|  |
| --- |
| Daniel Simonsen  01-12-2023 |

|  |
| --- |
|  |
| Produktrapport |
| CitizenTaxi |



|  |
| --- |
| **Elev:**  Daniel Simonsen |
| **Firma:**  Skoleoplæringscenter |
| **Projekt:**  CitizenTaxi |
| **Uddannelse:**  Datatekniker m. speciale i programmering |
| **Projektperiode:**  06/11/2023 – 07/12/2023 |
| **Afleveringsdato:**  01/12/2023 |
| **Fremlæggelsesdato:**  07/12/2023 |
| **Vejledere:**  Frank Rosbak  Lars Thise Pedersen  Per Madsen |
|  |

# Titelblad



Techcollege Aalborg,

Struervej 70,

9220 Aalborg

# Læsevejledning

Jeg anbefaler at læse procesrapporten først.

// TODO: Ord-/fortkortelsesliste

Indholds fortegnelse

[Titelblad i](#_Toc150439311)

[Læsevejledning 1](#_Toc150439312)

[Indledning 3](#_Toc150439313)

[Case beskrivelse 3](#_Toc150439314)

[Problemformulering 3](#_Toc150439315)

[Afgrænsning 3](#_Toc150439316)

[Kravsspecifikation // TODO: Update 4](#_Toc150439317)

[(Testspecifikation) 5](#_Toc150439318)

[Produktdokumentation 6](#_Toc150439319)

[**Databasediagram** 6](#_Toc150439320)

[**Programarkitektur** 6](#_Toc150439321)

[**Klassediagram** 6](#_Toc150439322)

[**Sikkerhed** 6](#_Toc150439323)

[**Test** 6](#_Toc150439324)

[Brugervejledning 7](#_Toc150439325)

[Bilag // TODO 8](#_Toc150439326)

# Indledning

Borgerne i Region Nordjylland klager over den lange ventetid, når de ringer til Flexsygehustaxa. De vælger derfor at ringe til lægesekretærerne, og bede dem om at bestille en taxa for borgeren.

Lægesekretærerne kan derefter ringe via internt nummer og få taxaen booket til borgeren hurtigere, end hvis borgeren selv havde ringet, dog tilføjer det endnu en opgave hos lægesekreterne, som kunne have været undgået, hvis borgeren selv havde bestilt taxaen.

# Case beskrivelse

## Problemformulering

Hvordan kan man få borgerne til selv at bestille deres Flexsygehustaxa uden hjælp fra lægesekretærerne og uden den unødvendige lange ventetid, og kan man ellers formindske lægesekretærernes ekstra arbejde?

## Afgrænsning

For at gennemføre projektet som en prototype løsning til Flexsygehustaxas lange ventetid, afgrænser jeg bl.a. sygehus adgang (f.eks. bruge deres eksisterende login oplysninger) og rigtig taxa bestilling (hvis de har et offentligt api til rådighed).

Produktet har egne brugere og simulation af taxabestilling af denne årsag.

Derudover har jeg nogle krav på min kravsspecifikation, som jeg også blev nødt til at afgrænse grundet tid, som f.eks. kritiske beskeder på eksterne services og Google Maps support.

# Kravsspecifikation

Min kravsspecifikation er sorteret efter prioritet, så det er let at overskue, hvilke krav er mest vigtige for projektet.

A close-up of a text

Description automatically generated

Som vist på billedet, har jeg opfyldt alle mine **skal**-krav, halvdelen af mine **burde**-krav, ca. 75% af mine Kunne krav, og en kvartdel af mine **gerne**-/**hvis tid**-krav. Desværre giver det en hel score på 70% af mine krav, der er opfyldt, og selvom mange af mine uopfyldte krav omhandler brugeroplevelse og ikke projektets hovedfunktionalitet, ville jeg stadig have ønsket, at jeg kunne have nået at opfylde nogle flere krav, og evt. komme op på en samlet score på i hvert fald 80%.

Havde jeg haft f.eks. nogle dage mere til projektet, ville jeg nok have fuldført alle mine krav, undtagen Front3, Live opdatering efter bestilling, da jeg ikke kunne få SignalR til at sende notifikationer til min frontend via kørende timer og callbacks i min backend.

Selvom jeg ikke har opfyldt alle mine krav, har jeg stadig fået mig et funktionelt produkt, der sagtens kunne bruges i det virkelige liv som beta program.

# Testspecifikation

**Unit- og integrationstests**

Jeg har implementeret unit- og integrationstests i min backend, hvor jeg f.eks. tester mine Repository klasser (DataAccessTests), min LoginService (BusinessTests) og mine API controllere (ApiTests).

For at have et funktionelt projekt, skal man være sikker på, at man kan håndtere alle former for muligheder og håndtere dem korrekt. En unit- og integrationstest er ikke kun med til at sikre, at din ”unit/integration” fungerer som forventet, men er også med til at fortælle dig, at du har lavet en fejl i din implementering, hvis dine tests har virket tidligere.

Da min backend er lavet i C#, har jeg to muligheder for at unitteste mine Visual Studio projekter: NUnit og xUnit.

Begge unittest biblioteker fungerer på samme måde, i at man opretter en testklasse med nogle testmetoder annoteret med [Test] eller [Fact].

CitizenTaxi bruger NUnit, da NUnit tilbyder SetUp og TearDown metoder, hvilke fungerer som lifecycle funktioner for hver test. Denne funktionalitet bruges til at oprette en InMemory database til hver testmetode og sikre at samme database er slettet korrekt.

Billedet nedenunder illustrerer, hvordan SetUp og TearDown metoderne bruges i praksis. Billedet er fra DataAcecssTest projektet i [ABaseRepositoryTest](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Tests/DataAccessTests/ABaseRepositoryTest.cs#L38) filen.

A computer screen shot of a program code

Description automatically generated

Når man laver en unittest, er der oftest tale om 3 trin: Arrange, Act og Assert.

Arrange-delen er, hvor man arrangerer sine variabler, så de er klar til brug over Act.

Act-delen er, hvor man udfører selve testhandlingen. Her skal selve testen forekomme, hvor man tester alle mulige kombinationer, som kan indsættes og returneres af den testede funktion.

Assert-delen er, hvor man fortæller unittestværktøjet, hvad man forventer de returnerede variablers værdier indeholder baseret på de inputs, som den testede funktion har fået.

CitizenTaxi benytter af samme struktur med en evt. ekstra arrange lag efter act. Dette lag er med til at gøre assert koden mere læselig.

Eksempelvis i [ARepositoryTest.Add](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Tests/DataAccessTests/ABaseRepositoryTest.cs#L84) metoden, tester jeg mine repositories’ Add metode, og benytter af det ekstra arrange lag, for at gøre arrangekoden lettere læsligt.

A computer screen shot of a program code

Description automatically generated

Da jeg bruger OOP og nedarvninger i både mine repositories og controllers, har jeg oprettet ABaseRepositoryTest og ABaseControllerTest. Disse klasser indeholder det meste logik indenfor et repository og en controller, så jeg kan vedligeholde min kode DRY.

Læg mærke til at klasserne af A som præfiks, som fungerer ligesom interfaces med deres I præfiks. ”A” præfikset er med til at illustrere, at klassen er abstrakt og bør nedarves fra i en reel test som min [BookingsControllerTest](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/Tests/ApiTests/BookingsControllerTest.cs), der nedarver fra min [ABaseControllerTest](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/Tests/ApiTests/ABaseControllerTest.cs).

A computer screen shot of a program

Description automatically generated

**User- & Usabilitytest**

Da CitizenTaxi er inspireret af et reelt problem, som lægesekretærerne i Frederikshavn står over for, har jeg kontaktet 2 medarbejdere, som har udført en remote user-/usabilitytest af produktet.

**28. november**

Da jeg ikke er i stand til at observere mine testere hverken fysisk eller online, var der noget forvirring omkring opgaverne, jeg havde lavet til dem. Mine testere var også udsat for en input fejl, hvor hjemmesiden ikke registrerede den indsatte værdi, og nægtede testeren forbi næste trin.

Efter den feedback fik jeg fikset fejlen og blev lovet, at de ville prøve igen i morgen.

**29. november**

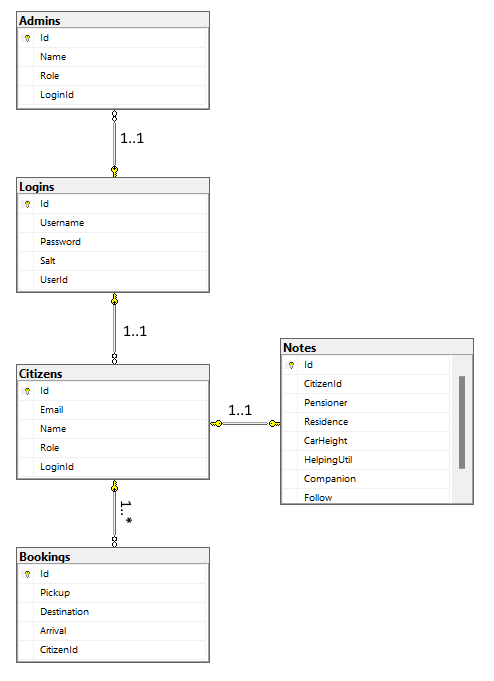
Jeg lavede 2 hovedopgaver til projektet – én som borger og én som lægesekretær. Efter en tester færdiggjorde borgeropgaven sprang testeren videre til lægesekretæropgaven og blev forvirret over, hvor testeren skulle trykke ”Opret borger” henne. Efter testeren kikkede nærmere, fandt testeren opretknappen, men fortalte at testeren aldrig ville have kikket der, da det var for langt væk fra hoved-contenten, testeren blev præsenteret for.

Jeg valgte derefter at flytte knappen tættere mod hoved-contenten og gjorde knappen større og derfor nemmere at få øje på.

Til sidst kom de 2 testere med noget feedback til både min testopgave men også ift. farvevalg af redigeringsknappen.

# Produktdokumentation

## **Databasediagram**



Databasediagrammet viser alle mine relationer mellem mine SQL-tabeller. Her indgår bl.a. at en borger har ét notat, men kan have flere bestillinger. Derudover er der selvfølgelig 1-til-1 forhold mellem bruger og login.

## **Programarkitektur**

Programarkitekturen består af min egen version af N-lagsstrukturen, hvor jeg har tilføjet et ”Common” lag og benytter af mit eget DanhoLibrary bibliotek, så mine lag ligner billedet nedenunder.

A diagram of a business

Description automatically generated

**Når en borger vil bestille en taxa**

For at en borger kan bestille en taxa, skal de logge ind på systemet først. Borgeren skal indtaste deres brugernavn og kodeord for at logge ind. Loginoplysningerne, burde i produktion, fås fra lægesekretærerne, da taxasystemet er et ”indelukket” system, der kun involverer borgere, der er berettige til gratis sygehustaxatransport.

En autoriseret borger får adgang til adskillige endpoints i API’et, som bl.a. deres eget notat og egne taxabestillinger.

Efter borgeren er logget ind, omstilles borgeren til borgersiden, hvor de bliver præsenteret for deres notat, deres bestillinger, og evt. hjælp til rettelse af notat eller oprettelse af taxabestilling.

A screenshot of a computer

Description automatically generated

**Eksisterende notat**

For at borgeren kan have adgang til gratis taxa, skal borgeren have et notat tilknyttet. Notatet er med til at hjælpe taxaservicen, til at finde og forberede den rette taxa samt service, for at borgeren kan føle sig tryg under turen.

Har borgeren *ikke* et notat tilknyttet, kan de ikke bestille en taxa. Notatet oprettes af lægesekretærerne og ideelt burde en borger ikke kunne eksistere i systemet, hvis de ikke har et notat.

I produktet slukkes der for bestillingsknappen, så borgeren ikke kan bestille en taxa.

A screenshot of a computer

Description automatically generated

**Formproces og validering**

For at undgå forvirring hos borgeren under bestillingsprocessen, er processen delt op i én side pr. spørgsmål. Hvert spørgsmål valideres, når borgeren trykker på Videre knappen.

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generatedA screenshot of a video chat

Description automatically generated

Hver gang borgeren trykker på ”Videre” og ”Afslut”, kaldes [onFormSubmit](https://github.com/DanielSimonsen90/CitizenTaxi/blob/e0def4f60fa4b343d31a4722e16c9b0399ddd66e/Projects/Frontend/src/components/pages/BookTaxi/BookTaxiLayout.tsx#L51) funktionen i [BookTaxiLayout](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Frontend/src/components/pages/BookTaxi/BookTaxiLayout.tsx).

Funktionen serialiserer formens data og tilføjer den nye data til den eksisterende data.

A screen shot of a computer code

Description automatically generated

Til sidst i funktionen sættes ”setSubmitted” til true, som trigger en update/change lifecycle event via [useUpdateEffect](https://github.com/DanielSimonsen90/CitizenTaxi/blob/e0def4f60fa4b343d31a4722e16c9b0399ddd66e/Projects/Frontend/src/components/pages/BookTaxi/BookTaxiLayout.tsx#L64) hooket.

Billede af useUpdateEffect hook koden ses på næste side.

A screen shot of a computer program

Description automatically generated

Hele hooket er med til at følge borgeren videre i bestillingsprocessen. Er formen invalid, får borgeren en notifikation. Har payloaden ikke en citizenId, for at sætte led mellem bestilling og borgeren, sættes det. Kan borgeren fortsætte i processen, gås der til næste trin. Til sidst, sendes en ”updateBooking” eller ”createBooking” action gennem [dispatch](https://github.com/DanielSimonsen90/CitizenTaxi/blob/e0def4f60fa4b343d31a4722e16c9b0399ddd66e/Projects/Frontend/src/components/pages/BookTaxi/BookTaxiLayout.tsx#L26) funktionen fra [useApiActions](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Frontend/src/hooks/useApiActions.ts#L12) hooket.

**Dispatching**

For at sende requests til API’et, er der lavet et action dispatching system. Den returnerede funktion fra useApiActions hooket tilbyder forskellige actions samt prædefinerede argumenter. Dispatch funktionen fungerer som en ”super-funktion”, der sender alle mulige requests til API’et. Hvis noget går galt, sendes en notifikation, ellers returneres dataen fra API’et.

A screen shot of a computer program

Description automatically generated

Når en updateBooking eller createBooking sendes til dispatch funktionen, fanges actionen i et kæmpe switch-statement. Actionen sendes til API’et dynamisk, og når responsdataen kommer tilbage fra API’et, opdateres borger provider state til det nyeste data.

Efter provideren opdateres, sendes en notifikation til borgeren om hvordan kaldet gik.

**Auth middleware**

Når hjemmesiden sender en request til API’et, går den først igennem [AuthMiddleware](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/Business/Middlewares/AuthMiddleware.cs) fra Business-laget. Middlewaren er med til at validere klientens autoritet. Klienten *skal* være logget ind på en valid konto og have en valid access-/refresh token i deres cookies, ellers er requesten invalid.

A screen shot of a computer program

Description automatically generated

Mere og dybere infromation omkring AuthMiddlewaren findes længere nede i rapporten.

**BookingController & BaseController**

Når klienten kommer igennem AuthMiddlewaren med POST request til endpoint /api/bookings, håndterer [BookingsControlleren](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/API/Controllers/BookingsController.cs) requesten via [CreateBooking](https://github.com/DanielSimonsen90/CitizenTaxi/blob/e0def4f60fa4b343d31a4722e16c9b0399ddd66e/Projects/Backend/API/Controllers/BookingsController.cs#L31C57-L31C57) metoden i API-laget.

A screen shot of a computer code

Description automatically generated

Efter null checks på citizen og citizen.Note, begynder den reelle oprettelsesproces via [BaseController](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/API/Controllers/BaseController.cs)’s [CreateEntity](https://github.com/DanielSimonsen90/CitizenTaxi/blob/e0def4f60fa4b343d31a4722e16c9b0399ddd66e/Projects/Backend/API/Controllers/BaseController.cs#L57) metode.

BaseController klassen nedarver fra Microsoft.AspNetCore.Mvc.ControllerBase, ligesom en normal ASP.NET controller, men har mere funktionalitet i retning mod EntityFramework.

Klassen injecter f.eks. UnitOfWork servicen, implementerer InternalServerError og TooManyRequests responskoder som kaldefunktioner, og generelle CRUD operationer på TEntity genericen.

Når CreateEntity metoden kaldes fra BookingController, indsættes payload og unitOfWork.Bookings repositoriet. Disse objekter er hoveddelen af oprettelsesprocessen. Derudover angives 3 generic typer: TEntity, TDTO og TPayload, så CreateEntity metoden ved hvilken entity type der er tale om, hvilken type DTO den skal konvertere til, og hvilken type payload den for fra parameteren.

A computer screen shot of a program code

Description automatically generated

Først tjekkes om payloaden er valid og derefter, om der findes et id i payloaden. Er payloaden angivet forkert (som f.eks. mangler en værdi) er det en BadRequest, ligesom hvis payloaden har et id. En oprettelses payload må *ikke* have et id fra oprettelsesstagen, da EntityFramework burde generere et nyt id til entityen.

Når objekttjekene er gennemført succesfuldt, oprettes og gemmes den nye entity og returneres med i en 201 Created http kode samt en DTO adapteret entity.

DTO’en bruges til at undgå objekt cycling problemer, når objektet konverteres til en JSON streng.

**BookingsRepository & BaseRepository**

For at operere CRUD operationer på en bestilling, bruges [BookingsRepository](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/DataAccess/Repositories/BookingRepository.cs) i Data-access-laget. BookingsRepository objektet er specificeret til CRUD operationer på netop Booking modellen, og har bl.a. egen GetFromCitizen metode, som står ud fra andre repositories, der nedarver fra [BaseRepository](https://github.com/DanielSimonsen90/DanhoLibrary/blob/master/DanhoLibrary/NLayer/BaseRepository/BaseRepository__CRUD.cs).

A screen shot of a computer code

Description automatically generated

BaseRepository stammer fra mit eget C# bibliotek, DanhoLibrary, hvor jeg på forhånd har defineret al basal funktionalitet, når det gælder CRUD operationer på EntityFramework modeller. Alle specialiserede repositories nedarver fra BaseRepository.

Når en bestilling tilføjes via BookingsRepository, kaldes [AddAsync](https://github.com/DanielSimonsen90/DanhoLibrary/blob/16cfda8c2e9d9b04eaf0b3c1f0607c0a257df290/DanhoLibrary/NLayer/BaseRepository/BaseRepository__CRUD.cs#L26) metoden, som orignalt sammer fra BaseRepository klassen.

A screen shot of a computer code

Description automatically generated

AddAsync metoden tager imod en TEntity generic, hvor i dette tilfælde i BookingsRepository er en Booking type. Her tjekkes om entitien ikke er null og ikke i forvejen eksisterer i databasen.

Når tjekkene er færdige, tilføjes entiten til \_dbSet, som er den del af EntityFramework, for at tilføje en ændring i cachen, som i BaseController, via UnitOfWork, gemmes i databasen via SaveChangesAsync.

A computer code with text

Description automatically generated

**Svaret tilbage til klienten**

Når svaret fra API’et sendes til klienten, viderestilles borgeren til borgersiden, hvor de får en opdateret UI samt notifikation om, hvornår taxaen ankommer ift. deres bestilte tid.

A screenshot of a computer

Description automatically generated

## **Klassediagram**

Her er klassediagramet over mine entities, der nedarver fra hinanden samt gør brug af BaseEntity fra mit eget bibliotek, DanhoLibrary.

A diagram of a computer

Description automatically generated

## **Sikkerhed**

Jeg har implementeret sikkerhed i både database, api og frontend.

**Databasesikkerhed**

Da jeg gemmer på brugere og deres logins, skal jeg selvfølgelig beskytte mine brugeres data med sikkerhed. Her indgår bl.a. min grund til at have delt mine logins og brugere til deres egne tabeller. Hvis nogen får adgang til min brugertabel, har de stadig ikke adgang til at logge ind som brugeren, men har de i stedet adgang til min logintabel, kan de kun se loginoplysninger som brugernavn, salt, kodeord og brugerid.

Brugernes kodeord er enkrypteret ved brug af hashing med den gemte salt værdi. På den måde kan hackere ikke bruge koderne til noget, da de skal dekrypteres for, at værdien er læselig.

I min [LoginService](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Business/Services/LoginService.cs) på mit Business-lag definerer jeg [GenerateEncryptedPassword](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Business/Services/LoginService.cs#L90) og [IsCorrectPassword](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Business/Services/LoginService.cs#L112), som begge er metoder, der enkrypterer og dekryptere kodeordene.

A computer screen shot of a program

Description automatically generated

Dog kan man argumentere for, at der ikke er nok sikkerhed i password hashing – selv med salt. Det ville være bedre praksis at inkludere brugerens id med i enkrypteringsprocessen, sådan at flere brugere kan have samme kodeord, men ikke have den samme enkrypterede værdi.

**API-sikkerhed**

Der er implementeret API sikkerhed i form af klassisk API access- og refresh-key/token samt udløbsdato på henholdsvis begge nøgler. Dette betyder, at ingen kan komme ind på mit API, medmindre de er logget ind med en valid brugerkonto. De eneste endpoints der ikke er beskyttet af API keys, er oprettelse af bruger og login.

I stedet for at implementere Microsoft Identity, som mange andre ville have gjort med en C# backend, har jeg i stedet lavet mit eget autentificeringssystem. Dette består af en middleware, der eksekverer noget kode, før klienten kommer igennem til det rigtige endpoint.

Min [AuthMiddleware](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/Business/Middlewares/AuthMiddleware.cs)klasse registreres som en middleware service i [Program](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/API/Program.cs#L79) filen i mit API projekt.



Når klienten sender en request til mit API, kalder ASP.NET AuthMiddleware’s [InvokeAsync](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Business/Middlewares/AuthMiddleware.cs#L18) metode sammen med en context, der kan hjælpe middlewarefunktionens funktionalitet.

I min implementation af InvokeAsync, tjekker jeg om det anmodet endpoint *ikke* er /users/authenticate og *ikke* er /users med en POST request. Hvis anmodningen er /users/authenticate eller POST på /users, må klienten gerne komme videre i processen, da jeg ikke kræver autentificering til disse endpoints.

A screen shot of a computer program

Description automatically generated

Alle andre endpoints vil falde ind i min if-statement, som vil tjekke om klienten har en valid AuthTokens objekt i deres cookies. Er den ikke valid og ikke har en valid refreshtoken, forbydes klienten adgang med en Unauthorized respons. Har klienten en udløbet accesstoken men valid refreshtoken, generere og gemmer AuthService et ny AuthTokens objekt til klienten, uden at klienten behøver at gøre noget.

Når klientens tokens er verificerede, får klienten lov til at gå videre til det ønskede endpoint via \_next kaldet.

Jeg får min [AuthService](https://github.com/DanielSimonsen90/CitizenTaxi/blob/main/Projects/Backend/Business/Services/AuthService.cs) til at klare det meste af AuthTokens oprettelsen og cookie tilføjelsen på anmodningsobjektet.

Jeg bruger bl.a. min AuthService til at kalde [GetAuthTokens](https://github.com/DanielSimonsen90/CitizenTaxi/blob/351ef3afd59ac447d64a9a92c39415c6253d207d/Projects/Backend/Business/Services/AuthService.cs#L106) i min middleware.

A computer screen with text and images

Description automatically generated

Funktionen leder efter en cookie med navn ”citizen\_taxi\_authentication” (angivet med konstantstrengen COOKIE\_KEY), konverterer json værdien om til et AuthTokens objekt, og til sidst finder den cachede værdi af objektet via \_cacheService.

Fejler noget i koden, som f.eks. cookien findes ikke eller kunne ikke konverteres rigtigt, antages at cookiens værdi ikke er valid, og derfor returnerer GetAuthTokens null, så klienten ikke kommer igennem middlewaren.

**Frontend-sikkerhed**

Selve hjemmesiden er beskyttet i form af provider state management. Al vigtig information sendes ned igennem forskellige providers (services), som f.eks. bruger- og borgerdata.

Dette er f.eks. med til at sikre, at selvom man snyder med at sætte react’s state via dev-tools eller browser konsollen, ville man ikke kunne udnytte systemet og evt. se administrationssiden som borger.

# Brugervejledning

# Bilag // TODO