Forside

Titelblad

Forord

# Indholdsfortegnelse

# Læsevejliedning

# 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

Pin tidsplan er lagt ved 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 først 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 tilfælde vil betyde et API med tilhørende database, som kan kommunikere frem og tilbage med 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 metodeafsnit 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.

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 min kæreste til at læse mine rapporter igennem og hjælpe mig med korrektur på dem.

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 at bruge 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 have skullet 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.

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

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

Node.js er et open-source alternativ, hvor man kan eksekvere JavaScript serverside, og man kunne derfor have valgt det og holdt 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 og mening 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 på kodesiden. 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.

### 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[[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.

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

Min PWA er programmeret i et framework der hedder Vue.js. Vue er noget jeg har arbejdet med før, og jeg synes at det er 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 den samme 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 respositoriet 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 lære mere om at arbejde med Git, og jeg greb den[[11]](#footnote-11).

Jeg ville også lige 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 på, 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, som jeg vil 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, hvor jeg kunne tilføje min JWT token og så kalde 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 at gøre mit API tilgængeligt over nettet når jeg kører det på min maskine, bruger jeg 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. Jeg har valgt at bruge Conveyor som proof-of-concept af min fulde løsning, da jeg ikke har hverken et domæne eller en offentlig IP-adresse til rådighed, og skolen heller ikke har.

Hvis jeg havde en server og et domæne til rådighed ville jeg lave mit API til et Docker image, som jeg så kunne køre på min server, og derefter route trafikken mod et subdomæne til en exposed port på den container. Derved ville man kunne tilgå mit API udefra via det subdomæne.

### Firebase

Fordi min PWA ikke kan køres gennem Visual Studio som man normalt kører ting, er Conveyor ikke en løsning. For at gøre min PWA tilgængelig online bruger jeg Firebase. Firebase er en google service hvor man gratis kan oprette en bruger og deploye services, som derefter kan tilgås udefra.

Hvis jeg havde en server og et domæne til rådighed, ville jeg bruge Docker til at gøre min PWA tilgængelig udefra som forklaret i det foregående afsnit.

### 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.

### 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 projekt 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.

# Væsentlige elementer fra produktrapporten

# Afgrænsning?

Navngive ture?

Log ud?

Mere præcis logning/milis i javascript?

Gem token i cookie

Forever login like google, ask tobias (refresh token)

# Realiseret Tidsplan

Lavede mere projekt før jeg skrev overordnet arkitektur.

# Konklusion

# Kildeliste

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)