

Produktrapport

CitizenTaxi



Daniel Simonsen
01-12-2023

Titelblad

TECHCOLLEGE

Techcollege Aalborg,
Struervej 70,
9220 Aalborg

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

Læsevejledning

Til projektet er der lavet en proces- og en produktrapport. Jeg anbefaler at læse procesrapporten først, da det er vigtigt at forstå problemet samt tidsbegrænsning for at få mest ud af denne rapport.

Undervejs vil du støde på ord markeret med blå skrift og understreg. Disse ord fungerer som eksterne links, som kan være med til at hjælpe med forståelsen af det relevante emne.

Indholdsfortegnelse

Indledning	2
Case beskrivelse	2
Problemformulering	2
Afgrænsning	2
Kravsspecifikation	3
Produktdokumentation	4
Databasediagram	4
Klassediagram	5
Programarkitektur	6
Sikkerhed	16
Brugervejledning	20
Administrator-/Lægesekretærbrugervejledning	21
Borgervejledningen	28
Testspecifikation	33

Indledning

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

Lægesekretærene 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 lægesekretærene, 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 lægesekretærene 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. sygehusadgang (f.eks. bruge deres eksisterende loginoplysninger) og rigtig taxabestilling (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. Google Maps support til markering af destination/afhentning og kritiske beskeder på eksterne services som mail eller sms.

Kravsspecifikation

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

Opfyldt	Krav Id	Krav titel	Kategori	Prioritet	Usecase (beskrivelse)
Ja	Front1	Omstilling til login	Frontend	Skal	Brugeren bliver omstillet til loginsiden, når de ikke er logget ind
Ja	Front2	Viderestilling efter login	Frontend	Skal	Efter login skal brugeren viderestilles til /admin eller /patient alt efter rolle
Ja	Entity1	Role property	Entity	Skal	En bruger har en "Role" property, der bestemmer om brugeren er en sekretær/admin eller patient
Ja	Entity2	Citizen 1..1 Notat	Entity	Skal	En borger kan kun have 1 notat
Ja	Entity3	Citizen 1..* Booking	Entity	Skal	En borger kan have flere bookings
Ja	Operation1	Admin CRUD Borger	Operation	Skal	På admin siden skal administratoren kunne udføre "CRUD" operationer på Citizen modellen
Ja	Operation2	Admin CRUD Notat	Operation	Skal	På admin siden skal administratoren kunne udføre "CRUD" operationer på Note modellen
Ja	Operation3	Admin CRUD Booking	Operation	Skal	På admin siden skal administratoren kunne udføre "CRUD" operationer på Booking modellen
Ja	Sikker1	Kodeordsenkryptering	Sikkerhed	Skal	Brugernes kode skal enkrypteres før loginoplysningerne gemmes i databasen
Ja	Operation5	Citizen CRUD egen booking	Operation	Skal	På borgersiden kan borgeren udføre CRUD operationer på deres egne bookings/bestillinger
Ja	Operation4	Citizen R eget notat	Operation	Burde	På borgersiden kan borgeren se deres eget notat
Nej	Operation6	Ingen UD på Booking en 1 time inden	Operation	Burde	Brugere kan ikke udføre UPDATE eller DELETE operationer på booking, når der er en time tilbage til ankomsttid
Ja	UX1	Side pr. Spørgsmål	UX	Burde	Borgern bør præsenteres for taxabooking formularen med en side pr. Spørgsmål, så borgeren ikke bliver overbelastet af information
Nej	UX2	Notifikationscheckbox	UX	Burde	Brugeren kan stoppe notifikationer ved hjælp af checkbox efter bookingsprocessen er færdig
Delvist	Front3	Live opdatering efter bestilling	Frontend	Kunne	Når en bestilling udføres, oprettes der forbindelse til en live-opdatering af taxaens estimerede tid
Ja	Noti1	Notifikationsinterval	Notifikation	Kunne	Borgeren skal modtage en notifikation på hjemmesiden, når der er 1 time, 30 minutter, 20 minutter, 15 minutter, 10 minutter og 5 minutter tilbage til at taxaen er ankommet samt en notifikation, når taxaen er ankommet.
Ja	Sikker2	Usertoken	Sikkerhed	Gerne	Efter brugerlogin gemmes en token i cookies, som er med til at verificere brugerens identitet
Nej	Noti2	Kritisk besked på mail	Notifikation	Gerne	Kritiske meddelelser fra taxa servicen (f.eks. forskinkelse eller aflysning) skal sendes på mail
Nej	Noti3	Kritisk besked på sms	Notifikation	Hvis tid	Kritiske meddelelser fra taxa servicen (f.eks. forskinkelse eller aflysning) skal sendes på sms
Nej	UX3	Google maps adressen	UX	Hvis tid	Borgeren kan vælge adresse v.h.a Google maps, når der oprettes en booking

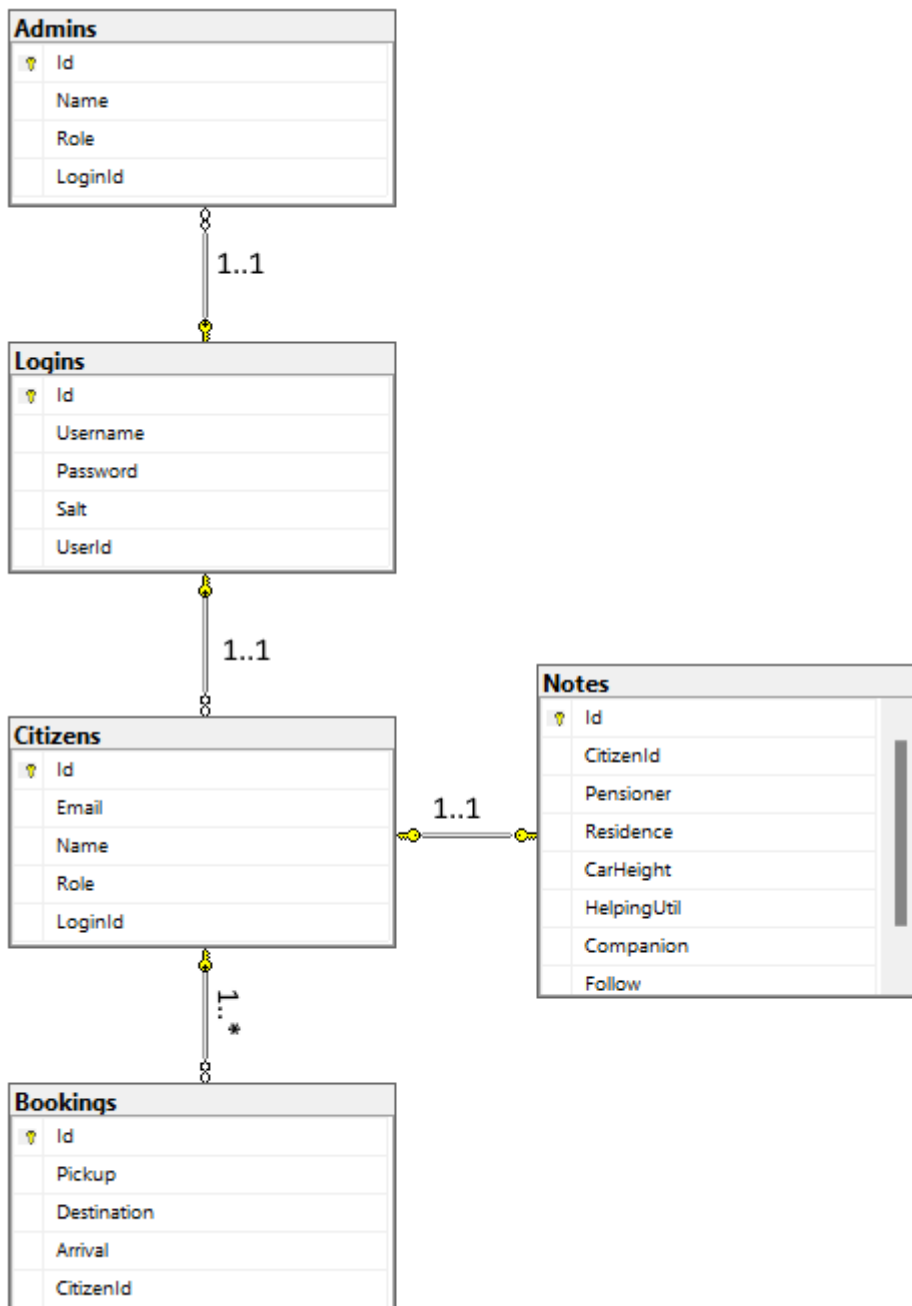
Som vist på billedet, har jeg opfyldt alle mine **skal**-krav, halvdelen af mine **burde**-krav, ca. 75% af mine **kunne**-krav, og en kvartdel af mine **gerne**-/**hvis tid**-krav. Desværre giver det en hel score på 70% af mine krav, der er opfyldt, og selvom mange af mine uopfyldte krav omhandler brugeroplevelse 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 f.eks. haft nogle flere dage til projektet, ville jeg nok have fuldført alle mine krav, undtagen Front3, Liveopdatering 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.

Produktdokumentation

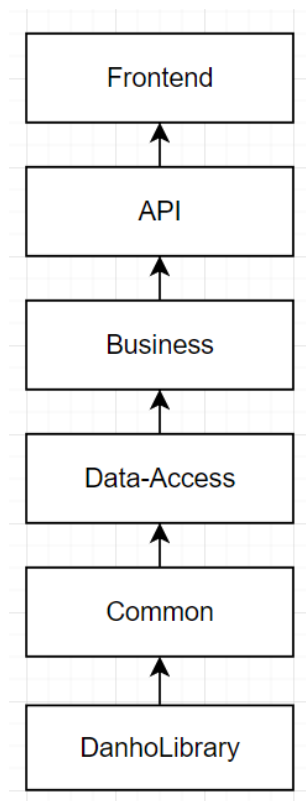
Databasediagram



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 mit eget DanhoLibrary bibliotek, så mine lag ligner billedet nedenunder.



N-lagsstrukturen er med til at separere bestemte typer for logik i deres eget lag. F.eks. befinder alle modeller i Common, alt database-relateret kode i Data-Access og resterende hjælperfunktionalitet som services, middlewares og exceptions i Business-laget.

For at illustrere projektets struktur tages der udgangspunkt i en situation, hvor en borger vil bestille en taxa.

Når en borger vil bestille en taxa

For at en borger kan bestille en taxa, skal borgeren logge ind på systemet først. Borgeren skal indtaste sit 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 berettiget 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 borgeren bliver præsenteret for sit notat, sine bestillinger, og evt. hjælp til rettelse af notat eller oprettelse af taxabestilling.

The screenshot shows the CitizenTaxi web application interface. At the top, the header includes the logo 'CitizenTaxi' with the tagline 'Svenskegrønne projekt' and a user status 'Du er logget ind som Test Borger' with a 'Log ud' button. The main content is divided into two columns. The left column, titled 'Dine bestillinger', contains the text 'Du har ingen bestillinger endnu.' and a 'Bestil en ny tid' button. The right column, titled 'Dit notat', contains several input fields for 'Bilhøjde', 'Hjælpe midler', 'Ledsager', 'Følges', and 'Bopæl', each with a dropdown menu. Below these fields is a section titled 'Er dit notat forkert?' with a text input field containing '+45 00 00 00 00' and a link 'Ring til lægesekretærene direkte'. The footer contains copyright information '© 2023 Daniel Simonsen', a version number 'CitizenTaxi af Daniel Simonsen, Datafileskriver elev id:pd111123', and a link 'Se projektet på GitHub'.

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 tryk under turen.

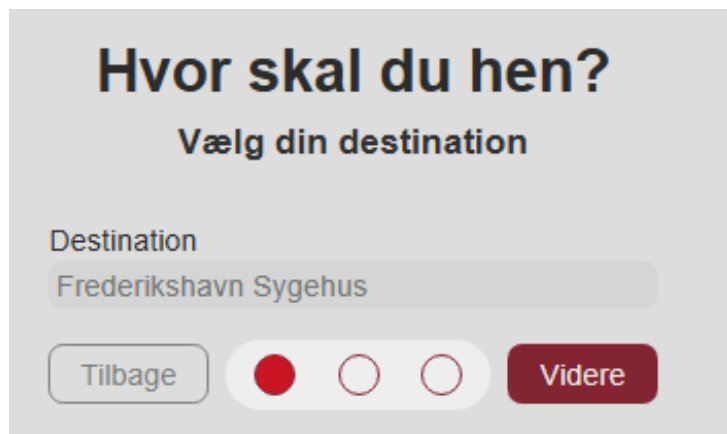
Har borgeren *ikke* et notat tilknyttet, kan borgeren ikke bestille en taxa. Notatet oprettes af lægesekretærene, og ideelt set burde en borger ikke kunne eksistere i systemet, hvis borgeren ikke har et notat.

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

The screenshot shows a close-up of the 'Dit notat' section. A message box is displayed over the 'Bestil en ny tid' button, containing the text 'Du skal have et notat for at kunne bestille en tid. notat endnu.' The 'Dit notat' title is visible in the background.

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.



Hver gang borgeren trykker på "Videre" og "Afslut", kaldes [onFormSubmit](#) funktionen i [BookTaxiLayout](#).

Funktionen serialiserer formens indkommende data og tilføjer det nye data til det eksisterende data.

```
// Function to update the payload state
const updatePayload = (data: Partial<BookingStepsPayload>) => setPayload(prev => ({ ...prev, ...data }));

// When the step is submitted, serialize the form data and update the payload state
// This will automatically trigger the useEffect hook below to navigate to the next step
const onFormSubmit = (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();
  const data = serializeForm<BookingStepsPayload>(e.target as HTMLFormElement);
  updatePayload(data);
  setSubmitted(true);
};
```

Til sidst i funktionen sættes "setSubmitted" staten til true, som trigger en update/change lifecycle event via [useUpdateEffect](#) hooket.

```
// When any of the dependencies update, run the callback
useUpdateEffect(() => {
  // If the form hasn't been submitted, don't do anything as this was triggered by internal state changes
  if (!submitted) return;

  // If the user has more steps to complete, navigate to the next step
  if (canContinue) {
    setSubmitted(false);
    return navigateToStep(step + 1);
  }

  // If the user has completed all steps, make sure the citizenId is set
  if (!payload.citizenId) return updatePayload({ citizenId: citizen?.id });

  // Once the steps are completed and the citizenId is set, send the booking to the API
  (async function sendBooking() {
    const arrival = dateAsUTC(new Date(`${payload.date}T${payload.time}Z`));

    await dispatch('id' in payload && payload.id !== undefined ? 'updateBooking' : 'createBooking', {
      ...payload,
      // Convert the date and time to a Date object as UTC
      arrival
    } as BookingModifyPayload<any>);

    // Navigate to the citizen page once the booking has been sent
    navigate('/');
  })();
}, [submitted, payload, canContinue, navigate]);
```

Hele hooket er med til at følge borgeren videre i bestillingsprocessen. Er formen ugyldig, får borgeren en notifikation. Har payloaden ikke et 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](#) funktionen fra [useApiActions](#) hooket.

Dispatching

For at sende requests til API'et, er der lavet et action-dispatching system. Den returnerede dispatch funktion fra useApiActions hooket tilbyder forskellige actions, der tager imod argumenter relevant for actionen. 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.

```
export default function useApiActions({
  async function dispatch<Action extends ActionNames>(  
    switch (action) {  
      case 'createCitizen':  
      case 'createNote':  
      case 'createBooking':  
      case 'updateCitizen':  
      case 'updateBooking':  
      case 'updateNote': {  
        // These are the same object, but typescript interprets them as different types due to update having an id and creat  
        const [createPayload] = args as Actions['createCitizen' | 'createNote' | 'createBooking'];  
        const [updatePayload] = args as Actions['updateCitizen' | 'updateBooking' | 'updateNote'];  
  
        const isUpdateAction = action.includes('update');  
        const response = await Request<any, Guid>(isUpdateAction ? `${baseEndpoint}/${updatePayload.id}` : baseEndpoint, {  
          method: isUpdateAction ? 'PUT' : 'POST',  
          body: createPayload // Doesn't matter whether createPayload or updatePayload is used, as they are the same object  
        });  
  
        if (response.success) {  
          if (setCitizens && action.includes('Citizen')) setCitizens(citizens => {  
          });  
          if (setCitizen) setCitizen(citizen => {  
          });  
          else console.warn(`No setter was provided for ${action}, so the response data was not set.`);  
        }  
  
        showNotification({  
          message: response.success ? `${entityName} ${isUpdateAction ? 'opdateret' : 'oprettet'}` : response.text,  
          type: response.success ? 'success' : 'error',  
        });  
      }  
    }  
  )  
}
```

Når en createBooking sendes til dispatch funktionen, efter borgeren har trykket "Afslut", 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 anmodningen gik.

Auth middleware

Når hjemmesiden sender en anmodning til API'et, går den først igennem [AuthMiddleware](#) fra [Business-laget](#). Middlewareen er med til at validere klientens autoritet. Klienten *skal* være logget ind på en valid konto og have en valid access-/refresh token i klients cookies, ellers er anmodningen ugyldig.

```
public async Task InvokeAsync(HttpContext context)
{
    if (!context.Request.Path.HasValue) throw new Exception("Request path is null");

    // If user isn't re-authenticating or creating an account, check if they have valid tokens
    if (!context.Request.Path.Value.Contains("/users/authenticate") &&
        !(context.Request.Path.Value.Contains("/users") && context.Request.Method == "POST") &&
        !context.Request.Path.Value.Contains("/users/ping"))
    {
        try
        {
            // Get the AuthTokens object from the cookies
            AuthTokens? auth = _authService.GetAuthTokens(context.Request);

            // If no tokens are found, or the access & refresh tokens are expired, throw Unauthorized
            if (auth is null || (auth.AccessToken.IsExpired && auth.RefreshToken.IsExpired)) throw new Exception("Unauthorized");
            // If access token is expired and refresh token is not, generate new tokens and save them to cookies
            if (auth.AccessToken.IsExpired) _authService.GenerateTokensAndSaveToCookies(auth.UserId, context.Response);
        }
        catch (Exception ex)
        {
            // Return 401 Unauthorized from the exception
            context.Response.StatusCode = StatusCodes.Status401Unauthorized;
            await context.Response.WriteAsync(ex.Message);
            return;
        }
    }

    // Continue to the next middleware
    await _next(context);
}
```

Mere og dybere information om AuthMiddleware findes længere nede i rapporten.

BookingController & BaseController

Når klienten kommer igennem AuthMiddleware'en med POST request til endpoint /api/bookings, håndterer [BookingsControlleren](#) requesten via [CreateBooking](#) metoden i [API-laget](#).

```
/// <summary>
/// Create a <see cref="Booking"/> in the database from the given <paramref name="payload"/>.
/// This uses the <see cref="BaseController.CreateEntity{TPayload,TEntity}"/> method to DRY up the code
/// </summary>
/// <param name="payload">The payload from the frontend</param>
/// <returns>HTTP response from <see cref="BaseController.CreateEntity{TPayload, TEntity}(TPayload,
BaseRepository{TEntity, Guid})"/></returns>
0 references
[HttpPost] public async Task<IActionResult> CreateBooking([FromBody] BookingModifyPayload payload)
{
    Citizen? citizen = unitOfWork.Citizens.GetWithRelations(payload.CitizenId, Citizen.RELATIONS);
    if (citizen is null) return NotFound($"Citizen with id {payload.CitizenId} was not found");
    if (citizen.Note is null) return BadRequest($"Citizen with id {payload.CitizenId} does not have a
        note, and can therefore not have bookings.");

    return await CreateEntity<BookingModifyPayload, BookingDTO, Booking>(payload, unitOfWork.Bookings);
}
```

Efter null checks på citizen og citizen.Note begynder den reelle oprettelsesproces via [BaseController's CreateEntity](#) 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 BookingsController, indsættes payload og unitOfWork.Bookings repositoret. 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 får fra parameteren.

Billede af CreateEntity funktionen findes på næste side.

```
/// <summary> Create an entity in the database from the given payload and reposi ...
6 references
public async Task<IActionResult> CreateEntity<TPayload, TDTO, TEntity>(TPayload payload,
    BaseRepository<TEntity, Guid> repository)
    where TEntity : BaseEntity<Guid> // The entity must be a BaseEntity with a Guid as Id
    where TDTO : ABaseDTO // The DTO must be a ABaseDTO
    where TPayload : ABaseModifyPayload // The payload must be a ABaseModifyPayload
{
    try
    {
        // If the payload is not valid, return a bad request with the ModelState, so the frontend can see
        // what is wrong
        if (!ModelState.IsValid) return BadRequest(ModelState);
        if (payload.Id is not null && payload.Id != Guid.Empty) return BadRequest("Id must be null or
            empty");

        // Adapt the payload to the entity and save it to the database
        var entity = payload.Adapt<TEntity>();
        var created = await repository.AddAsync(entity);
        await unitOfWork.SaveChangesAsync();

        // Return the created entity
        return Created($"api/{controller}/{created.Id}", created.Adapt<TDTO>());
    }
    // If any exception is thrown from the AddAsync method, return appropriate HTTP response.
    catch (ArgumentNullException ex) { return BadRequest($"Invalid argument provided: {ex.Message}"); }
    catch (ArgumentException ex) { return BadRequest(ex.Message); }
    // Backup exception in case something else went wrong
    catch (Exception ex) { return InternalServerError(ex.Message); }
}
```

Først tjekkes om payloaden er gyldig, 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 oprettelsespayload må *ikke* have et id fra oprettelsesstagen, da EntityFramework burde generere et nyt id til entityen.

Når objektjekkene er gennemført succesfuldt, oprettes og gemmes den nye entity, og anmodningen 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](#) i [Data-access-laget](#).

BookingsRepository objektet er specificeret til CRUD operationer på netop Bookingmodellen og har bl.a. egen GetFromCitizen metode, som står ud fra andre repositories, der nedarver fra [BaseRepository](#).

```
public class BookingRepository : BaseRepository<Booking, Guid>
{
    1 reference
    public BookingRepository(CitizenTaxiDbContext context) : base(context) { }

    2 references
    public List<Booking> GetFromCitizen(Citizen citizen) => Booking.RELATIONS
        .Aggregate(_dbSet.AsQueryable(), (current, relation) => current.Include(relation))
        .Where(booking => booking.Citizen.Id == citizen.Id)
        .ToList();
}
```

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

Når en bestilling tilføjes via BookingsRepository, kaldes [AddAsync](#) metoden, som oprindeligt kommer fra BaseRepository klassen.

```
public virtual async Task<TEntity> AddAsync(TEntity entity)
{
    if (entity is null) throw new ArgumentNullException(nameof(entity));
    if (Exists(entity.Id)) throw new ArgumentException($"Entity with id {entity.Id} already exists.");

    EntityEntry<TEntity> entry = await _dbSet.AddAsync(entity);
    return entry.Entity;
}
```

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

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

```
// Adapt the payload to the entity and save it to the database
var entity = payload.Adapt<TEntity>();
var created = await repository.AddAsync(entity);
await unitOfWork.SaveChangesAsync();
```


Svaret tilbage til klienten

Når svaret fra API'et sendes til klienten, viderestilles borgeren til borgersiden, hvor borgeren får en opdateret UI samt notifikation om, hvornår taxaen ankommer i forhold til den bestilte tid.

CitizenTaxi

Svendeprøve-projekt

Du er logget ind som Test Borger

Borger

Log ud

Dine bestillinger

Til Frederikshavn Sygehus

Fra Solvej 10, 0000 God by

01/12/2023,
16:30

Bestil en ny tid

Ændre bestilling

Afbestil

Dit notat

Bilhøjde

Alle

Hjælpe midler

Ingen

Ledsager

Alene

Følges

Nej

Bopæl

Solvej 10, 0000 God by

Er dit notat forkert?

Du kan ringe til lægesekretærene og få det rettet med nummeret.

+45 00 00 00 00

— eller —

[Ring til lægesekretærene direkte](#)

© 2023 Daniel Simonsen

CitizenTaxi af Daniel Simonsen, Datatekniker-elev h6pd111123

[Processrapport](#)[Produktrapport](#)

[Se projektet på GitHub](#)

BEMÆRK

Din taxa ankommer om 16 minutter

Sikkerhed

Jeg har implementeret sikkerhed i både database, API og frontend for at undgå falske eller lækket data.

Databasesikkerhed

Da jeg gemmer på brugere og deres logins, skal jeg sikre mig at beskytte mine brugeres data med noget 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 de har i stedet adgang til min logintabel, og dermed kan de kun se loginoplysninger som brugernavn, salt, kodeord og brugerid.

Brugernes kodeord er enkrypteret ved brug af hashing med den gemte saltværdi. På den måde kan hackere ikke bruge adgangskoderne til noget, da de skal dekrypteres, for at værdien er læselig og kan bruges i produktet.

I min [LoginService](#) på mit [Business-lag](#) definerer jeg [GenerateEncryptedPassword](#) og [IsCorrectPassword](#), som begge er metoder, der enkrypterer og dekrypterer kodeordene.

```
/// <summary> Generates a secure password hash and salt from the given unencrypt ...  
3 references | 1/1 passing  
public static string GenerateEncryptedPassword(string unencrypted, string salt)  
{  
    // Convert unencrypted string into hash data with provided salt  
    byte[] hash = Rfc2898DeriveBytes.Pbkdf2(  
        Encoding.UTF8.GetBytes(unencrypted),  
        Convert.FromBase64String(salt),  
        iterations,  
        hashAlgorithm,  
        keySize);  
  
    return Convert.ToBase64String(hash);  
}  
2 references | 1/1 passing  
public static string GenerateSalt() => Convert.ToBase64String(RandomNumberGenerator.GetBytes(keySize));  
  
/// <summary> Boolean representation of whether the given unencrypted password m ...  
1 reference  
public static bool IsCorrectPassword(string unencrypted, string salt, string encrypted)  
{  
    byte[] saltValue = Convert.FromBase64String(salt);  
    byte[] hash = Rfc2898DeriveBytes.Pbkdf2(  
        Encoding.UTF8.GetBytes(unencrypted),  
        saltValue,  
        iterations,  
        hashAlgorithm,  
        keySize);  
    string convertedUnencrypted = Convert.ToBase64String(hash);  
  
    return convertedUnencrypted == encrypted;  
}
```

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 i enkrypteringsprocessen, således 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 [AuthMiddlewareklasse](#) registreres som en middleware service i [Program](#) filen i mit [API-lag](#).

```
app.UseMiddleware<AuthMiddleware>();
```

Når klienten sender en anmodning til mit API, kalder ASP.NET AuthMiddleware's [InvokeAsync](#) metode sammen med en context, der kan hjælpe middlewarefunktionens funktionalitet.

I min implementering af InvokeAsync, tjekker jeg om det anmodede 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.

```
public async Task InvokeAsync(HttpContext context)
{
    if (!context.Request.Path.HasValue) throw new Exception("Request path is null");

    // If user isn't re-authenticating or creating an account, check if they have valid tokens
    if (!context.Request.Path.Value.Contains("/users/authenticate") &&
        !(context.Request.Path.Value.Contains("/users") && context.Request.Method == "POST"))
    {
        try
        {
            // Get the AuthTokens object from the cookies
            AuthTokens? auth = _authService.GetAuthTokens(context.Request);

            // If no tokens are found, or the access & refresh tokens are expired, throw Unauthorized
            if (auth is null || (auth.AccessToken.IsExpired && auth.RefreshToken.IsExpired)) throw new Exception("Unauthorized");
            // If access token is expired and refresh token is not, generate new tokens and save them to cookies
            if (auth.AccessToken.IsExpired) _authService.GenerateTokensAndSaveToCookies(auth.UserId, context.Response);
        }
        catch (Exception ex)
        {
            // Return 401 Unauthorized from the exception
            context.Response.StatusCode = StatusCodes.Status401Unauthorized;
            await context.Response.WriteAsync(ex.Message);
            return;
        }
    }

    // Continue to the next middleware
    await _next(context);
}
```

Alle andre endpoints vil falde ind i min if-statement, som vil tjekke, om klienten har en valid AuthTokens objekt i sine cookies. Er den ikke valid, og AuthTokens objektet ikke har en valid refreshtoken, forbydes klientens

adgang med en Unauthorized respons. Har klienten en udløbet accesstoken men valid refreshtoken, genererer 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](#) til at klare det meste af AuthTokens oprettelsen og cookietilføjjelsen på anmodningsobjektet.

Jeg bruger bl.a. min AuthService til at kalde [GetAuthTokens](#) i min middleware.

```
/// <summary>
/// Get the <see cref="AuthTokens"/> from the cookies
/// </summary>
/// <param name="request">Request object that has the cookies</param>
/// <returns>The found <see cref="AuthTokens"/>, if any</returns>
2 references
public AuthTokens? GetAuthTokens(HttpRequest request)
{
    try
    {
        string cookie = request.Cookies[COOKIE_KEY];
        AuthTokens parsedCookie = JsonSerializer.Deserialize<AuthTokens>(cookie, new JsonSerializerOptions()
        {
            // Map the camelCase cookie to PascalCase
            PropertyNameCaseInsensitive = true,
        }) ?? throw new Exception("Cookie is null");

        // Look for the auth in the cache
        _cacheService.AuthTokens.TryGetValue(parsedCookie.AccessToken.Value, out AuthTokens? auth);

        // Return the result
        return auth;
    }
    catch
    {
        // If the cookie is not found or invalid, return null
        return null;
    }
}
```

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 det at cookiens værdi ikke er gyldig, og derfor returnerer `GetAuthTokens` null, så klienten ikke kommer igennem middlewareen.

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 browserkonsollen, ville man ikke kunne udnytte systemet og evt. se administrationssiden som borger.

Brugervejledning

Da siden er delt op i 2 forskellige roller, borger og lægesekretær/administrator, er der 2 forskellige brugervejledninger. Da borgerrollen er afhængig af lægesekretærrollen, fordi borgerrollen kræver et notat for at kunne bestille en taxa, forklares administrationsvejledningen først, selvom det reelt set omhandler færrest mennesker.

Produktet findes på URL'en: <https://citizentaxi.netlify.app/>

Generel brugervejledning

For at få adgang til CitizenTaxi skal du logge ind. Som demo kan du vælge at oprette din egen konto ved tryk på den blåkantede "Opret"-knap, hvor du skal følge formularen og oprette din konto. Som officiel løsning fjernes opretknappen, da borgerne til CitizenTaxi bør oprettes af lægesekretærene eller systemadministratorerne.

The screenshot shows the CitizenTaxi login and registration interface. At the top, it says 'CitizenTaxi' and 'Svendeprøve-projekt'. Below this is a 'Login' section with the text 'For at benytte CitizenTaxi, skal du logge ind.' It contains two input fields: 'Brugernavn' (username) with the value 'testadmin' and 'Kodeord' (password) with masked characters. There are buttons for 'Opret' (Create), 'eller' (or), and 'Log ind' (Log in). Below the login section is a 'Har du ikke en konto?' (Don't you have an account?) section with the text 'Du kan oprette en konto her' (You can create an account here). It includes a small disclaimer: 'I produktet skal borgerne kunne trykke på opretknappen, så de kan lave kontoer i systemet med deres notat.' This section has input fields for 'Navn' (Name), 'Rolle' (Role) with a dropdown menu showing 'Borger' (Citizen), 'Email', 'Brugernavn' (Username), and 'Kodeord' (Password). There is an 'Opret konto' (Create account) button at the bottom.

Loginoplysninger til Admin:

Brugernavn: testadmin

Kodeord: testadmin123

Loginoplysninger til Borger

Brugernavn: testborger

Kodeord: testborger123

Prøv gerne at lave egen konto.

Administrator-/Lægesekretærbrugervejledning

Når du er logget ind som administrator, præsenteres du for 3 modeller/undersider.

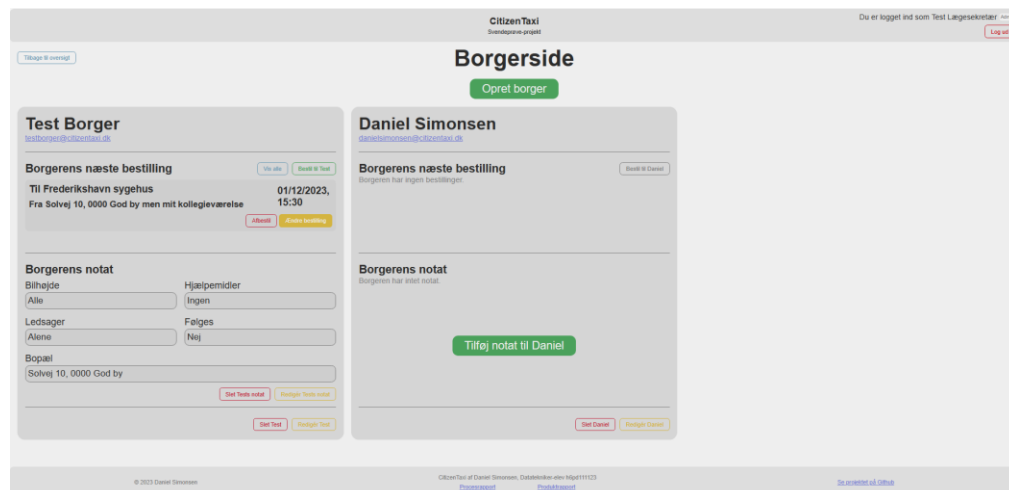
Alle undersider er med til at overskue den udvalgte model uden for meget information. Borgersiden viser alt om en borger, notatsiden viser alle borgere med deres notat, og bestillingssiden viser alle borgere med deres taxabestillinger.



Alle former for borgerinformation findes inde på borgersiden, hvorimod de resterende 2 sider kun viser den data, der er relevant for den valgte model.

Borgerside

På borgersiden vises alle borgere i systemet. Her renderes et kort pr. borger med alle borgerens information såsom mail, bestillinger og notat.



Som CitizenTaxi administrator har man adgang til oprettelse, redigering og sletning af borgere, borgernes notater og borgernes taxabestillinger.

Opret en borger

For at oprette en borger skal man trykke på den grønne "Opret borger" knap. Dette åbner for Opret Borger modallen, hvor man kan indtaste relevante informationer om en borger.

Her er der bl.a. også mulighed for at tilføje et notat direkte til borgeroprettelsen ved tryk på den blå "Tilføj notat" knap. Når formularen er sendt til API'et, kan borgeren logge ind og benytte CitizenTaxi's service.

The screenshot shows the 'Opret Borger' modal form. It has a title bar with 'CitizenTaxi' and a close button. The form contains the following fields: 'Borgerens navn', 'Borgerens e-mail', 'Borgerens brugernavn', and 'Borgerens kodeord'. There are 'Vis kodeord' buttons next to the 'Borgerens kodeord' field. At the bottom, there are two buttons: 'Tilføj notat' (blue) and 'Opret borger' (green).

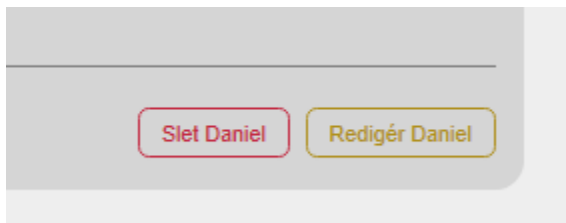
This screenshot shows the 'Opret Borger' modal form with dropdown menus. The fields are: 'Borgerens navn', 'Borgerens e-mail', 'Borgerens brugernavn', 'Borgerens kodeord' (with a 'Vis kodeord' button), 'Bilhøjde' (dropdown with 'Alle' selected), 'Hjælpemiddel' (dropdown with 'Ingen' selected), 'Ledsager' (dropdown with 'Alene' selected), 'Følges' (dropdown with 'Nej' selected), and 'Bopæl'. At the bottom, there are two buttons: 'Fortryd notat' (red) and 'Opret borger med notat' (green).

Vælger man *ikke* at tilføje et notat med i borgeroprettelsen, kan man senere hen tilføje et notat til borgeren.

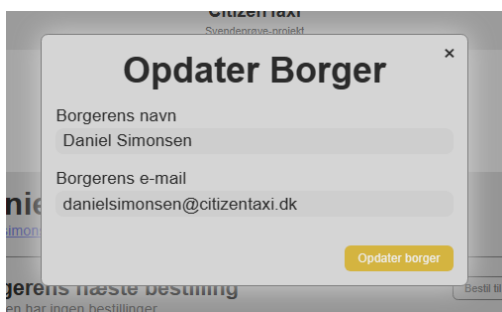
The screenshot shows the CitizenTaxi admin interface for a citizen named Daniel Simonsen. It includes the citizen's name, email, and a section for 'Borgerens næste bestilling' with a 'Bestil til Daniel' button. Below that is a section for 'Borgerens notat' with a 'Tilføj notat til Daniel' button. At the bottom, there are 'Slet Daniel' and 'Redigér Daniel' buttons.

Opdatering af borger

Kommer der undervejs nogle ændringer til borgeren som email- eller navneskift, kan man trykke på den gule "Redigér <Borgerens navn>" knap i bunden af borgerens kort.

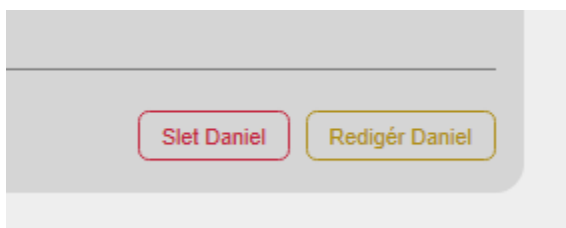


Når der trykkes på knappen, åbnes der en redigeringsmodal, der ligner meget oprettelsesmodallen. Her har man adgang til at ændre borgerens navn og email.



Sletning af borger

Skal man slette en borger, kan man trykke på den røde "Slet <Borgerens navn>" knap i bunden af borgerens kort.



Her åbnes en sletningsmodal for at være sikker på, at du ikke har trykket på slet-knappen ved en fejltagelse.



Oprettelse af notat

Har borgeren intet notat, kan man tilføje det enten i borgerkortet på borgersiden eller i borgerkortet på notatsiden.

Der vises en stor grøn "Tilføj notat til <Borgerens navn>". Når knappen er trykket, vises oprettelsesmodallen til et notat.

The screenshot shows a modal window titled "Opret notat" (Create note) with a close button (X) in the top right corner. The modal contains several dropdown menus and a text input field. The dropdowns are labeled: "Bilhøjde" (selected: "Alle"), "Hjælpe middel" (selected: "Ingen"), "Ledsager" (selected: "Alene"), and "Følgning" (selected: "Nej"). Below these is a text input field labeled "Bopæl". At the bottom of the modal are two buttons: "Fortryd" (Cancel) and "Opret notat" (Create note).

Her er der bl.a. prædefinerede valgmuligheder, som er relevante for et borgernotat.

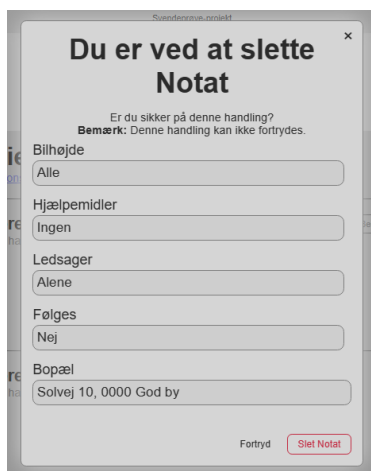
Redigering af notat

Flytter borgeren bopæl eller har borgeren brug for nyt hjælpemiddel mm. kan det selvfølgelig ændres på. I bunden af notatsektionen i borgerkortet, kan man trykke på den gule "Redigér <Borgerens navn>s notat", hvor redigeringsmodallen for borgerens notat vises.

The screenshot shows a modal window titled "Opdater notat" (Update note) with a close button (X) in the top right corner. The modal contains the same dropdown menus and text input field as the "Opret notat" modal, but with pre-filled values: "Bilhøjde" (Alle), "Hjælpe middel" (Ingen), "Ledsager" (Alene), "Følgning" (Nej), and "Bopæl" (Struervej 70, 9000 Aalborg). At the bottom of the modal are two buttons: "Fortryd" (Cancel) and "Opdater notat" (Update note). The background shows a blurred view of the "Borgersiden" (Citizen's page) with a "Redigér Daniels notat" button visible.

Sletning af notat

I de fleste tilfælde kommer man ikke til at slette borgerens notat uden at slette borgeren, men kommer man ud for sådan en situation, hvor det kan være nødvendigt, er der selvfølgelig en mulighed. I bunden af notatsektionen af borgerkoret findes en rød "Slet <Borgerens navn>s notat", hvor sletningsmodallen vises.

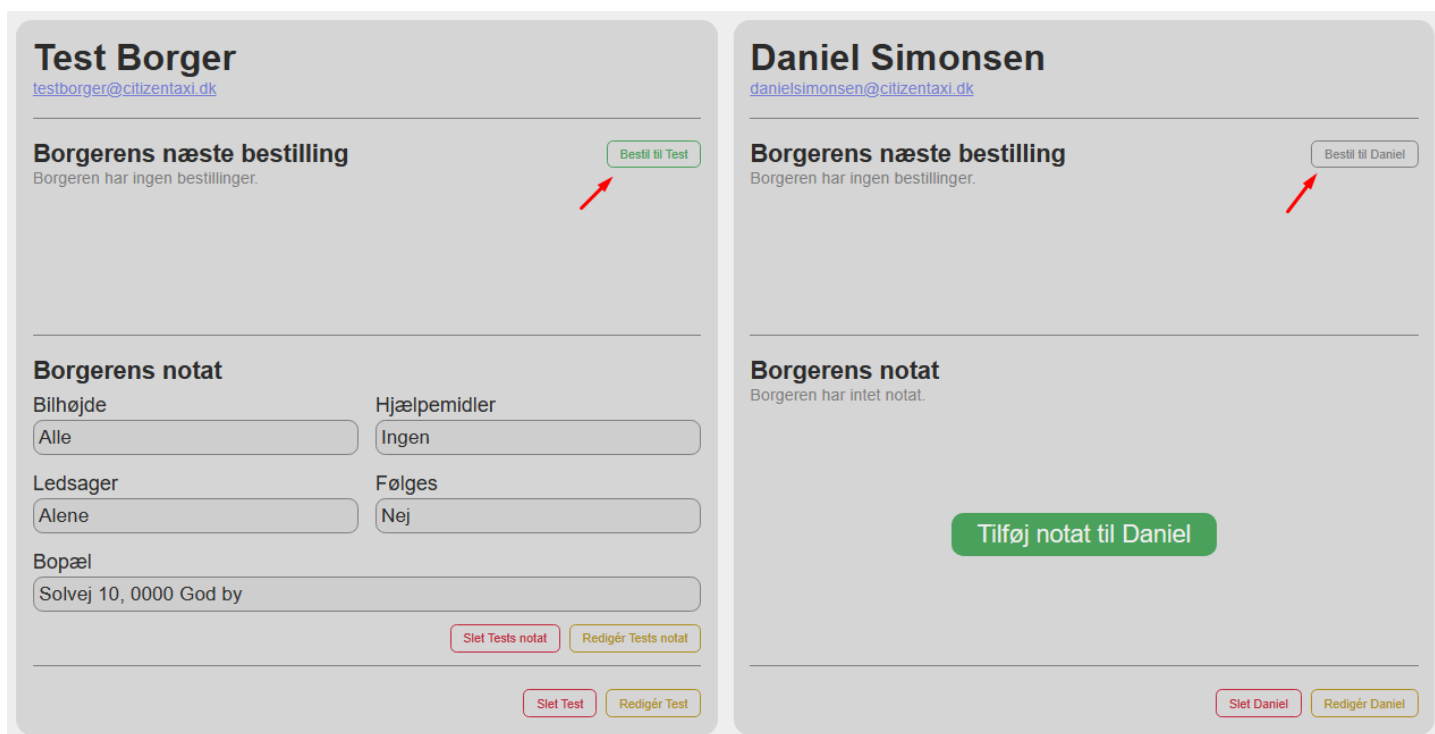


The modal is titled "Du er ved at slette Notat" (You are about to delete Note). It asks "Er du sikker på denne handling?" (Are you sure about this action?) and includes a warning: "Bemærk: Denne handling kan ikke fortrydes." (Note: This action cannot be undone). Below this are several input fields for form data: "Bilhøjde" (Alle), "Hjælpebidler" (Ingen), "Ledsager" (Alene), "Følges" (Nej), and "Bopæl" (Solvej 10, 0000 God by). At the bottom right, there is a red button labeled "Slet Notat" and a "Fortryd" (Undo) link.

Tilføjelse af taxabestilling

Da nogle borgere er mere IT-skræmte end telefoniskræmte, er der også mulighed for at lægesekretærene kan bestille en taxa for borgerne.

For at kunne bestille en taxa til en borger, skal borgeren have et notat tilknyttet. Her vises de forskellige "Bestil til <Borgerens navn>" knapper, hvor Daniel ikke har et notat tilknyttet, men Test Borger har.



The image shows two side-by-side user profile cards. The left card is for "Test Borger" (testborger@citizentaxi.dk) and the right is for "Daniel Simonsen" (danielsimonsen@citizentaxi.dk). Both cards have a section for "Borgerens næste bestilling" (Citizen's next booking) with a "Bestil til [Name]" button. A red arrow points to the "Bestil til Test" button on the left and the "Bestil til Daniel" button on the right. Below this is the "Borgerens notat" (Citizen's note) section. For "Test Borger", this section contains a form with fields for "Bilhøjde", "Hjælpebidler", "Ledsager", "Følges", and "Bopæl", along with "Slet Tests notat" and "Redigér Tests notat" buttons. For "Daniel Simonsen", this section states "Borgeren har intet notat." (The citizen has no note) and features a large green "Tilføj notat til Daniel" (Add note to Daniel) button. At the bottom of each card are buttons to "Slet [Name]" and "Redigér [Name]".

Når den grønne "Bestil til <Borgerens navn>" knap trykkes på, åbnes der en oprettelsesmodal til en taxabestilling.

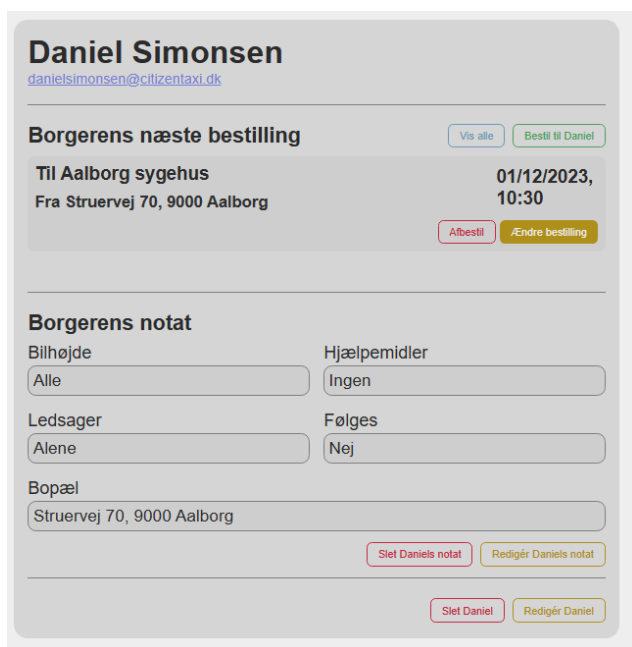


The screenshot shows a modal window titled "Opret taxabestilling" with a close button (X) in the top right corner. The form contains the following fields and controls:

- Til:** A text input field for the destination.
- Ankomst dato:** A date input field showing "30/11/2023" with a calendar icon.
- Ankomst tid:** A time input field showing "--:--" with a clock icon.
- Fra:** A text input field for the pickup location.
- Buttons:** "Fortryd" (Cancel) and "Opret bestilling" (Create booking).

Her indtastes destination, afhentning, ankomstdato og ankomsttid, så taxaservicen har alle de nødvendige oplysninger.

Når du trykker på "Opret bestilling", opdateres borgerens kort med den nye bestilling.



The screenshot shows a profile page for "Daniel Simonsen" with the email "danielsimonsen@citizentaxi.dk". The page displays the "Borgerens næste bestilling" (Citizen's next booking) and "Borgerens notat" (Citizen's note).

Borgerens næste bestilling

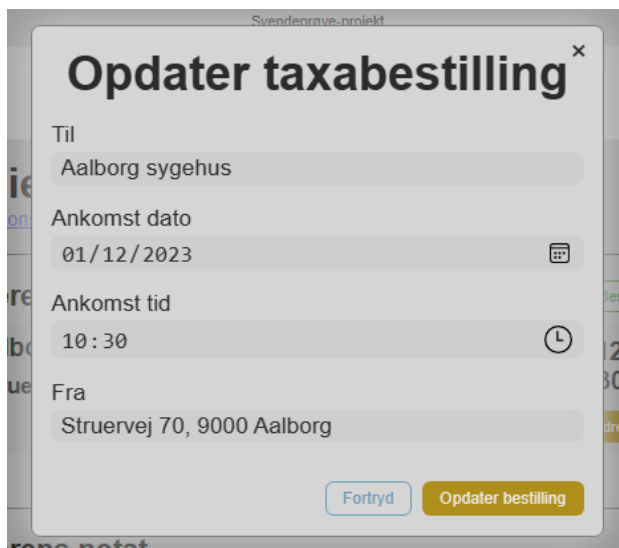
Til Aalborg sygehus	01/12/2023,
Fra Struervej 70, 9000 Aalborg	10:30
Afbestil /Endre bestilling	

Borgerens notat

Bilhøjde: <input type="text" value="Alle"/>	Hjælpemidler: <input type="text" value="Ingen"/>
Ledsager: <input type="text" value="Alene"/>	Følges: <input type="text" value="Nej"/>
Bopæl: <input type="text" value="Struervej 70, 9000 Aalborg"/>	
Slet Daniels notat	Redigér Daniels notat
Slet Daniel	Redigér Daniel

Redigering af taxabestilling

For at redigere en taxabestilling, kan man trykke på den gule "Ændre bestilling" ved notatsektionen. Når knappen er blevet trykket på, åbnes redigeringsmodallen til den valgte bestilling.



The screenshot shows a modal window titled "Opdater taxabestilling" with a close button (X) in the top right corner. The window contains the following fields:

- Til:** Aalborg sygehus
- Ankomst dato:** 01/12/2023 (with a calendar icon)
- Ankomst tid:** 10:30 (with a clock icon)
- Fra:** Struervej 70, 9000 Aalborg

At the bottom of the modal, there are two buttons: "Fortryd" (light blue) and "Opdater bestilling" (yellow).

Sletning/Afbestilling af taxabestilling

For at afbestille en borgers taxabestilling, kan man trykke på den røde "Afbestil"-knap ved bestillingen. Dette åbner for sletningsmodallen.



The screenshot shows a modal window titled "Du er ved at afbestille Daniel Simonsens taxa." with a close button (X) in the top right corner. The window contains the following text and fields:

Er du sikker på at du vil afbestille?

Til Aalborg sygehus	01/12/2023,
Fra Struervej 70, 9000 Aalborg	10:30

Denne handling kan ikke fortrydes.

At the bottom of the modal, there are two buttons: "Fortryd" (light blue) and "Afbestil" (red).

Borgervejledningen

Når du er logget ind som borger, vil du møde denne side.

CitizenTaxi
Svendeprøve-projekt

Du er logget ind som Test Borger [Log ud](#)

Dine bestillinger

Du har ingen bestillinger endnu.

Bestil en ny bid

Dit notat

Bilhøjde
Alle

Hjælpebidler
Ingen

Ledsager
Alene

Følges
Nej

Bopæl
Solvej 10, 0000 God by

Er dit notat forkert?
Du kan ringe til lægesekretærene og få det rettet med nummeret:

+45 00 00 00 00

— eller —

[Ring til lægesekretærene direkte](#)

© 2023 Daniel Simonsen

CitizenTaxi af Daniel Simonsen, Datatekniker-elev h5pd111123
[Processrapport](#) [Produktrapport](#)

[Se projektet på GitHub](#)

OBS: Har du oprettet en borgerkonto fra loginsiden, har du *ikke* et notat i højre side. Notatet er nødvendigt for at kunne bestille en taxa. Derfor ser du måske denne side.

CitizenTaxi
Svendeprøve-projekt

Du er logget ind som Daniel Simonsen [Log ud](#)

Dine bestillinger

Du har ingen bestillinger endnu.

Bestil en ny bid

Dit notat

Intet notat endnu.

Er dit notat forkert?
Du kan ringe til lægesekretærene og få det rettet med nummeret:

+45 00 00 00 00

— eller —

[Ring til lægesekretærene direkte](#)

© 2023 Daniel Simonsen

CitizenTaxi af Daniel Simonsen, Datatekniker-elev h5pd111123
[Processrapport](#) [Produktrapport](#)

[Se projektet på GitHub](#)

Bestilling af taxa

Som borger kan du se dine bestillinger og bestille en ny taxatid. For at kunne bestille en ny taxatid, kan du trykke på den blå "Bestil en ny tid" knap i højre hjørne.

The screenshot shows the CitizenTaxi web application. The header includes the logo 'CitizenTaxi' with the subtitle 'Svendepreve-projekt' and a user status 'Du er logget ind som Test Borger' with a 'Log ud' button. The main content area is divided into two sections. On the left, there is a large grey box with a blue button labeled 'Bestil en ny tid'. On the right, the section is titled 'Dit notat' and contains three input fields: 'Bilhøjde' with the value 'Alle', 'Hjælpebidler' with the value 'Ingen', and 'Ledsager' which is empty.

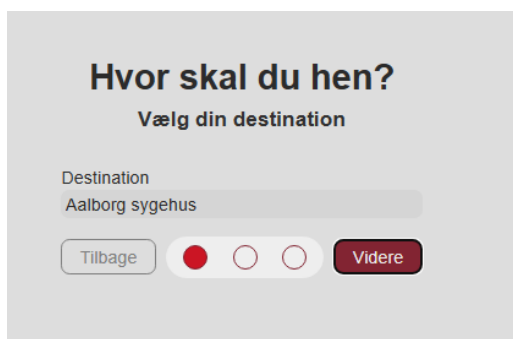
Når du trykker på knappen, bliver du viderestillet til bestillingssiden. Her præsenteres du for en 3-trins formular.

The screenshot shows the first step of the booking process. The header is the same as the previous screenshot. The main content area has a link '← Annullér bestillingsprocessen' in the top left. The title is 'Hvor skal du hen?' with the subtitle 'Vælg din destination'. Below this, there is a 'Destination' label and a text input field containing 'Frederikshavn Sygehus'. At the bottom of this section are three buttons: 'Tilbage' (disabled), a red button with a white dot (active), and 'Videre' (disabled). The footer contains copyright information '© 2023 Daniel Simonsen', project details 'CitizenTaxi af Daniel Simonsen, Datatekniker-elev h5pd111123', and links to 'Procesrapport' and 'Produktrapport'. A link 'Se projektet på GitHub' is also present.

Ønsker du at annullere bestillingsprocessen, kan du til en hver tid trykke på det blå "Annullér bestillingsprocessen" link i venstre hjørne.

A close-up of the link '← Annullér bestillingsprocessen' located in the top left corner of the booking form.

Efter du har besvaret din destination, kan du trykke på den røde "Videre" knap for at komme videre til næste trin.



På trin 2 bliver du præsenteret for dato og tid. Her skal du angive datoen, du vil hentes på, og hvornår du skal være ved din destination.



Til trin 3 indtastes din afhentningsadresse.



Når du afslutter din bestillingsformular, oprettes din bestilling under dine bestillinger, og du får en notifikation om, hvornår din taxa ankommer.

The screenshot shows the 'CitizenTaxi' web interface. At the top, it says 'Du er logget ind som Test Borger'. The main section is titled 'Dine bestillinger' and shows a single order: 'Til Aalborg sygehus, Fra Struervej 70, 9000 Aalborg' on '01/12/2023, 12:00'. There are buttons for 'Bestil en ny tid', 'Ændre bestilling', and 'Afbestil'. To the right, under 'Dit notat', there is a list of items: 'Bilhøjde', 'Alle', 'Hjælpemidler', 'Ingen', 'Ledsager', 'Alene', 'Følges', 'Nej', 'Bopæl', 'Solvej 10, 0000 God by'. Below this, a section asks 'Er dit notat forkert?' with a phone number '+45 00 00 00 00' and a link to 'Ring til lægesekretæren, direkte'. At the bottom right, a blue box says 'BEMÆRK: Din taxa ankommer om 13 timer'.

Afbestilling

Fortryder du din taxabestilling, kan du afbestille ved klik på den røde knap ved den bestilling, du gerne vil slette.

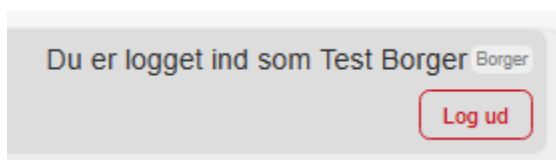
This is a close-up of the order details from the previous screenshot. It shows the date and time '01/12/2023, 16:30' and three buttons: 'Bestil en ny tid' (blue), 'Ændre bestilling' (yellow), and 'Afbestil' (red).

Når du har trykket på "Afbestil", åbnes der en afbestillingsmodal, hvor du kan bekræfte afbestillingen.

The screenshot shows a confirmation modal titled 'Du er ved at afbestille din taxa.' with a close button (X). The text asks 'Er du sikker på at du vil afbestille?'. Below this, the order details are repeated: 'Til Frederikshavn Sygehus 01/12/2023, Fra Solvej 10, 0000 God by 16:30'. A warning states 'Denne handling kan ikke fortrydes.' At the bottom, there are two buttons: 'Fortryd' and 'Afbestil' (red).

Logud af CitizenTaxi

Når du er færdig med CitizenTaxi, kan du logge ud i højre hjørne ved tryk på den røde ”Log ud” knap.



Testspecifikation

For at have et funktionelt stykke software, er det smart at teste, om det overhovedet virker – både om koden gør som forventet, og om produktets funktionalitet fungerer som kunden/kunderne forventer.

Herunder benyttes Unit- og integrationstests til test af kode og usabilitytest til test af produktet overfor en bruger.

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, om 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, som 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, således at der kan køre flere tests samtidig uden problemer.

Billedet nedenunder illustrerer, hvordan SetUp og TearDown metoderne bruges i praksis. Billedet er fra DataAcecssTest-projektet i [ABaseRepositoryTest](#) filen.

```
/// <summary>
/// This method is called before each test, re-creating in-memory database.
/// </summary>
[SetUp]
0 references
public void Setup()
{
    var context = new CitizenTaxiDbContext(
        new DbContextOptionsBuilder<CitizenTaxiDbContext>()
            .UseInMemoryDatabase(Guid.NewGuid().ToString()).Options);
    UnitOfWork = new UnitOfWork(context);
}

/// <summary>
/// This method is called after each test, disposing the UnitOfWork
/// and releasing the in-memory database and its resources.
/// </summary>
[TearDown]
0 references
public void TearDown()
{
    UnitOfWork.Dispose();
}
```

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 i Act-delen.
- 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 samme struktur med et ekstra arrange-lag efter act. Dette lag er med til at gøre assert-koden mere læselig.

Eksempelvis i [ARepositoryTest.Add](#) metoden tester jeg mine repositories' Add metode, og benytter af det ekstra arrange lag for at gøre arrangekoden lettere læseligt.

```
/// <summary>
/// Testing <see cref="BaseRepository{TEntity, TId}.Add(TEntity)"/>
/// </summary>
[Test]
0 references
public void Add()
{
    // Arrange
    var entity = new TEntity();
    var length = Repository.GetAll().Count();

    // Act
    Repository.Add(entity);

    // Save changes so entity can be retrieved by GetAll
    // Get looks in cache first, but GetAll queries the database.
    UnitOfWork.SaveChanges();

    // Arrange post act
    var gottenEntity = Repository.Get(entity.Id);
    var newLength = Repository.GetAll().Count();
    var expectedNewLength = length + 1;

    // Assert
    Assert.Multiple(() =>
    {
        Assert.That(entity.Id, Is.Not.EqualTo(Guid.Empty), "addedEntity.Id is not equal to Guid.Empty"); // The Add method assigned a value to TEntity.Id
        Assert.That(gottenEntity, Is.EqualTo(entity), "Get(entity.Id) is equal to addedEntity"); // When retrieving entity by Id, the same entity was returned from Add
        Assert.That(newLength, Is.EqualTo(expectedNewLength), "New count is equal to +length"); // 1 was added to the amount of items in the collection
    });
}
```

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 (Don't Repeat Yourself).

Læg mærke til at klasserne med A som præfiks fungerer ligesom interfaces med deres "I" præfiks. "A" præfikset er med til at illustrere, at klassen er abstrakt og bør nedarves fra en reel testklasse, som min [BookingsControllerTest](#), der nedarver fra min [ABaseControllerTest](#).

```
/// <summary>
/// Testing <see cref="BookingsController"/>
/// </summary>
0 references
internal class BookingsControllerTest : ABaseControllerTest<
    Booking,
    BookingDTO,
    BookingModifyPayload,
    BookingRepository>
{
    /// <summary>
    2 references
    protected override BaseController CreateController(UnitOfWork uow) => new
        BookingsController(uow);
    /// <summary> Point to UnitOfWork.Bookings
    15 references | 1/1 passing
    protected override BookingRepository Repository => UnitOfWorkMock.Object.Bookings;

    2 references
    [Test] public override async Task CreateEntity() => await CreateEntity
        (TestConstants.TEST_BOOKING_PAYLOAD);
    1 reference
    [Test] public override async Task GetEntities()...
    2 references
    [Test] public override async Task GetEntity() => await GetEntity
        (TestConstants.TEST_BOOKING, Booking.RELATIONS);
    [Test]
    2 references
    public override async Task UpdateEntity()...
    2 references
    [Test] public override Task DeleteEntity() => DeleteEntity(TestConstants.TEST_BOOKING);
}
```

User- & Usabilitytest

Da CitizenTaxi er inspireret af et reelt problem, som lægesekretærene i Frederikshavn står overfor, har jeg kontaktet 2 medarbejdere og 2 andre personer uden tilknytning til sygehusvæsenet, som har udført en remote user-/usabilitytest af produktet.

Alle testpersonerne var i stand til at følge opgaverne, jeg havde givet til dem, og kom med forskellige former for feedback til både hjemmesidens design og funktionalitet. Denne feedback blev taget imod, og produktet blev opdateret med de nye ændringer.