

***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**

Allan Kaa Jensen

**Forord**

Denne rapport er en af to skrevet til svendeprøveprojektet FunRun af Mathias Frederik Græsholt.

Projektet er udarbejdet i perioden 21. marts, 2022 til 19. april 2022, med vejledning fra Frank Rosbak og Lærke Brandhøj Kristensen.

I denne rapport beskriver jeg hvordan jeg kom frem til mit endelige produkt, som beskrives i denne rapports case beskrivelse og problemformulering, og i den tilhørende produktrapport.

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.

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.

Jeg takker alle involverede og håber at det er interessant læsning.

**Indholdsfortegnelse**

[Læsevejledning 6](#_Toc100658234)

[Indledning 6](#_Toc100658235)

[Kravspecifikation 7](#_Toc100658236)

[Funktionalitet 7](#_Toc100658237)

[Testkonditioner 8](#_Toc100658238)

[Krav 9](#_Toc100658239)

[Information om teknologier 11](#_Toc100658240)

[API 11](#_Toc100658241)

[Entity Framework 11](#_Toc100658242)

[JSON 11](#_Toc100658243)

[bcrypt 11](#_Toc100658244)

[Database 12](#_Toc100658245)

[T-SQL 12](#_Toc100658246)

[PWA 12](#_Toc100658247)

[Vue.js 13](#_Toc100658248)

[Hosting 13](#_Toc100658249)

[Overordnet arkitektur 14](#_Toc100658250)

[PWA - Afsender 14](#_Toc100658251)

[API - Modtager 16](#_Toc100658252)

[Database 17](#_Toc100658253)

[Api - Afsender 18](#_Toc100658254)

[PWA - Modtager 19](#_Toc100658255)

[Resultat 22](#_Toc100658256)

[Brugervejledninger 23](#_Toc100658257)

[1. Installation af app 23](#_Toc100658258)

[2. Oprettelse af bruger 23](#_Toc100658259)

[3. Log ind 23](#_Toc100658260)

[4. Ny løbetur 24](#_Toc100658261)

[5. Mine Løbeture 24](#_Toc100658262)

[6. Vis løbetur 24](#_Toc100658263)

[7. Slet Løbetur 24](#_Toc100658264)

[Kildeliste 26](#_Toc100658265)

# Læsevejledning

Denne rapport er en af to der hører til svendeprøveprojektet FunRun.

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.

Samlet kildeliste findes bagerst i rapporten

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.

Projektets kode, dele af hvilken forekommer i afsnittet Overordnet Arkitektur, er dokumenteret på engelsk grundet at resten af koden er skrevet på engelsk.

Links til dele a projektet i denne rapport er afkortet med tjenesten Bitly.

Dette er gjort for læselighed og belejlighed, men linkene leder stadig til de præcis samme adresser.

# Indledning

Som teknologien skrider frem, kræves mindre og mindre af vores kroppe, som flere timer end aldrig før får lov at sidde stille.

Procenten af befolkningen der lider af overvægt, er stadigt stigende, og dette er et problem for folkesundheden der nedsætter livskvaliteten.

Folk der ønsker at komme dette til livs, kan vælge et væld a motionsmuligheder for at holde sig i form, men en af de mest tilgængelige motionsform er nok løb.

Løb kræver ikke noget udstyr, og næsten alle kan gøre det på et niveau der passer til dem.

Det eneste ville måske være et system der kunne vise ens ture og give en indblik i relevant statistik deromkring, og derved fastholde løbere, som uden konkret bogføring og visualisering af deres fysiske motion hurtigt kunne miste interesse.

Projektet FunRun er et svar på dette, og denne rapport omhandler produktet og teknologierne bag det.

# 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 | X |  |  | Database skal inspiceres grundigt i Microsoft SQL Server Management Studio efter opsætning. |
| SF1-A | X |  |  | Kontrollér via swagger og Microsoft SQL Server Management Studio at API’ets endpoints modtager og lagre data korrekt. |
| SF1-B | X |  |  | Kontrollér i Microsoft SQL Server Management Studio at data fra PWA forekommer i databasen som forventet. |
| SF2-A | X |  |  | Kontrollér via swagger at API’ets endpoints returnere data som forventet. |
| SF2-B | X |  |  | Kontrollér at data fra database forekommer i PWA som forventet. |
| SS1-A | X |  |  | Kontrollér via Microsoft SQL Server Management Studio at brugeres passwords bliver hashed i databasen. |
| SS2-B | X |  |  | Kontrollér at brugere oprettet med samme password får forskellige hashes i databasen. |
| PO1 | X |  |  | Kontrollér at en uafhængig enhed kan downloade og benytte PWA’en. |
| PF1 | X |  |  | Kontrollér at brugere kan oprettes og logge ind og ud uden problemer. |
| PF2 | X |  |  | Kontrollér at oplysninger fra PWA om lokation kommer ordentligt igennem til databasen og med den rette frekvens. |
| PG1 | X |  |  | Kontrollér at PWA kan fremvise tidligere ture korrekt og hensigtsmæssigt. |
| PG2 | X |  |  | 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 obligatoriske 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.

Projektets GitHub repository: <https://github.com/Graesholt/Svendeproeve>

Derudover er GitHub også konfigureret med GitHub Actions til at builde, teste, og deploye API, og builde og deploye PWA ved hvert push. Det vil sige at API er sat op med fuldbyrdet Continuous Integration, og PWA er sat op med Continuous Deployment.

Projektets GitHub Actions side: <https://github.com/Graesholt/Svendeproeve/actions>

**Sikkerhed**

Der er blevet taget højde for sikkerhed i projektet ved at brugeres passwords bliver hashed 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 udveksling af information mellem API og PWA via HTTPS, den mere sikre udvidelse af HTTP.

Ydermere validere systemet brugernavn og kodeord, det vil sige al tekst i systemet der er indtastet af en bruger, før denne interagerer med databasen på nogen måde. På denne måde opnås sikkerhed mod SQL injection angreb.

**Test**

Projektets API har en række unit tests der sikrer at funktionen til at validere brugere fungere som forventet. Fordi denne del af API’et egnede sig til unit testing, gav dette mulighed for at bruge test driven development. Da password valideringen skulle implementeres, blev testene skrevet som det første, for at have en række data der skulle tillades eller forbydes, og derefter blev koden dertil skrevet.

API’ets unit tests: <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/UnitTests/UnitTests.cs>

Ud over dette, har API’et en mængde tests af endpoints. Disse black-box testes, da API’ets afhængighed af tokens gjorde endpointmetoder vanskelige, hvis ikke umulige at unit teste. Testene kalder i stedet endpoints udefra og kontrollerer at de får de korrekte responses, og at uautoriserede burgere ikke kan få manipulere information der ikke tilhører dem.

API’ets black-box tests: <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/UnitTests/BlackboxTests.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 ved at autotestscases er utroligt smarte og effektive, men tager lang tid at opbygge korrekt.

Brugertestcases er vedlagt som bilag 1.

Ud over dette har systemet været testet løbende af et antal uafhængige brugere.

**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, samt dets SQL server, kører på Microsoft Azure. Hoveddelene, API og PWA kører som services, 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 krav, skal projektet opfylde mindst et af en række valgfrie krav. Dette projekt opfylder to.

**Klient/Server**

Projektet har en server, i form af en SQL server, der kører på Microsoft Azure.

Projektet klient er PAW’en, som kan tilgå data fra serveren via API’et.

**APP Udvikling**

Projektet besvarer kravet for App Udvikling ved dets brug af en Progressive Web App. Denne app er udviklet i Vue.js, og er den del af systemet som brugere vil have adgang til og interagere med API’et igennem.

Link til app: <https://polite-grass-08a9c5303.1.azurestaticapps.net>

# Information om teknologier

Dette afsnit vil omhandle information om de forskellige teknologier der indgår i 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 udveksle data[[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 data i tabeller, med kolonner og rækker, kan data tilføjes og tilgås som objekter, hvilket gør dataene meget nemmere at arbejde med og forstå 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 håndgribeligt 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 velegnet 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 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 hashed 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. Det gør også at databasen hurtigt kan sættes op igen, skulle dette være nødvendigt.

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. En PWA kodes i 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 mellemmand (viewmodel). 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 navigations- eller side-bar, 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 projektets 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).

Projektets 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;

    console.log(runId);

    //Ready to log points, now that runId is known

    status.value = "run started";

    //Test if wakeLock exists. Will fail on many traditional computers, but not on

devices such as phones.

    if ("wakeLock" in navigator) {

      try {

        //Set wakeLock, preventing screen from locking

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

      } catch (err) {

        //Error occurred

        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 runId 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 wakeLock, 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);

    //Configure point object

    let point = {

      longitude: position.coords.longitude,

      latitude: position.coords.latitude,

      altitude: position.coords.altitude,

    };

    console.log("point", point);

    //If a run is in progress

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

      //Post point to API

      axios.post(process.env.VUE\_APP\_API\_URL + "api/Point/" + runId, point, {

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

      //Add point to map polyline

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

      //If first point since run started

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

        status.value = "running";

        //Set Timer start time

        startTime = Date.now();

        //Update timer ten times a second

        timerInterval = setInterval(updateTimer, 10);

        //Change button to show it will now end run

        refStartRunButton.value = "Stop løbetur";

      }

    }

    //Move Map

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

    //Move current location tooltip

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

  },

  () => {},

  { enableHighAccuracy: true }

);

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

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

navigator.geolocation.watchPosition()bliver kørt sammen med oprettelsen af kortet når siden indlæses, of funktionen i den 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 runId 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 wakeLock, så der ikke sendes flere punkter, der går ikke ressourcer til at opdatere en timer der ikke længere er på skærmen, og brugerens enhed igen er 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();

    //Do not trust client time, server time is reliable

    run.dateTime = DateTime.UtcNow;

    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>)

API’et 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. dateTime genereres serverside i UTC format 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)

{

    //Do not trust client time, server time is reliable

    point.dateTime = DateTime.UtcNow;

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

r.runId == runId);

    if (run == null)

    {

        return NotFound();

    }

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

    {

        return Unauthorized();

    }

    run.points.Add(point);

    //\_context.Points.Add(point);

    await \_context.SaveChangesAsync();

    return StatusCode(201);

}

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

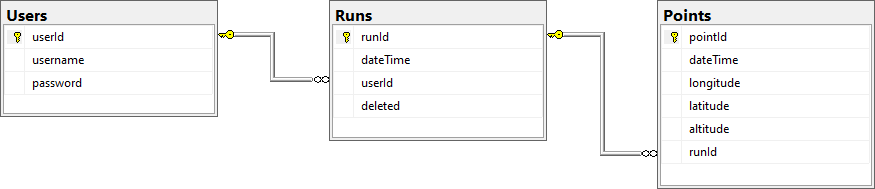
(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/WebAPI/WebAPI/Controllers/PointController.cs>)

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 databasekald 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 token kun til at sikre at kaldet er gyldigt, for at forhindre en forkert User i at tilføje ugyldige Points til Runs de ikke ejer. 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, som forklaret ovenfor, 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();

    }

    if (run.points.Count() == 0)

    {

        return UnprocessableEntity();

    }

    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.

## 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"

: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 elementer 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;

    //Header configured. UTC time converted to locale time with en-GB formatting

    refHeader.value = new Date(refRun.value.dateTime + "Z").toLocaleDateString(

"en-GB") + " - " + new Date(refRun.value.dateTime + "Z").toLocaleTimeString(

"en-GB");

    //Map: creation

    var map = L.map("run-map");

    //Map: tilelayer configuration

    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);

    //mapLatLngBounds holds the bounds we are interesting in viewing on map

    var mapLatLngBounds = new L.LatLngBounds();

    //mapPointList holds the LatLngs of each point of the Run

    var mapPointList = [];

    //Loops through points configuring both variables

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

      var pointLatLng = new L.LatLng(element.latitude, element.longitude);

      mapLatLngBounds.extend(pointLatLng);

      mapPointList.push(pointLatLng);

    });

    //Set map bounds

    map.fitBounds(mapLatLngBounds);

    //Map: polyline configuration

    var runPolyline = new L.Polyline(mapPointList, {

      color: "blue",

      opacity: 0.5,

      smoothFactor: 1.5,

    }).addTo(map);

    //Map: start tooltip configuration

    var startTooltip = L.tooltip({

      direction: "top",

      permanent: true,

    })

      .setLatLng(mapPointList[0])

      .setContent("Start")

      .addTo(map);

    //Map: end tooltip configuration

    var endTooltip = L.tooltip({

      direction: "bottom",

      permanent: true,

    })

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

      .setContent("Slut")

      .addTo(map);

[CODE OMITTED]

  })

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

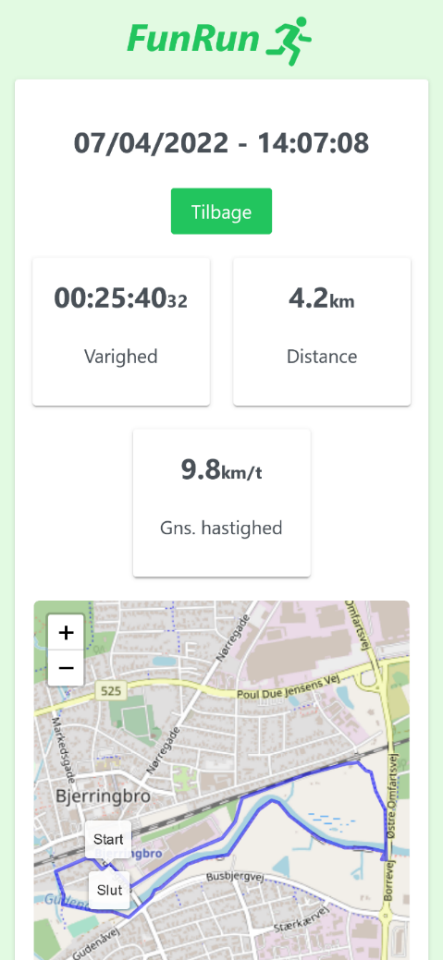
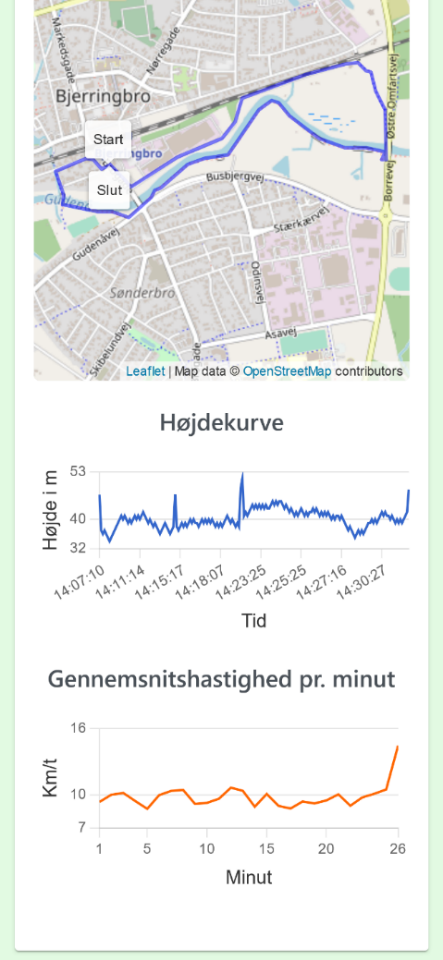
(Fuld kode <https://github.com/Graesholt/Svendeproeve/blob/main/pwa/src/views/run-view.vue>)

I denne kode forspørges indformationen fra API’et, og når den ankommer bliver den gemt i objektet refRun som nogen af dom-elementerne er bundet til porperties på. Sidens header bliver omregnet fra UTC tid fra databasen til lokal tid og dato, med Engelsk formatering. Efterfølgende bliver sidens kort sat op, og der loopes gennem turens points hovedsageligt for at gøre to ting. Først og fremmest extendes mapLatLngBounds med hvert punkt for at lave de yderlighedspunkter brugeren er interesseret i at få vist på kortet. Derudover skabes en brugbar liste (mapPointList) over koordinator til fremvisning derpå. Efter loopet sættes kortets synsfelt, og efter dette oprettes og konfigureres runPolyline, som viser selve turen fra mapPointList, 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 omregning og formatering af de resterende databunde værdier på siden, disse værende refRun.duration, refRun.distance, refRun.avgSpeedInMetersPerSecond, refAltitudePointList, og refAvgSpeedPerMinutePointList. For de to sidste variabler loopes hen over data fra refRun, og dette formateres til visning i diagrammerne på siden.

## 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!

Denne manual rettet mod nye brugere og brugere der ønsker et mere detaljeret indblik i brugen af systemet.

Punkterne i manualen er som udgangspunkt beregnede til at blive fulgt kronologisk

## 1. Installation af app

FunRun kan installeres på en enhed via en a flere browsere, skønt der er browsere der ikke understøtter installation a apps på denne måde.

Denne guide går ud fra at brugeren bruger Google Chrome, som er gratis.

1. Naviger til adressen <https://polite-grass-08a9c5303.1.azurestaticapps.net> på din enhed.

Når siden er indlæst vil du se FunRuns ”login”-side.

1. Tryk på de tre små prikker i øverste højre hjørne for at få en dropdown-menu med valgmuligheder.
2. Tryk på ”Installer FunRun”.
3. Bekræft at du gerne vil installere FunRun, i den pop-up dialog der vises.
4. Når appen er installeret, vil du kunne lokalisere den på din appoversigt som enhver anden app. Du er nu klar til åbne FunRun, fuldstændig uafhængig af din browser.

## 2. Oprettelse af bruger

Dette punkt går ud fra at brugeren har installeret FunRun på deres enhed (se punkt 1).

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 6 tegn.

Hvis dine kodeord ikke er indtastet ens, vil du få dette at vide. Dit kodeord skal være mindst 8 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

Dette punkt går ud fra at brugeren har registreret en FunRun bruger (se punkt 2).

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 din bruger af appen, er det så nemt som at trykke på den røde ”log ud”-knap.

## 4. Ny løbetur

Dette punkt går ud fra at brugeren har logget ind via en FunRun bruger (se punkt 3).

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.

Hvis det er første gang du ser denne side, vil du blive spurgt om FunRun må få adgang til din lokationsdata. Denne vil kun blive indsamlet i fremtiden når du kan se denne side, og det er kun dig der kan få dataene at se efterfølgende.

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”-knappen vil returnere dig til oversigten over dine tidligere ture.

## 5. Mine Løbeture

Dette punkt går ud fra at brugeren har optaget en løbetur (se punkt 4).

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 for viste ture.

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

## 6. Vis løbetur

Dette punkt går ud fra at brugeren har optaget en løbetur (se punkt 4).

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”-knappen vil returnere dig til oversigten over dine tidligere ture.

## 7. Slet Løbetur

Dette punkt går ud fra at brugeren har kendskab til ”Mine løbeture”-siden (se punkt 5).

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.

Slettet data er skjult for brugeren, men eksisterer stadig i FunRuns database. Hvis du ved et uheld har slettet en tur du gerne vil have gendannet, kontakt da kundeservice, og hvis muligt vil FunRuns personale forsøge at fremfinde dine data.

FunRun tager ikke ansvar for bruger slettet data der ikke umiddelbart kan fremfindes og gendannes, da dette er en tidskrævende proces.

# Kildeliste

What is an API?

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

(Sidst besøgt 12/04/2022)

What is Entity Framework?

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

(Sidst besøgt 12/04/2022)

Introducing JSON

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

(Sidst besøgt 12/04/2022)

Hashing in Action: Understanding bcrypt

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

(Sidst besøgt 12/04/2022)

SQL - Overview

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

(Sidst besøgt 12/04/2022)

The Declarative Nature of SQL

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

(Sidst besøgt 12/04/2022)

What are Progressive Web Apps?

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

(Sidst besøgt 12/04/2022)

Vue.js overview

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

(Sidst besøgt 12/04/2022)

What is Microsoft Azure: How Does It Work and Services

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

(Sidst besøgt 12/04/2022)

**Bilag 1**

**Brugertestcases**

FunRun systemet kan kører på to hovedplatforme, web og app. Hver test beskriver hvilke platforme den kan udføres på, og hver test bør udføres på alle egnede platforme før den kan beskrives som bestået.

Hvis en test fejler, beskriv fejlen kortfattet i Note/fejl kolonnen ud for det trin hvor den forekom, og aflever papiret i udviklingsafdelingen. Kontroller venligst at navn på tester og software version (forefindes nederst på loginsiden) er udfyldt, og at platform er tydeligt markeret.

Til tests der kræver login, hvor andet ikke er specificeret, bruges brugertestcase oplysningerne:

Brugernavn: FunRunUserTesting

Kodeord: tQeWsEtRiTnYg5

**1. Registreringsside test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Start FunRun. | System starter på enten login eller oversigtsside. |  |
| 2 | (Hvis på oversigtsside)  Tryk på ”Log ud”-knappen. | System viser loginside. |  |
| 3 | Tryk på ”Registrer”-knappen. | System viser Registreringsside. |  |
| 4 | Tryk på ”Registrer”-knappen. | Siden viser en brugernavn-fejl. |  |
| 5 | Indtast et brugernavn med mindre end 6 karakterer. | Ingen ændring. |  |
| 6 | Tryk på ”Registrer”-knappen. | Siden viser en brugernavn-fejl. |  |
| 7 | Indtast et brugernavn med mere end 30 karakterer. | Ingen ændring. |  |
| 8 | Tryk på ”Registrer”-knappen. | Siden viser en brugernavn-fejl. |  |
| 9 | Indtast et brugernavn mellem 8 og 30 karakterer der indeholder ”#”. | Ingen ændring. |  |
| 10 | Tryk på ”Registrer”-knappen. | Siden viser en brugernavn-fejl. |  |
| 11 | Indtast brugernavn: FunRunUserTesting | Ingen ændring. |  |
| 12 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 11 | Indtast et brugernavn der bruger den nuværende tid og følger konventionen: FRT-(år).(måned).(dag)-(time).(minut).(sekund).FRT, for eksempel ”FRT-2022.04.12-17.24.05.FRT” | Ingen ændring. |  |
| 12 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 13 | Indtast et kodeord med mindre end 8 karakterer. | Ingen ændring. |  |
| 14 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 15 | Indtast et kodeord med mere end 99 karakterer. | Ingen ændring. |  |
| 16 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 17 | Indtast et kodeord med mellem 8 og 99 karakterer der kun indeholder store bogstaver. | Ingen ændring. |  |
| 18 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 19 | Indtast et kodeord med mellem 8 og 99 karakterer der kun indeholder små bogstaver. | Ingen ændring. |  |
| 20 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 21 | Indtast et kodeord med mellem 8 og 99 karakterer der indeholder store og små bogstaver, men hverken tal eller specialkarakterer. | Ingen ændring. |  |
| 22 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 23 | Indtast et kodeord med mellem 8 og 99 karakterer der indeholder store og små bogstaver, mindst et tal, og ”#”. | Ingen ændring. |  |
| 24 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 25 | Indtast kodeord: tQeWsEtRiTnYg5  I kun øverste felt. | Ingen ændring. |  |
| 26 | Tryk på ”Registrer”-knappen. | Siden viser en kodeord-fejl. |  |
| 27 | Indtast kodeord: tQeWsEtRiTnYg5  I nederste felt. | Ingen ændring. |  |
| 28 | Tryk på ”Registrer”-knappen. | System viser success-besked efterfulgt af loginside. |  |

\*Testbrugere oprettet per instruktionen ovenfor bliver slettet automatisk i databasen

**2. Loginside test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Start FunRun. | System starter på enten login eller oversigtsside. |  |
| 2 | (Hvis på oversigtsside)  Tryk på ”Log ud”-knappen. | System viser loginside. |  |
| 3 | Indtast brugernavn: FunRunUserTesting | Ingen ændring. |  |
| 4 | Indtast et vilkårligt, men ukorrekt kodeord. | Ingen ændring. |  |
| 5 | Tryk på ”Log ind”-knappen. | Siden viser en fejl. |  |
| 6 | Indtast kodeord: tQeWsEtRiTnYg5 | Ingen ændring. |  |
| 7 | Tryk på ”Log ind”-knappen. | System viser oversigtsside. |  |

**3. Ny løbetur test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Log ind på FunRun. | System viser oversigtsside. |  |
| 2 | Tryk på ”Ny løbetur”-knappen | System viser ”Ny løbetur”-side.  Kort opdaterer position periodisk. |  |
| 3 | Tryk på ”Start løbetur”-knappen. | Knap bliver grå og ændrer tekst. |  |
| 4 | Vent på a løbetur starter. | Knap bliver rød og ændrer tekst. Timer starter. |  |
| 5 | Gå en kort tur i 30 sekunder. | Kort opdaterer position periodisk, og tur indtegnes derpå. |  |
| 6 | Tryk på ”Stop løbetur”-knappen. | System viser informationsside. |  |

**4. Oversigtsside test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Log ind på FunRun. | System viser oversigtsside. |  |
| 2 | (Hvis ingen løbeture i øversigt)  Lav en løbetur så mindst en vises. | Mindst en løbetur vises i oversigt. |  |
| 3 | Sæt begge kalender datoer så de viser et tidsrum før seneste løbetur fandt sted. | Seneste løbetur forsvinder fra oversigt. |  |
| 4 | Sæt begge kalender datoer så seneste løbetur vises. | Mindst en løbetur vises i oversigt. |  |
| 5 | Sæt begge kalender datoer så de viser et tidsrum efter seneste løbetur fandt sted. | Seneste løbetur forsvinder fra oversigt. |  |
| 6 | Sæt begge kalender datoer så seneste løbetur vises. | Mindst en løbetur vises i oversigt. |  |
| 7 | Sæt begge kalender til den dato seneste løbetur fandt sted. | Seneste løbetur vises stadig i oversigt. |  |

**5. Informationsside test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Log ind på FunRun. | System viser oversigtsside. |  |
| 2 | (Hvis ingen løbeture i øversigt)  Lav en løbetur så mindst en vises. | Mindst en løbetur vises i oversigt. |  |
| 3 | Tryk på en løbetur i oversigten. | System viser informationsside. |  |
| 4 | Kontrollér at dato øverst stemmer overens med dato på oversigtsside. | Ingen ændring. |  |
| 5 | Kontrollér at de tre felter øverst på siden viser data. | Ingen ændring. |  |
| 6 | Kontrollér at kortet viser løbetur, samt start og slut punkter. | Ingen ændring. |  |
| 7 | Kontrollér at kortet starter centreret på løbeturen, og at hele løbeturen kan ses når siden hentes. | Ingen ændring. |  |
| 8 | Kontrollér at begge diagrammer under kortet viser data fra turen. | Ingen ændring. |  |
| 9 | Tryk på ”Tilbage”-knappen. | System viser oversigtsside. |  |
| 10 | Tryk på ”Ny løbetur”-knappen | System viser ”Ny løbetur”-side.  Kort opdaterer position periodisk. |  |
| 11 | Tryk på ”Start løbetur”-knappen. | Knap bliver grå og ændrer tekst. |  |
| 12 | Naviger tilbage via systemnavigation før knappen bliver rød og ændrer tekst. | System viser oversigtsside. |  |
| 13 | Tryk på seneste løbetur i oversigten. | System viser informationsside.  Overskriften læser ”Ingen punktdata” |  |
| 14 | (Hvis overskriften læser en dato har turen nået at generere et eller flere punkter) gentag trin 9 til 13. |  |  |

**6. Slet løbetur test**

Platform: web, app

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Log ind på FunRun. | System viser oversigtsside. |  |
| 2 | (Hvis ingen løbeture i øversigt)  Lav en løbetur så mindst en vises. | Mindst en løbetur vises i oversigt. |  |
| 3 | Tryk på en løbeturs røde ”Slet”-knap. | Efter et kort øjeblik genhentes listen uden løbeturen. |  |

**7. Informationstyveri test**

Platform: web

Tester:

Version:

|  |  |  |  |
| --- | --- | --- | --- |
| **Trin** | **Instruktion** | **Forventet resultat** | **Note/fejl** |
| 1 | Log ind på FunRun. | System viser oversigtsside. |  |
| 2 | (Hvis ingen løbeture i øversigt)  Lav en løbetur så mindst en vises. | Mindst en løbetur vises i oversigt. |  |
| 3 | Tryk på en løbetur i oversigten. | System viser informationsside. |  |
| 4 | Notér tallet til sidst i sidens adresse i noten til dette trin. | Ingen ændring. |  |
|  | Tryk på ”Tilbage”-knappen. | System viser oversigtsside. |  |
|  | Tryk på ”Log ud”-knappen. | System viser loginside. |  |
|  | Indtast brugernavn: FunRunTestingUnauthorized | Ingen ændring. |  |
|  | Indtast kodeord:  tQeWsEtRiTnYg5 | Ingen ændring. |  |
|  | Tryk på ”Log ind”-knappen. | System viser oversigtsside. |  |
|  | Naviger til informationssiden for den anden brugers løbetur (udskift ”runs” i adressen med ”run/[tallet du noterede]”). | System viser informationsside.  Overskriften læser ”Ugyldigt runId!” efter indlæsning. |  |

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)