

***FunRun***

Produktrapport

**Svendeprøve**

**Elev**

Mathias Frederik Græsholt

**Projektnavn**

FunRun

**Uddannelsessted**

TECHCOLLEGE

Struervej 70,

9220 Aalborg

**Elevplads**

EMD International

Niels Jernes Vej 10

9220 Aalborg Ø

**Projektperiode**

21. marts, 2022 - 19. april 2022

**Afleveringsdato**

19. april 2022

**Fremlæggelsesdato**

26. april 2022

**Vejledere**

Frank Rosbak

Lærke Brandhøj Kristensen

Censor

??

**Forord**

**Indholdsfortegnelse**

[Læsevejledning 5](#_Toc100429737)

[Indledning? 5](#_Toc100429738)

[Kravspecifikation 6](#_Toc100429739)

[Funktionalitet 6](#_Toc100429740)

[Testkonditioner 7](#_Toc100429741)

[Krav 8](#_Toc100429742)

[Information om teknologier 10](#_Toc100429743)

[API 10](#_Toc100429744)

[Entity Framework 10](#_Toc100429745)

[JSON 10](#_Toc100429746)

[bcrypt 10](#_Toc100429747)

[Azure 11](#_Toc100429748)

[Database 11](#_Toc100429749)

[T-SQL 11](#_Toc100429750)

[PWA 11](#_Toc100429751)

[Vue.js 12](#_Toc100429752)

[Overordnet arkitektur 13](#_Toc100429753)

[PWA - Afsender 13](#_Toc100429754)

[API - Modtager 15](#_Toc100429755)

[Database 16](#_Toc100429756)

[Api - Afsender 16](#_Toc100429757)

[PWA - Modtager 17](#_Toc100429758)

[Resultat 22](#_Toc100429759)

[Brugervejledninger 23](#_Toc100429760)

[1. Installation af app 23](#_Toc100429761)

[2. Oprettelse af bruger 23](#_Toc100429762)

[3. Log ind 23](#_Toc100429763)

[4. Ny løbetur 24](#_Toc100429764)

[5. Mine Løbeture 24](#_Toc100429765)

[6. Vis løbetur 24](#_Toc100429766)

[7. Slet Løbetur 24](#_Toc100429767)

[Kildeliste 26](#_Toc100429768)

# Læsevejledning

Denne rapport er en af to der hører til svendeprøveprojektet FunRun, skrevet af Mathias Frederik Græsholt. Dette er produktrapporten og den anden er procesrapporten.

I produktrapporten beskrives projektets produkt og teknologier. I procesrapporten beskrives forløbet, og hvordan produktet blev formet.

Det anbefales at man starter med at læse produktrapporten, og får et indblik i hvad projektet er, før man læser procesrapporten, hvori det beskrives hvordan det endelige produkt tog form.

I denne rapport bruges en del tekniske forkortelser, men især to er vigtige at kende:

**API** står for Application Programming Interface, og er en type program der faciliterer kommunikation mellem to andre teknologier.

**PWA** står for Progressive Web App, og er en type hjemmeside der kan downloades som en app.

Begge teknologier bliver beskrevet dybere i rapportens teknologiafsnit.

# Indledning

Denne rapport omhandler en proof-of-concept løsning på en motionsapp.

Systemet gør en bruger i stand til at optage løbeture og anden motionsbevægelse, at se alle deres tidligere ture på en overskuelig måde, og at få detaljeret information on en given tur, deriblandt visning på et kort.

Projektets udgangspunkt beskrives yderligere i tilhørende procesrapports case beskrivelse og problemformulering.

Rapporten indeholder kravspecifikation og testkonditioner, samt hvordan projektet lever op til de obligatoriske og valgfrie krav der er stillet. De teknologier der indgår i projektet beskrives, og der gives et indblik i funktionaliteten bag systemets hovedfunktion, at optage og vise ture. Der er ydermere inkluderet brugervejledninger i hvordan systemet benyttes.

# 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 Microsoft SQL Server Management 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-A |  |  |  | Kontrollér at brugeres passwords bliver hashet i databasen. |
| SS2-B |  |  |  | Kontrollér at brugere oprettet med samme password får forskellige hashes i databasen. |
| PO1 |  |  |  | Kontrollér at en uafhængig enhed kan downloade og benytte PWA’en. |
| 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.

Github link

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 få en grad af continous integration ind over.

Azure har også CI

Github actions side

**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**

Begge projektets dele kører som services på Azure, og er tilgængelige udefra.

En traditionel, fysisk server er trods forespørgsel ikke blevet stillet til rådighed for projektet.

**Valgfri elementer**

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

**APP Udvikling**

Projektet besvare kravet for App Udvikling ved det brug af en Progressive Web App. Denne app er udviklet i Vue.js, og er den del af systemet som brugere vil interagere med.

Appen kan tilgås på XXX

# 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).

## 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).

## Hosting

Alle projektet tre dele er hostede på Microsoft Azure.

Microsoft Azure er Microsofts Cloud Computing platform, på hvilken man kan hoste en bred vifte af forskellige cloud services til at opbevare eller transformere data. Microsoft Azure tilbyder en pay-per-use model, og er den hurtigst voksende Cloud Computing platform på markedet[[11]](#footnote-11).

Projekteta API er hosted i form af en App Service forbundet med en SQL database.

Projektets PWA er hosted i form af en Static Web App.

# Overordnet arkitektur

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



Arkitekturdiagram tegnet via Draw.io

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

## 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 ----------

await axios

  .post(process.env.VUE\_APP\_API\_URL + "api/Run", "", {

    headers: { Authorization: `Bearer ${localStorage.getItem("jwtToken")}` },

  })

  .then(async function (response) {

    console.log(response.data);

    runId = response.data.runId;

    startTime = Date.now();

    console.log(runId);

    //GetPoint

    status.value = "run started";

    //Test if wakeLock exists. Will fail on many traditional computers, but not devices such as phones.

    if ("wakeLock" in navigator) {

      try {

        lock = await navigator.wakeLock.request("screen");

      } catch (err) {

        // Error or rejection

        console.log("Wake Lock error: ", err);

      }

    }

  });

----------- CODE END -----------

(Fuld kode: <https://github.com/Graesholt/Svendeproeve/blob/main/pwa/src/views/newrun-view.vue>)

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.

Ud over dette ændres status.value til senere brug, og der startes en Wake Lock, som gør at en enheds skærm ikke vil låse automatisk.

Selve turens punkter bliver tilføjet med følgende kode.

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

watchId = navigator.geolocation.watchPosition(

  (position) => {

    console.log("position", position);

    let point = {

      longitude: position.coords.longitude,

      latitude: position.coords.latitude,

      altitude: position.coords.altitude,

    };

    console.log("point", point);

    //PostPoint

    if (status.value == "run started" || status.value == "running") {

      axios.post(process.env.VUE\_APP\_API\_URL + "api/Point/" + runId, point, { headers: { Authorization: `Bearer ${localStorage.getItem("jwtToken")}` } });

      runPolyline.addLatLng(new L.LatLng(position.coords.latitude, position.coords.longitude));

      if (status.value == "run started") {

        status.value = "running";

        timerInterval = setInterval(updateTimer, 10);

        refStartRunButton.value = "Stop løbetur";

      }

    }

    //MoveMap

    map.panTo(new L.LatLng(position.coords.latitude, position.coords.longitude));

    runnerTooltip.setLatLng(new L.LatLng(position.coords.latitude, position.coords.longitude));

  },

  () => {},

  { enableHighAccuracy: true }

);

----------- CODE END -----------

(Fuld kode: <https://github.com/Graesholt/Svendeproeve/blob/main/pwa/src/views/newrun-view.vue>)

navigator.geolocation.watchPosition()bliver kært sammen med oprettelsen af kortet når siden indlæses, kører hver gang den får en ny position fra enhedens GPS. En enheds GPS afhængig af kompleksitet kan optage mange værdier, men for projektet her er den kun væsentligt at behandle latitude, longitude, og altitude.

Så snart status.value er blevet ændret til "run started" af det forrige kald, vil efterfølgende punkter, ud over at opdatere kortet, blive sendt til databasen sammen med ID’et på den igangværende tur, samt tilføjes til runPolyline som er en linje på kortet der viser ens bevægelse løbende.

Ud over dette startes en timer på siden, og knappen opdateres til at kunne stoppe turen. Når brugerens tur er slut, trykker de på knappen igen, og dette ender watchPosition() (via clearWatch()), ender timeren, og frigiver Wake Lock, så der ikke sendes flere punkter, der går ikke ressourcer til at opdatere en timer der ikke længere er på skærmen, og brugerens enhed er igen fri til at låse skærmen.

## API - Modtager

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

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

        [HttpPost]

        public async Task<ActionResult<Run>> NewRun()

        {

            var run = new Run();

            run.dateTime = DateTime.Now; //Do not trust client time, server time is infallable

            run.user = await \_context.Users.FirstOrDefaultAsync(u => u.userId == GetUserId());

            run.deleted = false;

            \_context.Runs.Add(run);

            await \_context.SaveChangesAsync();

            return CreatedAtAction("GetRun", new { id = run.runId }, run);

        }

----------- CODE END -----------

(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/WebAPI/Controllers/RunController.cs>)

Den bruger metoden GetUserId() til at læse userId ud fra den token den modtager i headeren, 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 modtager et Point, modtages punktets data samt hvilket Run det hører til.

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

[HttpPost("{runId}")]

public async Task<ActionResult<Point>> NewPoint(Point point, int runId)

{

point.dateTime = DateTime.Now; //Do not trust client time, server time is infallable

var run = \_context.Runs.Include("points").FirstOrDefault(r => r.runId == runId);

run.points.Add(point);

//\_context.Points.Add(point);

await \_context.SaveChangesAsync();

return StatusCode(201);

}

----------- CODE END -----------

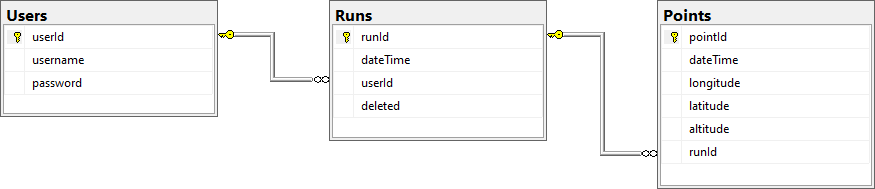
(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/WebAPI/Controllers/PointController.cs>)

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 hvilket 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 et 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 Point en User, da punktet i forvejen er tilknyttet et Run, som allerede er tilknyttet en User.

## Database

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



Entity Relation Diagram (ERD) genereret via Microsoft SQL Server Management Studio 18

Users tabellen indeholder data om brugere, som for eksempel brugernavn og password. Passwords er hashede med bcrypt og aldrig opbevaret eller behandlet af systemet i plain text. 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.

## Api - Afsender

Når en bruger har afsluttet en tur, vil de blive viderestillet til en side der viser information om den. Præcis samme side kan tilgås ved at vælge deres nyligste løbetur på oversigten der vises efter login.

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

        [HttpGet("{runId}")]

        public async Task<ActionResult<RunStats>> GetRun(int runId)

        {

            var run = await \_context.Runs.Include("user").Include("points").FirstOrDefaultAsync(r => r.runId == runId);

            //.Include("points") gets a list of points associated with this run as a property on the object.

            if (run.user.userId != GetUserId())

            {

                return Unauthorized();

            }

            if (run == null)

            {

                return NotFound();

            }

            RunStats runStats = new RunStats(run);

            return runStats;

        }

----------- CODE END -----------

(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/WebAPI/Controllers/RunController.cs>)

I stedet for at sende et Run objekt, som er hvad der er oprettet og lagret 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. De nye properties der bliver beregnet ud fra de allerede tilgængelige data er hvor lang tid turen varede (duration), hvor hurtigt der blev løbet i gennemsnit (avgSpeedInMetersPerSecond), og hvor hurtigt der i gennemsnit blev løbet minut for minut (avgSpeedPerMinuteInMetersPerSecond). Disse data udregnes serverside for at spare brugerens enhed.

Mapcentering

## PWA - Modtager

Når PWA’en modtager informationen bliver den fordelt på siden og præsenteret for brugeren.

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

<div class="center-div top-stat-div">

        <Card class="center-text stat-card">

          <template #title>

            <p class="stat-card-field">

              {{ refRun.duration }}<span class="subscript">{{ refDurationMilliseconds }}</span>

            </p>

          </template>

          <template #content>

            <p class="stat-card-field">Varighed</p>

          </template>

        </Card>

        <Card class="center-text stat-card">

          <template #title>

            <p class="stat-card-field">

              {{ refRun.distance }}<span class="subscript">{{ refDistanceUnit }}</span>

            </p>

          </template>

          <template #content>

            <p class="stat-card-field">Distance</p>

          </template>

        </Card>

      </div>

      <div class="center-div bottom-stat-div">

        <Card class="center-text stat-card">

          <template #title>

            <p class="stat-card-field">{{ refRun.avgSpeedInMetersPerSecond }}<span class="subscript">km/t</span></p>

          </template>

          <template #content>

            <p class="stat-card-field">Gns. hastighed</p>

          </template>

        </Card>

      </div>

      <div id="run-map"></div>

      <p class="center-text chart-header">Højdekurve</p>

      <div class="center-div scheme-div">

        <line-chart :data="refAltitudePointList" xtitle="Tid" ytitle="Højde i m" class="scheme" empty="Henter data..." :curve="false" :points="false" :min="refAltitudeSchemeMin" :max="refAltitudeSchemeMax"></line-chart>

      </div>

      <p class="center-text chart-header">Gennemsnitshastighed pr. minut</p>

      <div class="center-div scheme-div">

        <line-chart :data="refAvgSpeedPerMinutePointList" xtitle="Minut" ytitle="Km/t" class="scheme" empty="Henter data..." :curve="false" :points="false" :min="refAvgSpeedPerMinuteSchemeMin" :max="refAvgSpeedPerMinuteSchemeMax" :xmin="1" :xmax="refAvgSpeedPerMinuteSchemeLength" :colors="['#ff6600']"></line-chart>

      </div>

----------- CODE END -----------

(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/pwa/src/views/run-view.vue>)

Øverst på siden ses tre Card elementer der vil indeholde varighed, distance, og gennemsnitshastighed. Efter disse følger en div der vil blive til sidens map, og til sidst ses to line-chart der viser turens højdekurve, og hastigheden per minut.

Alle disse elementer, undtagen kortet, bruger Vue.js’ indbyggede databinding til automatisk at opdatere når deres databundne ref()-værdier ændrer sig.

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

axios

  .get(process.env.VUE\_APP\_API\_URL + "api/Run/" + router.currentRoute.value.params.runId, {

    headers: { Authorization: `Bearer ${localStorage.getItem("jwtToken")}` },

  })

  .then(function (response) {

    console.log(response.data);

    refRun.value = response.data;

    refHeader.value = new Date(refRun.value.dateTime).toLocaleDateString() + " - " + new Date(refRun.value.dateTime).toLocaleTimeString();

    console.log(refRun.value.centerLatitude);

    var map = L.map("run-map").setView(new L.LatLng(refRun.value.centerLatitude, refRun.value.centerLongitude), 14);

    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {

      attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',

      maxZoom: 19,

      tileSize: 256,

    }).addTo(map);

    var mapPointList = [];

    refAltitudePointList.value = [];

    refRun.value.points.forEach((element) => {

      mapPointList.push(new L.LatLng(element.latitude, element.longitude));

      if (refAltitudePointList.value.length == 0 || roundToTwo(element.altitude) != refAltitudePointList.value[refAltitudePointList.value.length - 1][1]) {

        refAltitudePointList.value.push([new Date(element.dateTime).toLocaleTimeString(), roundToTwo(element.altitude)]);

        if (!refAltitudeSchemeMin.value || element.altitude < refAltitudeSchemeMin.value) {

          refAltitudeSchemeMin.value = element.altitude;

        }

        if (!refAltitudeSchemeMax.value || element.altitude > refAltitudeSchemeMax.value) {

          refAltitudeSchemeMax.value = element.altitude;

        }

      }

    });

    console.log(refAltitudePointList);

    refAltitudeSchemeMin.value = Math.round(refAltitudeSchemeMin.value - AltitudeSchemeMargin);

    refAltitudeSchemeMax.value = Math.round(refAltitudeSchemeMax.value + AltitudeSchemeMargin);

    for (let i = 0; i < refRun.value.avgSpeedPerMinuteInMetersPerSecond.length; i++) {

      var speedInKmt = roundToTwo((refRun.value.avgSpeedPerMinuteInMetersPerSecond[i] \* 3600) / 1000)

      refAvgSpeedPerMinutePointList.value.push([i + 1, speedInKmt]);

      if (!refAvgSpeedPerMinuteSchemeMin.value || speedInKmt < refAvgSpeedPerMinuteSchemeMin.value) {

        refAvgSpeedPerMinuteSchemeMin.value = speedInKmt;

      }

      if (!refAvgSpeedPerMinuteSchemeMax.value || speedInKmt > refAvgSpeedPerMinuteSchemeMax.value) {

        refAvgSpeedPerMinuteSchemeMax.value = speedInKmt;

      }

    }

    console.log(refAvgSpeedPerMinutePointList);

    refAvgSpeedPerMinuteSchemeLength.value = refRun.value.avgSpeedPerMinuteInMetersPerSecond.length;

    refAvgSpeedPerMinuteSchemeMin.value = Math.round(refAvgSpeedPerMinuteSchemeMin.value - AvgSpeedPerMinuteSchemeMargin);

    refAvgSpeedPerMinuteSchemeMax.value = Math.round(refAvgSpeedPerMinuteSchemeMax.value + AvgSpeedPerMinuteSchemeMargin);

    var runPolyline = new L.Polyline(mapPointList, {

      color: "blue",

      opacity: 0.5,

      smoothFactor: 1.5,

    }).addTo(map);

    var startTooltip = L.tooltip({

      direction: "top",

      permanent: true,

    })

      .setLatLng(mapPointList[0])

      .setContent("Start")

      .addTo(map);

    var endTooltip = L.tooltip({

      direction: "bottom",

      permanent: true,

    })

      .setLatLng(mapPointList[mapPointList.length - 1])

      .setContent("Slut")

      .addTo(map);

    var durationSplit = refRun.value.duration.split(".");

    while (durationSplit[1] > 100) {

      durationSplit[1] = durationSplit[1] / 10;

    }

    refRun.value.duration = durationSplit[0];

    refDurationMilliseconds.value = Math.round(durationSplit[1]);

    if (refRun.value.distance < 1000) {

      refRun.value.distance = Math.round(refRun.value.distance);

      refDistanceUnit.value = "m";

    } else if (refRun.value.distance >= 1000) {

      refRun.value.distance = roundToTwo(refRun.value.distance / 1000);

      refDistanceUnit.value = "km";

    }

    refRun.value.avgSpeedInMetersPerSecond = roundToTwo((refRun.value.avgSpeedInMetersPerSecond \* 3600) / 1000);

  })

----------- CODE END -----------

(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/pwa/src/views/run-view.vue>)

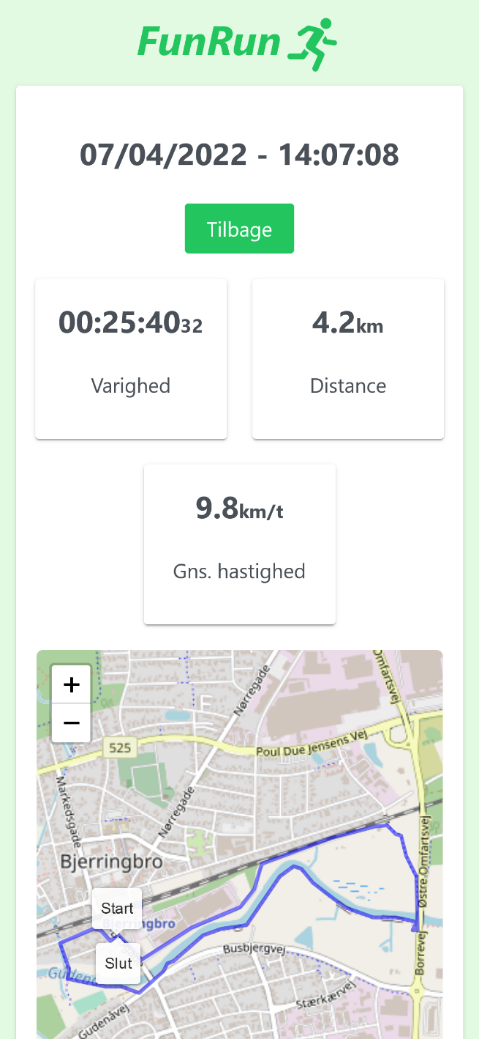
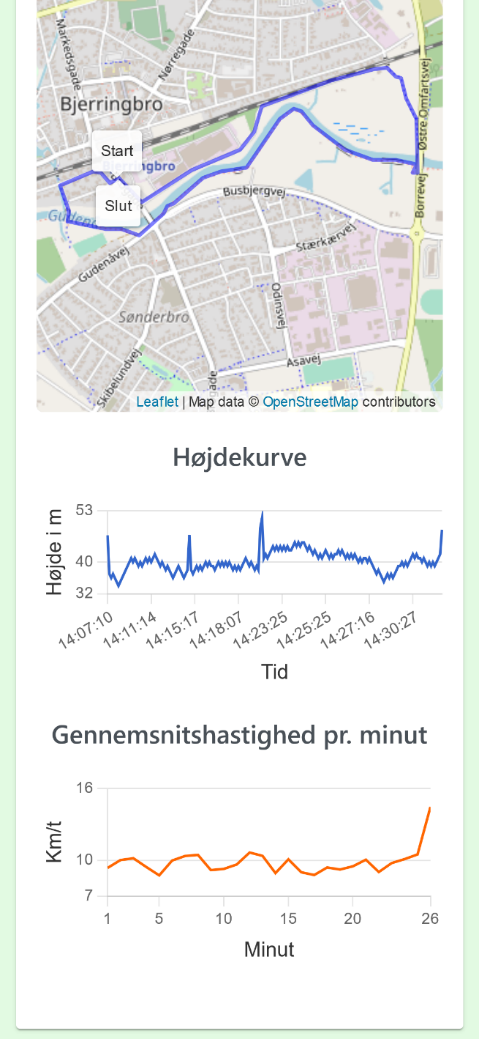
Mapcentering

I denne kode forspørges indformationen fra API’et, og når den ankommer bliver den sat gemt i objektet refRun som nogen af dom-elementerne er bundet til. Efter det bliver sidens kort sat op, og der loopes gennem turens points for at lave en brugbar liste (mapPointList) over koordinator til fremvisning derpå (samtidig skabes en liste (refAltitudePointList) til højdekurven, siden den skal gennem samme data). Efter dette oprettes og konfigureres runPolyline, som viser selve turen, og startTooltip og endTooltip, som viser start og slut på kortet.

En mængde kode er udeladt her ved afmærkningen [CODE OMITTED], for læselighed og forståelsens skyld. Koden herfra var formatering og omregning af de resterende databunde værdier på siden, disse værende refRun.duration, refRun.distance, refRun.avgSpeedInMetersPerSecond, og refAvgSpeedPerMinutePointList, som looper hen over anden data end refAltitudePointList, men er konfigureret på meget samme måde.

## Resultat

Resultatet af denne udveksling af information er at en bruger kan bede sin enhed om at logge deres position til en central database mens de bevæger sig, og efterfølgende er i stand til at få præsenteret deres bevægelse visuelt, samt information om varighed, hastighed, og mere, som vist nedenfor.

# Brugervejledninger

Velkommen til FunRun!

## 1. Installation af app

1. Naviger til XXXXX på din enhed, i en browser der kan downloade Progressive Web Apps. Hvis du er i tvivl om en given browser tilbyder denne funktionalitet, kan du altid bruge Google Chrome, som er gratis.
2. Når siden er indlæst, bør du finde hvordan man installere den som app via den browser du har valgt at benytte. I Chrome skal du trykke på de tre små prikker i øverste højre hjørne for at få en dropdown-menu med valgmuligheder. En af disse vil være ”Installer FunRun”.
3. Bekræft at du gerne vil installere FunRun, hvis en pop-up dialog vises.
4. Når appen er installeret, vil du kunne lokalisere den på din appoversigt som en hver anden app. Du er nu klar til åbne den, fuldstændig uafhængig af din browser.

## 2. Oprettelse af bruger

Installer først?

1. Åben appen.

Når appen åbner vil du se en login skærm.

1. Tryk på ”registrer”-knappen nederst på siden.

Du vil nu se en side hvor du kan oprette en ny bruger.

1. Indtast et brugernavn og et kodeord.
2. Tryk på ”registrer”-knappen når dine oplysninger er tastet ind.

Hvis brugernavnet allerede er i brug, vil du få dette at vide. I dette tilfælde vil du skulle vælge et andet brugernavn hvis du vil bruge systemet. Dit brugernavn skal mindst være X tegn, og må ikke indeholde specialtegn.

Hvis dine kodeord ikke er indtastet ens, vil du få dette at vide. Dit kodeord skal være mindst X tegn, og indeholde mindst ét stort, og ét småt bogstav, samt mindst ét tal eller specialtegn.

Hvis oplysningerne bliver godtaget, vil du blive omdirigeret til login siden. Du er nu klar til at logge ind med din nye bruger.

## 3. Log ind

Registrer først?

1. Åben appen

Når appen åbner vil du se en login skærm.

1. Indtast de oplysninger du brugte da du registrerede dig som ny bruger.
2. Tryk på ”log ind”-knappen når dine oplysninger er tastet ind.

Hvis oplysningerne bliver godtaget, vil du blive logget ind. Du er nu klar til at bruge systemet.

Når du vil logge ud

## 4. Ny løbetur

Login først

Giv lov til location!

Når du ser oversigten over dine tidligere ture er du klar til at optage en ny løbetur.

1. Tryk på ”Ny løbetur”-knappen øverst på siden.

Du vil nu blive omstillet til en side med et kort og en timer der står på 0.

1. Når du er klar til at begynde, tryk på ”Start løbetur”-knappen.

Efter et kort øjeblik vil din enhed sende din position til FunRuns database og tegne din bevægelse in på kortet løbende. Timeren vil også vise dig hvor længe du har løbet.

Dette virker bedst hvis du slår WiFi fra på din enhed, og holder skærmen tændt og appen i fokus.

1. Nå du er færdig med din tur, trykker du på ”Stop løbetur”-knappen.

Du vil nu blive omdirigeret til en side hvor du kan se information om din netop afsluttede løbetur.

Tilbage knap for at vende tilbage til mine løbeture

## 5. Mine Løbeture

Lav en løbetur først

1. Log ind i appen, eller tryk på ”tilbage”-knappen når du ser en færdig løbetur.

Du vil se en side med dine tidligere løbeture, markeret med dato og klokkeslæt.

Listen kan indeholde ti ture per side, og kalenderne over listen kan bruges til at afgrænse resultaterne via start og slutdato.

Hvis du trykker på en løbetur i listen, vil du blive omdirigeret til en side med information on turen.

## 6. Vis løbetur

Lav en løbetur først

1. Færdiggør en løbetur, eller tryk på en i listen over tidligere løbeture.

Du vil nu blive omdirigeret til en side hvor du kan se information om den givne løbetur.

Øverst på siden, under turens tidspunkt og sidens ”tilbage”-knap, ses varighed, distance, og gennemsnitshastighed.

Under disse ses et kort på hvilket turen, samt dens start- og slut-steder er markeret.

Nederst på siden ses to diagrammer. Det første viser en højdekurve over hvad din enhed har meddelt under turen, mens det nederste viser gennemsnitshastigheden per minut turen har varet.

Tilbage knap for at vende tilbage til mine løbeture

## 7. Slet Løbetur

Se mine løbeture først

1. Log ind i appen, eller tryk på ”tilbage”-knappen når du ser en færdig løbetur.

Du vil se en side med dine tidligere løbeture, markeret med dato og klokkeslæt.

Hver tur har i højre side en rød knap med en lille skrallespand.

1. Kontrollér at du ved hvilken tur har med at gøre før du trykker på den røde knap.
2. Tryk på den røde knap.

Turen vil inden længe forsvinde fra listen, og nu være markeret som slettet i FunRuns database.

Gendan slettet data

# Kildeliste

kildeliste titel forfatter årstal link

<https://aws.amazon.com/what-is/api/>

<https://www.entityframeworktutorial.net/what-is-entityframework.aspx>

<https://www.json.org/json-en.html>

<https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>

<https://www.tutorialspoint.com/sql/sql-overview.htm>

<https://vegibit.com/the-declarative-nature-of-sql/>

<https://web.dev/what-are-pwas/>

<https://v1.vuejs.org/guide/overview.html>

<https://www.simplilearn.com/tutorials/azure-tutorial/what-is-azure>

Test

Kontroller korrekt version!

FunRunTesting\_\_/\_\_/\_\_\_\_ \_\_:\_\_:\_\_

Password: tqewsetritnygu

Testbrugere bliver slettet automatisk

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)
11. <https://www.simplilearn.com/tutorials/azure-tutorial/what-is-azure> [↑](#footnote-ref-11)