

***FunRun’’’’’***

Procesrapport

**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.

Jeg beskriver projektets case og problemformulering, og redegør for hvordan jeg planlægger at gennemføre projektet, samt hvordan jeg endte med at afvige fra denne plan, og reflekterer over grundene til dette.

Jeg fortæller om de metoder og teknologier jeg har valgt at bruge til projektet, hvilke jeg kunne have valgt i stedet, og hvorfor jeg har valgt dem fra.

Til slut diskuterer jeg hvilke funktioner jeg har valgt fra i systemet for at inkludere andre, og konkluderer på projektforløbet, hvad jeg har lært, og hvad jeg tænker om det som helhed.

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

Indholdsfortegnelse

[Læsevejledning 5](#_Toc100658350)

[Indledning 5](#_Toc100658351)

[Case beskrivelse 6](#_Toc100658352)

[Problemformulering 6](#_Toc100658353)

[Estimeret Tidsplan 7](#_Toc100658354)

[Metode- og teknologivalg 8](#_Toc100658355)

[API 8](#_Toc100658356)

[ASP.NET, Node.js, Rust, og PHP 8](#_Toc100658357)

[bcrypt, SHA-family, Argon2 8](#_Toc100658358)

[Database 8](#_Toc100658359)

[Dokument- vs. relationelle databaser 9](#_Toc100658360)

[SQLite 9](#_Toc100658361)

[Entity Framework 9](#_Toc100658362)

[PWA 9](#_Toc100658363)

[Native App, PWA, WPF 9](#_Toc100658364)

[Angular, Blazor, React, og Vue.js 9](#_Toc100658365)

[Versionsstyring 10](#_Toc100658366)

[Dropbox, Git, Subversion 10](#_Toc100658367)

[IDE’er og hjælpeprogrammer 11](#_Toc100658368)

[Visual Studio 2022 11](#_Toc100658369)

[Swagger 11](#_Toc100658370)

[Conveyor 11](#_Toc100658371)

[Visual Studio Code 11](#_Toc100658372)

[Firebase 11](#_Toc100658373)

[Microsoft SQL Server Management Studio 11](#_Toc100658374)

[GitKraken 12](#_Toc100658375)

[Hosting 12](#_Toc100658376)

[Væsentlige elementer fra produktrapporten 13](#_Toc100658377)

[Realiseret Tidsplan 13](#_Toc100658378)

[Konklusion 14](#_Toc100658379)

[Kildeliste 15](#_Toc100658380)

# Læsevejledning

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

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 produktrapportens teknologiafsnit.

# 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 i denne rapport uddyber jeg den proces jeg gik igennem for at når frem til mit endelige produkt.

# Case beskrivelse

Løb er en kæmpe hobby. Ud over at være en god måde at få motion på i vores i stigende grad sedentære hverdag, er der flere og flere der tager det meget seriøst på hobbyplan.

I denne hektiske hverdag kan det være en udfordring at holde styr på både job, hus, samleverske, børn, og så ovenikøbet ens løberuter. Derfor er der behov for et system der er letvægtigt som kan gemme de ruter man løber uden at tvinge en bruger til at have flere enheder på sig end højest nødvendigt.

Andre problemer for løbere kan være at de taber motivationen, og mister overblikket over hvad de får ud af deres træning, eller at de igen og igen udskyder at komme i gang, fordi de ikke har nogen måde at overvåge deres fremskridt på.

# Problemformulering

Hvordan kan jeg opbevare en brugers løbedata, og præsentere dem med brugbar statistik og logning?

# Estimeret Tidsplan

Min estimerede tidsplan er vedlagt denne rapport som bilag 1.

Et valg jeg har taget i forhold til formatering, er at inkludere weekender og helligdage, i tilfælde af at jeg skulle vælge at arbejde på disse, kan man holde estimeret og realiseret tidsplan op ved siden af hinanden og sammenligne, i modsætning til hvis jeg kun havde indført dem på den realiserede.

Min plan for projektet er at angribe de formalia i forhold til rapport der skal gøres færdige og fastlægges inden for første uge først, det værende case beskrivelse, problemformulering, tidsplan, og kravspecifikation.

Efterfølgende vil jeg gerne fordybe mig i projektet og opbygge et minimum viable product, som i mit projekts tilfælde vil betyde et API med tilhørende database, som kan lagre data fra en PWA. Når der er hul igennem for den kommunikation, er den mest komplicerede del af projektet i min optik overstået.

Efter minimum viable product er opnået, vil jeg vende tilbage til rapporterne med en god ide om hvordan mit projekt er opbygget, for at skrive de store metode- og teknologi-afsnit i begge rapporter, samt afsnittet om mit projekts arkitektur.

Efter dette vil jeg atter vende tilbage til projektet for at finpudse kommunikationen mellem PWA og API, og udvide PWA’ens funktionalitet. Det er her jeg planlægger at kode fremvisning af den data jeg programmerede opsamling og lagring af i første omgang.

Sidst men ikke mindst vil jeg vende tilbage til rapporterne for at skrive brugervejledninger til nyligt implementeret funktionalitet, og derefter finpudse og skrive de sidste afsnit, navnlig forord, læsevejledninger, og konklusion. Det er også over påsken jeg nok vil få mine bekendte og allierede til at læse mine rapporter igennem og hjælpe mig med korrektur.

Jeg har valgt at sætte rigeligt tid af til hver opgave, som er noget jeg som regel gør når jeg planlægger, ligegyldigt formålet, så min tanke er at jeg nok kommer hurtigere igennem nogen opgaver end jeg har anført, i hvilket tilfælde jeg tænker at mit projekt har områder nok jeg kan tage fat i og udvide, skulle jeg have tid til det.

# Metode- og teknologivalg

I dette afsnit vil jeg begrunde hvorfor jeg har valgt de forskellige teknologier som projektet benytter.

## API

Jeg har valgt at bruge et API til mit projekt som bindeled mellem min database og PWA, da de ellers ville være nødsaget til at køre på samme enhed. Jeg bruger et ASP.NET webAPI fordi det er noget jeg har kendskab til, og man får enormt meget foræret i forhold til opsætning så man hurtigt kan komme i gang.

### ASP.NET, Node.js, Rust, og PHP

I stedet for ASP.NET kunne man have brugt et antal af forskellige andre teknologier til at opsætte et API.

Node.js er et open-source alternativ, hvor man kan eksekvere JavaScript serverside, og man kunne derfor have valgt at holde sig til mindre sprog på tværs af projektet. Node.js er også i særdeleshed interessant hvis man arbejder med en microservice arkitektur[[1]](#footnote-1).

Man kunne have valgt at bruge Rust, som flere og flere gør. Rust er hurtigere end ASP.NET og på visse punkter nemmere at arbejde med[[2]](#footnote-2).

Man kunne også have brugt PHP til at arbejde med databasen. PHP er ikke helt så hurtigt som ASP.NET, og egner sig dårligere til store virksomheder og systemer[[3]](#footnote-3).

### bcrypt, SHA-family, Argon2

Til hashing af passwords i databasen bruger jeg bcrypt, som er en af de sikreste måder at hashe passwords på.

Man kunne have brugt en af algoritmerne fra SHA-familien (SHA1, SHA256, eller SHA512) til at hashe passwords, men problemet er at de alle er ’hurtige’ algoritmer, mens bcrypt er ’langsom’. Det at en algoritme er hurtig er godt på mange tidspunkter, men lige når det kommer til passwords gør det den nemmere at knække og derfor usikker[[4]](#footnote-4).

Ellers kunne man have brugt Argon2, som er sikrere end bcrypt når det kommer til nogen ret specifikke typer angreb[[5]](#footnote-5).

## Database

Mit projekt benytter en MS-SQL database som er sat op via Entity Framework ud fra code-first princippet.

Jeg har valgt MS-SQL fordi den passer godt sammen med mit API, og det er igen noget jeg har kendskab til.

### Dokument- vs. relationelle databaser

Man kunne have brugt en dokumentdatabase i stedet for en relationel database. En dokumentdatabase er typisk hurtigere end en relationel, og bedre egnet til at arbejde med big data, men opbevarer ikke dataene i tabeller på samme måde. En dokumentdatabase har heller ikke indbyggede relationer.

Det er min opfattelse at dokumentdatabaser er rigtig gode til det de er lavet til, men lige så snart man skal ud i at bruge relationer er man bedre tjent med en relationel database, frem for at kode relationerne manuelt.

### SQLite

Man kunne også have lavet projektet med en SQLite database, som er en form for lille database der kan køre lokalt, uden en server. SQLite er typisk langsommere end en fuld database, og bytter også en del andre funktioner for sin evne til at køre separat.

Da filosofien bag mit system er at brugeren bør kunne logge ind og tilgå deres data hvor som helst, er SQLite ilde egnet.

### Entity Framework

Jeg har valgt at sætte min database op via Entity Framework, som er Microsofts bud på et Object Relational Mapping tool, eller ORM. Det vil sige at jeg kan arbejde med og lagre data i objekter i stedet for tabeller, og det gør dataene nemmere at arbejde med i koden. Ydermere, gør det at det er code-first at man undgår eventuelle brugerfejl i opsætningen og sammenkoblingen af en traditionel database som man sætter sammen med et API manuelt[[6]](#footnote-6).

## PWA

Jeg har valgt at lave en PWA til mit projekt. En Progressive Web App kan tilgås og installeres fra nettet på en bred vifte af enheder, og det er en nem måde at opnå en cross-platform løsning på.

### Native App, PWA, WPF

I stedet for en PWA kunne man have lavet en native app. Med en native app skulle man have begrænset sig i forhold til platform, og appen ville ikke længere være tilgængelig i en browser heller. En native app ville køre hurtigere end en PWA, på bekostning af at være en del sværere at sætte op. Native apps kan også tilgå funktioner på visse enhedskomponenter som PWA endnu ikke har adgang til, skønt denne funktionalitet er voksende hos PWA[[7]](#footnote-7).

Hvis jeg ikke skulle have lavet en app, kunne jeg i stedet have lavet en WPF applikation til desktop. WPF tager over for Windows Forms, og har en række ting der er blevet strømlinet og gjort nemmere, men jeg følte at det ville blive svært at forsvare ikke at bruge en lokal database, hvis det kørte som WPF, i hvilket tilfælde det ikke ville opfylde alle kravene. Derfor valgte jeg i stedet at lave en App. Med en WPF applikation skulle jeg også have fundet en anden måde at få data på, da en typisk computer ikke har en GPS indbygget. Svaret på dette kunne måske have været en embedded enhed, og så havde projektet set anderledes ud.

### Angular, Blazor, React, og Vue.js

Min PWA er programmeret i et framework der hedder Vue.js. Jeg har arbejdet med Vue.js før, og jeg finder det intuitivt.

Angular, som Vue.js er baseret på, ligger lige til højrebenet at nævne som et alternativ. Angular har dog et par ting der gør det bedre egnet til store systemer end Vue.js. For det første er det sværere at arbejde i og gå til, og for det andet er det langsommere end Vue.js. Desuden er der mange ting der skal gentages rundt omkring i koden, som også bidrager til at jeg vælger det fra[[8]](#footnote-8).

Blazor er også et alternativ, hvor Node.js kan køre JavaScript på serversiden og på den måde gøre at meget af koden ville være et og samme sprog, kan Blazor gøre lidt det samme, men modsat, så du kan kode din PWA i C#, og så på den måde opnå kodeensartethed på tværs af et projekt. Hvis jeg havde arbejdet i Blazor, ville mit projekt være meget ensartet at se på, og jeg vil hellere demonstrere et bredere kendskab[[9]](#footnote-9).

Man kunne også have skrevet i React. React og Vue.js ligner meget hinanden, og mange rigtig store hjemmesider og services bruger React. React burger dog en sværere syntax, og i React programmere du hele din hjemmeside eller applikation i JavaScript, hvor du i Vue.js kan dele det op i HTML, CSS, og JavaScript[[10]](#footnote-10).

## Versionsstyring

Versionsstyring er vigtigt. Lige så snart du er oppe og arbejde med et moderat komplekst system er det vigtigt at kunne gå tilbage i historikken hvis problemer skulle opstå.

Jeg har valgt at bruge Git til versionsstyring for mit system, selvom det ikke er det jeg har arbejdet mest med, da jeg mener at Git er fremtiden indenfor versionsstyring.

### Dropbox, Git, Subversion

I stedet for Git, kunne man have valgt at bruge Subversion, Git’s forgænger og det system jeg er mest fortrolig med. Git har nogen fordele over Subversion, som for eksempel at man kan committe lokalt uden at pushe til respository før man er klar, kontra at man i Subversion der nød til at pushe til repository hver gang man vil comitte. Ud over dette, er Git også bedre egnet til at arbejde med branches end Subversion. Et eller andet sted er forskellen på de to ikke så væsentlig i et enmandsprojekt på den her størrelse, men jeg så muligheden for at blive mere fortrolig med Git, og jeg greb den[[11]](#footnote-11).

Ydermere har GitHub en funktionalitet, GitHub Actions, som gør det muligt at sætte services op med Continouos Integration og Deployment. Begge mine services er konfigureret til at builde og deploye automatisk når jeg pusher til main, så ved tryk på en knap kan jeg ti minutter senere have ændringer ude som jeg kan teste på min enhed.

Jeg vil også nævne at Dropbox har 30 dages historik på alle filer, så man kunne til nød bruge det som et kortsigtet alternativ. Dropbox har dog ikke en brugervenlig måde at gendanne flere filer på en gang til en tidligere version, og ej heller har de en struktureret måde at sikre at alt kan bygge når det bliver comittet, så det ville være en mangelfuld afløser for enhver der er vandt til at arbejde med Git.

## IDE’er og hjælpeprogrammer

Gennem projektet har jeg brugt en serie af forskellige værktøjer til at udvikle mit projekt. Disse vil jeg begrunde i dette afsnit.

### Visual Studio 2022

Jeg har valgt at bruge Visual Studio 2022 til at udvikle mit API, da jeg arbejder i ASP.NET og begge er udviklet og vedligeholdt af Microsoft. Man kunne have bevæget sig op og ned i kompleksitet for ens IDE ved at vælge andre, men til udvikling a et funktionelt API har Visual Studio hverken mere eller mindre end jeg behøver. Ville man bevæge sig ned i kompleksitet kunne man vælge Visual Studio Code, som er mere letvægtigt.

### Swagger

Jeg har Swagger indbygget i mit API, som gør mig i stand til at teste alle mine endpoints uden behov for et tredjepartsprogram. Swagger er smart fordi det også giver dig meget information om dine endpoints, som for eksempel at den foreslår syntax til requests, hvor man så bare kan fylde ind. Med swagger havde jeg også mulighed for at enable authentication på siden, så jeg kunne kalde mit login endpoint for at få en JWT token, tilføje den til siden, og så uden problemer kalde og teste de endpoints der forventede sådan en.

Ellers kunne man have testet sine endpoints med Postman, som jeg har gjort før, men efter at arbejde med swagger tror jeg aldrig jeg vil bruge postman igen, i hvert fald ikke hvis jeg har valget om at få swagger indbygget fra starten. I Postman kan du skrive requests og sende til dine endpoints. Resultatet er det samme som swagger, men du har flere muligheder for at begå fejl, for eksempel ved at sende det forkerte format eller forkert syntax.

### Conveyor

Til testning af min app på en smart enhed, til at gøre mit API tilgængeligt over nettet når jeg kører det på min maskine, har jeg brugt en Visual Studio extension der hedder Conveyor. Conveyor gør det meget nemt at tilgå ens services på denne måde, primært med fokus på at teste dem.

### Visual Studio Code

Til at udvikle min PWA har jeg brugt Visual Studio Code. Man kunne også have brugt Visual Studio, men jeg kan godt lide at bruge det mest letvægtige jeg kan til en given opgave, og Visual Studio har ikke rigtig nogen funktioner jeg skal bruge til at udvikle en PWA.

Visual Studio Code er et letvægtigt alternativ til Visual Studio, og det er noget nemmere at finde rundt i. Det er også nemt at finde og konfigurere extensions i, og jeg bruger det meget når jeg koder i min fritid.

### Firebase

Fordi min PWA ikke kan køres gennem Visual Studio som man normalt kører ting, er Conveyor ikke en løsning når jeg vil teste den på en smart enhed. For at gøre min PWA tilgængelig online til test har jeg brugt Firebase. Firebase er en Google service hvor man gratis kan oprette en bruger og deploye services, som derefter kan tilgås udefra.

### Microsoft SQL Server Management Studio

Til at inspicere min database efter oprettelse gennem Visual Studio har jeg brugt Microsoft SQL Server Management Studio. Da det er en MS-SQL database, og Microsoft SQL Server Management Studio er den tilknyttede IDE, gav det mening og herigennem kunne jeg inspicere tabeller og kolonner i grafisk interface, og kontrollere at det hele så ud som jeg havde planlagt.

### GitKraken

Til at interagere med Git og mit repository på Github.com har jeg brugt GitKraken. Der er mange forskellige programmer til at bruge Git, for eksempel GitHub Desktop og Tortoise Git, men for mig at se kan de stort set det samme, så jeg valgte en jeg vidste var populær og som jeg har set andre bruge.

Jeg har ikke gjort meget andet i selve GitKraken end at pushe til GitHub, og til det har det virket fortrinligt.

## Hosting

Til endelig hosting af min løsning har jeg valgt at bruge Microsoft Azure.

Microsft Azure er den næststørste Cloud Computing platform, næst efter Amazon Web Services, men den hurtigst voksende[[12]](#footnote-12).

Jeg valgte at bruge Microsft Azure fordi den er nyere end Amazon Web Services, jeg har hørt mere positivt om den, og som endnu et Microsoft produkt kan jeg publishe til den direkte fra Visual Studio uden at skulle døje med extensions eller lignende. Jeg kunne også bruge Microsoft SQL Server Management Studio til at konfigurere firewall på min SQL database på Microsft Azure, som er endnu en ting jeg ikke nødvendigvis ved var gået gnidningsfrit på Amazon Web Services.

Microsoft Azure tillod mig også at sætte Continouos Integration/Deployment op på begge dele af projektet med GitHub actions. Jeg måtte lære at bruge Azure da jeg nåede til hosting, men havde jeg vidst hvor nemt det ville være når det var oppe at køre, havde jeg gjort det som det allerførste, og sparet mig selv at bruge Conveyor og Firebase til testing. Den eneste ulempe ved Microsoft Azure i den sammenhæng er at den er noget længere om at deploye, hvor Conveyor og Firebase tilsammen er næsten øjeblikkelige, så om jeg i virkeligheden havde sparet tid og besvær på den måde er lidt svært at sige. Når jeg siger det var nemt, skal det også siges at det tog et tocifret antal timer over flere dage, men de timer kunne lige så godt have været givet ud i starten, som lidt over halvvejs.

Alternativet til at bruge en Cloud Computing platform, kunne også være en traditionel server. Hvis jeg havde en fysisk server og et domæne til rådighed kunne jeg lave mit API og min PWA til Docker images, som jeg så kunne køre på serveren, og derefter route trafikken mod et subdomæne til exposede porte på containerne. Derved ville man kunne tilgå begge services udefra via subdomæner.

En traditionel server blev ikke, trods forespørgsel, stillet til rådighed for dette projekt.

# Væsentlige elementer fra produktrapporten

Jeg vil vælge at fremhæve afsnittet Overordnet Arkitektur fra Produktrapporten som repræsentativt af det der er kodemæssigt interessant ved projektet.

Ydermere, hvis der er forvirring eller spørgsmål til hvordan systemet fungerer, er brugervejledningerne sidst i produktrapporten skrevet til nye brugere af systemet, og de beskriver i detaljer hvordan man gør alt fra at installere appen til at oprette, vise, og slette løbeture.

# Realiseret Tidsplan

Min realiserede tidsplan er lagt ved denne rapport som bilag 2.

Mit projekt har mere eller mindre fulgt den fremgangsmåde, hvis ikke helt den tidsplan, som jeg lagde ud i starten.

Ligesom jeg forudsagde, tog mange opgaver mindre tid and jeg havde planlagt, hvilket gav mig tid til et længere mere koncentreret programmeringsforløb i midten, som gjorde at mit produkt blev bedre og pænere end jeg havde turde håbe at det ville.

Hvis jeg skulle planlægge om, ville jeg granulere mine overskrifter i tidsplanen yderligere, selvom jeg synes at jeg gjorde mig umage med det. Der er nogen ting der kom til at gå ind under overskrifter hvor de ikke helt passer på min realiserede tidsplan, som for eksempel at deploye services til Azure, som gik ind under opsætning siden jeg ikke havde et punkt til det.

Helt konkret var det fjollet at planlægge at skrive afsnittet Overordnet Arkitektur før halvdelen af arkitekturen var planlagt at være færdig.

Som sagt da jeg estimerede min tidsplan, så jeg det mest som en beskrivelse af min fremgangsmåde, og en guide for hvornår ting burde være færdige, og efter den intention har jeg fulgt den ret godt.

Mine logbog med detaljerede beskrivelser af hvad jeg har lavet dag for dag, er vedlagt som bilag 3.

# Konklusion

Gennem projektet har jeg udviklet en proof-of-concept løsning på det problem jeg lagde ud i min case beskrivelse og problemformulering, og jeg føler at systemet besvarer dem mere en tilfredsstillende.

Systemet jeg kom frem til benytter et API med tilhørende database, som en PWA kan udveksle Information med, deriblandt positionsdata fra enhedens GPS. Denne information bliver lagret i databasen, og kan senere ses af brugeren.

Projekter som disse ser jeg som en kæmpe læringsmulighed, og jeg har arbejdet med ting som jeg har kendskab til gennem min undervisning, fordi jeg til hverdag arbejder med kode på en måde der ikke egner sig godt til besvarelse af de krav der er stillet til dette projekt, men ud over disse har jeg også stiftet bekendtskab med teknologier og udfordringer jeg ikke ser i min hverdag, som for eksempel Git og Microsoft Azure, som jeg i den grad kan tage med mig videre.

Jeg går til enhver mulighed for at arbejde på tværs a flere teknologier med iver, for det er noget jeg synes er enormt spændende, og desværre ikke noget jeg laver meget af i min hverdag.

Det har været et stressfuldt og hårdt forløb, men jeg er stolt af resultatet.

Hvis jeg skulle arbejde videre med projektet har jeg nogen ideer funktionalitetsudvidelser jeg ville starte med;

Hvis systemet skulle bruges i den virkelige verden skulle der nok være en måde at give ture navne på, så man kunne genkende dem senere. Det ville dog kræve at hele siden der viser løbeture skulle gendesignes, da der på nogen enheder i forvejen ikke er plads til meget mere på skærmen.

Det kunne også være fedt at kunne ændre passwords, men for det skulle jeg have en måde at verificere at en bruger er hvem de er, så jeg ville skulle tilknytte enten e-mail eller telefonnummer til kontoerne.

Ellers ville det være fedt at kunne fremvise noget længere sigtet statistik over hvordan en løber har forbedret sig, for eksempel et skema der viser topfart på tværs af ture eller lign. Det åbenlyse i forhold til det her projekt er bare at vi ikke har så lang tid, specielt ikke til indsamling af data hvis systemet først skal opbygges, så jeg har valgt at bruge krudtet på ting der bedre kan vises frem, og på at få det hele til at køre så godt og stabilt som muligt.

Disse er ting der kunne inkluderes i projektet fremadrettet, men som det står synes jeg allerede det indeholder og demonstrerer meget forskellig funktionalitet.

# Kildeliste

Difference Between Node.js and Asp.net

<https://www.geeksforgeeks.org/difference-between-node-js-and-asp-net/>

(Sidst besøgt 12/04/2022)

Rust Language Gains Traction in .NET Community

<https://visualstudiomagazine.com/articles/2021/05/03/net-rust.aspx>

(Sidst besøgt 12/04/2022)

PHP Vs ASP.NET: How to Choose the Right One?

<https://www.pixelcrayons.com/blog/php-vs-asp-net-how-to-choose-the-right-one/>

(Sidst besøgt 12/04/2022)

What is the difference between bcrypt and SHA256?

<https://rietta.com/blog/bcrypt-not-sha-for-passwords/>

(Sidst besøgt 12/04/2022)

Password Hashing: Scrypt, Bcrypt and ARGON2

<https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e>

(Sidst besøgt 12/04/2022)

What is Entity Framework?

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

(Sidst besøgt 12/04/2022)

What are Progressive Web Apps?

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

(Sidst besøgt 12/04/2022)

Angular vs Vue: Which Framework to Choose in 2022?

<https://www.simform.com/blog/angular-vs-vue/>

(Sidst besøgt 12/04/2022)

Blazor vs Vue

<https://www.telerik.com/blogs/blazor-vs-vue-web-developers>

(Sidst besøgt 12/04/2022)

Vue.js vs React: Comparison of Two Most Popular JS Frameworks

<https://www.codica.com/blog/react-vs-vue/>

(Sidst besøgt 12/04/2022)

SVN vs Git: Which One Is Best for Your Needs?

<https://blog.hackbrightacademy.com/blog/svn-vs-git/>

(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

Estimeret Tidsplan

Chart

Description automatically generated

Bilag 2

Realiseret Tidsplan

Placeholder

1. <https://www.geeksforgeeks.org/difference-between-node-js-and-asp-net/> [↑](#footnote-ref-1)
2. <https://visualstudiomagazine.com/articles/2021/05/03/net-rust.aspx> [↑](#footnote-ref-2)
3. <https://www.pixelcrayons.com/blog/php-vs-asp-net-how-to-choose-the-right-one/> [↑](#footnote-ref-3)
4. <https://rietta.com/blog/bcrypt-not-sha-for-passwords/> [↑](#footnote-ref-4)
5. <https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e> [↑](#footnote-ref-5)
6. <https://www.entityframeworktutorial.net/what-is-entityframework.aspx> [↑](#footnote-ref-6)
7. <https://web.dev/what-are-pwas/> [↑](#footnote-ref-7)
8. <https://www.simform.com/blog/angular-vs-vue/> [↑](#footnote-ref-8)
9. <https://www.telerik.com/blogs/blazor-vs-vue-web-developers> [↑](#footnote-ref-9)
10. <https://www.codica.com/blog/react-vs-vue/> [↑](#footnote-ref-10)
11. <https://blog.hackbrightacademy.com/blog/svn-vs-git/> [↑](#footnote-ref-11)
12. <https://www.simplilearn.com/tutorials/azure-tutorial/what-is-azure> [↑](#footnote-ref-12)