter, og så vil softwaren sørge for resten[[6]](#footnote-6).

## PWA

En PWA, eller Progressive Web App, er en nem måde at udvikle en app der kan køre cross-platform, altså som virker både på Android og iOS, og sågar også Windows. Man koder en PWA med HTML, CSS, og JavaScript, så hvis man er komfortabel med de sprog i forvejen, kan man nemt gå til det.

En PWA får på en måde det bedste fra to verdener; den er tilgængelig overalt som en traditionel web-app, og den kan tilgå en enheds komponenter son en native app kan. En PWA kan dog ikke tilgå alle de komponenter en native app kan, men funktionaliteten er stødt stigende[[7]](#footnote-7).

Dette projekts PWA er skrevet i et framework der hedder Vue.js.

### Vue.js

Vue.js er et library til at bygge reaktive webinterfaces[[8]](#footnote-8).

Vue.js er som udgangspunkt bygget på arkitekturmønstret MVVM, eller Model-view-viewmodel. Dette vil sige at interfacet (view) og logikken (model) er adskilt og snakker sammen via en viewmodel, som er en form for mellemled. Dette foregår via databinding som Vue.js har indbygget, og som gør at man kan binde ui-elementer til bestemte data, som så opdaterer af sig selv når dataene gør, uden at det behøver at være en del af logikken[[9]](#footnote-9).

Ydermere er Vue.js komponentbaseret, som vil sige at man kan definere forskellige komponenter, som for eksempel en nav- eller sidebar, og så importere og bruge dem på flere sider. Det er noget der virkelig giver mening i forhold til andre former for programmering, hvor hvis noget kan nøjes med at stå kun et sted, er der kun et sted det skal ændres hvis det skulle blive aktuelt[[10]](#footnote-10).

# Overordnet arkitektur

Produktet består af tre dele; en database, en PWA, og et API der fungere som bindeled derimellem.

BILLEDEIEIE!?!?!

Dette afsnit vil forklare vejen som data tager gennem projektet, når man benytter projektets hovedfunktion som er at optage ture, og se dem derefter.

## PWA - Afsender

Når en bruger er klar til at løbe og trykker på ’Start Løbetur’ bliver den følgende kode afviklet i PWA’en.

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

Denne kode sender besked til API’et om at den gerne vil starte en ny tur, og med som header sender den sin token, så API’et kan kontrollere at den har lov til det. Dette kald besvares med det objekt der er oprettet i databasen og ID’et gemmes i en variabel der skal bruges når der tilføjes punkter, for at vide hvilken tur de hører til.

Selve turens punkter bliver tilføjet med følgende kode, som kører i et loop i en web-worker, grundet at den så kan køre selvom brugeren låser sin telefon.

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

I denne kode tilgås enhedens GPS og data derfra bliver omdannet til et format der kan sendes til API’et sammen med ID’et på den igangværende tur. En enheds GPS afhængig af kompleksitet kan optage mange værdier, men for projektet her er den kun væsentligt at behandle latitude, longitude, og altitude.

Når brugerens tur er slut, trykker de på knappen igen, og dette terminere web-workeren så der ikke sendes flere punkter.

## API - Modtager

Når API’et modtager at der skal startes en ny tur, modtager den faktisk ikke anden data end en token.

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

Den bruger metoden GetUserId() til at læse userID ud fra den token den modtager, og bruger denne info til at finde det User objekt som turen skal tilhøre. Tider genereres serverside for at de bliver så ensartede og korrekte som muligt, frem for at bruge tiden fra en brugers enhed.

Når API’et modtger et punkt, sendes punktets data samt hvilket run det hører til.

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

Punktet tilføjes til run.points i stedet for \_context.Points fordi at systemet aldrig arbejder med points fra databasen isoleret, men altid i konteksten af hvilken run det tilhører. Derfor, ved at tilføje dem på denne måde, kan man bede om at få points med som en del af et run objekt, i stedet for at skulle lave en separat database kald for at få dem leveret efterfølgende. API’ets PointController har faktisk slet ikke et endpoint til at udlevere points, kun til at modtage dem.

I dette tilfælde bruges dens token kun til at sikre at kaldet er gyldigt. Der er ikke grund til at tilknytte et punkt en user, da punktet i forvejen er tilknyttet en tur, som allerede er tilknyttet en user.

## Database

Projektets database er struktureret som påvist på entity relation diagrammet herunder.

BILDIEELDIELDIE

Users tabellen indeholder data om brugere, som for eksempel brugernavn og password, hashed med bcrypt og aldrig opbevaret eller behandlet i plaintext udover når brugeren selv taster det ind. Tabellen har også et userId som systemet bruger men som brugeren aldrig ser.

Runs tabellen indeholder det data der er vigtigt for en tur, alt sammen faktisk genereret serverside når den bliver oprettet.

Points tabellen indeholder data vigtig for et enkelt punkt tilknyttet en tur.

Testbrugerscript!

## Api - Afsender

Når en bruger har afsluttet en tur vil de blive viderestillet til en side der viser den, og dette er det endpoint der sender turens data.

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

I stedet for at sende et Run objekt, som er hvad det har oprettet og lagt i databasen, sendes her et RunStats objekt. RunStats er et DTO, et Data Transfer Object, som ikke er lagret i databasen, men i stedet bliver genereret når brugeren beder om information om en tur. Den nye information der bliver beregnet er hvor lang tid turen varede (duration), hvor hurtigt der blev løbet i gennemsnit (avgSpeedInKmt), og hvor hurtigt der i gennemsnit blev løbet minut for minut (avgSpeedInKmtPerMinute). Disse data udregnes serverside for at spare brugerens enhed besværet.

## PWA - Modtager

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

---------- CODE START ----------

----------- CODE END ----------- (Link till fuld kode)

## Resultat

Resultatet af denne udveksling af information er at en bruger kan sætte sin enhed til at logge deres lokation til en central database mens de bevæger sig, og efterfælgende er i stand til at få præsenteret deres bevægelse visuelt, samt information om varighed, hastighed, og andet, som vist på screenshottet(ental) nedenfor.

bileldeijijdle

# Brugervejledninger

## Oprettelse af bruger

## Login

## Ny løbetur

## Mine Løbeture

## Slet Løbetur

## Statistikker

# Kildeliste

1. <https://aws.amazon.com/what-is/api/> [↑](#footnote-ref-1)
2. <https://www.entityframeworktutorial.net/what-is-entityframework.aspx> [↑](#footnote-ref-2)
3. <https://www.json.org/json-en.html> [↑](#footnote-ref-3)
4. <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/> [↑](#footnote-ref-4)
5. <https://www.tutorialspoint.com/sql/sql-overview.htm> [↑](#footnote-ref-5)
6. <https://vegibit.com/the-declarative-nature-of-sql/> [↑](#footnote-ref-6)
7. <https://web.dev/what-are-pwas/> [↑](#footnote-ref-7)
8. <https://v1.vuejs.org/guide/overview.html> [↑](#footnote-ref-8)
9. <https://v1.vuejs.org/guide/overview.html> [↑](#footnote-ref-9)
10. <https://v1.vuejs.org/guide/overview.html> [↑](#footnote-ref-10)