CHALMERS

Dela din resa - En samåkningsapplikation

En webbapplikation som gör det möjligt att dela och boka bilresor för samåkning

Examensarbete inom Data- och Informationsteknik

BASHAR DUMAIRIEH
JONATHAN LAWRENCE

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

EXAMENSARBETE

Dela din resa - En samåkningsapplikation

En webbapplikation som gör det möjligt att dela och boka bilresor för samåkning

BASHAR DUMAIRIEH
JONATHAN LAWRENCE

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2021

Dela din resa - En samåkningsapplikation
En webbapplikation som gör det möjligt att dela och boka bilresor för samåkning
BASHAR DUMAIRIEH
JONATHAN LAWRENCE

Examinator: Jonas Duregård

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik
Göteborg 2021

Sammanfattning

Bilar bidrar till stora mängder utsläpp och även om utvecklingen går i rätt riktning är utsläppen fortfarande ett stort problem. Många reser dessutom själva vilket ytterligare bidrar till utsläppen eftersom det då finns fler bilar i trafiken. Syftet med projektet var att skapa en webbapplikation för desktop och mobil där man kan dela och boka resor för att utnyttja tomma säten och därmed minska bilar i trafiken och dess utsläpp. Utvecklingen av webbapplikationen omfattade undersökning av utvecklingsmiljöer, ramverk och programmeringsspråk för att hitta det mest lämpliga för projektet. Med hjälp av dessa verktyg och programmeringsspråk utvecklades webbapplikation, API och Databas. Resultatet av projektet blev ett fungerande system för både mobil och dator som omfattar användarkonton samt möjligheten att skapa och boka resor. Vissa planerade funktioner lyckades inte genomföras under projektets gång så det finns fortfarande plats för förbättring, som att kunna boka delar av resor och förbättra säkerhet bland annat.

Nyckelord: Ridesharing, Samåkning, Webbapplikation, React, JavaScript, ASP.NET, C#, SQL

Förord

Innan ni ’sätter tänderna i’ och begrundar denna studie vill vi först presentera ett antal personer som har kommit att bli mycket viktiga under arbetets gång. Vi vill också tacka vår handledare Sakib Sistek som bidragit med tips och idéer och för hans konstruktiva kritik samt hans positivitet och positiva inställning till vår prestation. Ytterligare två personer som förtjänar ett stort tack är min (Bashars) pappa och min bror Bilal som har bidragit med motiverande ’glädjerop’. Stort tack till våra vänner (Bilal Alsebai, Basel Al-jazaeri, Mohammed Anas Ataya) som hjälpte mycket under arbetets gång med deras engagemang, hjälp och extraordinära stöd. Sist, men absolut det mest avgörande, är vårt fina samarbete som fortlöpt under hela studiens gång, trots yttre påfrestningar. Vi riktar härmed stort tack till varandra.

BASHAR DUMAIRIEH och JONATHAN LAWRENCE

Innehållsförteckning

1. Inledning

Nya bilar blir miljövänligare och även elbilar börjar bli vanligare. Men bilar har fortfarande en väldigt stor påverkan på miljön. Personbilar är den största orsaken till utsläpp av växthusgaser inom transportsektorn i Sverige. Många som reser med bil reser ensamma vilket ytterligare bidrar till utsläppen. Personer som reser själva är de som kräver mest energi och har störst påverkan på miljön. Trenden går också i fel riktning och fler reser själva och det blir vanligare med flera bilar per hushåll [1][2].

Ett sätt att motverka detta är att få fler att samåka när de har samma eller liknande resor. Genom att bidra med en plattform där man kan dela sina resor är förhoppningen att fler reser tillsammans för att minska antalet bilar i trafiken och minska utsläppen. En extra bonus är att det kan tillföra en social aspekt där man har någon att prata med under längre resor.

1.1. Mål

Målet är att utveckla en fungerande webbapplikation som betjänar användaren på ett enkelt sätt genom att dela deras resor med varandra. Applikationen tillhandahåller back-end funktioner genom API:er skrivna i ASP.NET core och front-end skrivna i React. Applikationen har en responsiv design så att den kan anpassa sig på olika plattformar, e.g mobil, tablet, och dator.

Några av de viktigaste funktionerna som var högst på prioritetslistan är:

- Skapa ett konto
- Användarautentisering
- Sök efter tillgängliga resor.
- Addera en ny resa.
- Boka och avboka en resa.

Applikationen ska ha högt värde till kunden genom att betjäna olika tjänster som kan hjälpa till att spara pengar genom att man kan ta en liten avgift för resan. Till exempel att alla i bilen delar på bensinkostnaden för resan. Det är också tänkt att hjälpa miljön genom att minska antalet bilar och därmed utsläpp. Appen ska också hjälpa till att minska spridningen av Covid-19 genom att hålla användaren bort från kollektivtrafiken vilket är en av de vanligaste platserna för viruset.

1.2. Syfte

Syftet med projektet är att utveckla en webbaserad applikation där man kan dela sin resa med andra om man har lediga platser i bilen eller söka efter en resa om man vill åka med någon. Genom denna applikationen är syftet att minska antalet personer som reser ensamma med bil. Detta för att både minska utsläppen och antalet bilar i trafiken genom att utnyttja tomma säten.

1.3. Frågeställningar

- Är det möjligt att bygga en webbplats som hjälper användare att dela sina egna resor med andra med hjälp av flera olika verktyg och ramverk?
- Är det möjligt att bygga registrering och inloggningssystem i ASP.NET core och koppla den med React i Javascript?
- Vilket ramverk fungerar bäst för att utveckla API för att hämta data från databas tabeller?
- Är det möjligt att lägga till en karta för att se resplanen med hjälp av Google Maps?
- Går det att utveckla funktionen att boka delar av resor med hjälp av Google Map API?

1.4. Avgränsningar

Projektet kommer inte att avse:

- Utveckling av Karta och GPS system (Existerande API kommer att användas för detta)
- Utveckling av betalningssystem

Webbplatsen kommer inte att ha en direkt betalningsmetod på grund av bristande erfarenhet av cybersäkerhet utöver svårigheten att kommunicera med olika banker och betalningsföretag som exempelvis Paypal och Swish.

2. Teoretisk och Teknisk Bakgrund

Följande kapitel går igenom teoretisk bakgrund och ger förklaring till de olika verktyg och arbetssätt som använts under utvecklingen av applikationen.

2.1. Teoretisk Bakgrund

Det finns en intressant fallstudie [3] som undersöker den långvariga effekten av samåkning i Parisregionen. Den jämför hur det såg ut 2015 med tre olika scenarier för 2030 för att ta reda om samåkning är en realistisk lösning för att minska antalet bilar i trafiken och därmed biltrafikens utsläpp. Den nämner även hur samåkning kan resultera i kortare restider på grund av den minskade trafiken. De tre scenarierna för 2030 är:

1. Ingen förändring i samåkning
2. En ökning av samåkning för både korta och långa distanser
3. En ökning av samåkning för främst långdistansresor

Resultaten och skillnaderna mellan scenarierna studeras under rusningstrafiken morgon och kväll. Resultaten av studien visar följande förändringar för respektive scenario år 2030 jämfört med 2015:

1. Trafikvolymen ökar med ca 1%, trängseln minskar något (ca 1,5%) och även koldioxidutsläppen ser en minskning på 1,5% och 2,3% för morgon respektive kväll.
2. Trafikvolymen minskar med ca 22% och trängseln minskar med ca 11% och ca 4,6% för morgon respektive kväll. Som väntat minskar också koldioxidutsläppen med ca 18% och 11%.
3. Trafikvolymen minskar med ca 29% och trängseln minskar med ca 20% och 15% för morgon respektive kväll. Koldioxidutsläppen minskar med ca 36% och ca 29% för morgon och kväll.

Den intressanta slutsatsen att dra från denna studie är att samåkning för längre resor väntas ge den största förändringen i både minskat antal bilar i trafiken och minskade koldioxidutsläpp [3].

En av de största tjänsterna för samåkning erbjuds av BlaBlaCar som fokuserar på samåkning vid längre resor. De har dock ingen verksamhet någonstans i Norden. Detta projekt fokuserar också på samåkning för längre resor mellan städer. Detta kan exempelvis vara från Göteborg till Borås eller Göteborg till Stockholm.

2.2. Programmeringsverktyg

2.2.1. Microsoft Visual Studio Community

Microsoft Visual Studio Community är en gratis IDE. Visual Studio Community erbjuder verktyg och tillägg för att skapa datorprogram liksom webbplatser, webb appar, webbtjänster och mobilappar. Visual Studio Community kan också användas för kodning med flera olika programmeringsspråk såsom C#, C++, JavaScript och Python [4].

2.2.2 ASP.NET core framework

ASP.NET core är ett gratis ramverk från Microsoft. .core förenar de föregående versioner av .Net som är ASP.NET Mvc och ASP.NET web API till en enda programmeringsmodell. I detta ramverk utvecklades alla API:er med hjälp av programmeringsspråket C# [5].

2.2.3. Microsoft SQL Server Management Studio 18

Microsoft SQL Server Management Studio 18 är en gratis mjukvara av Microsoft som används för att konfigurera, hantera och administrera alla komponenter inom Microsoft SQL Server. I denna programvara utvecklades databasen med alla tabeller och relationer mellan varandra, så att alla resor, användardata, och bokningssystem sparas och hanteras här automatiskt [6].

2.2.4. ReactJS A JavaScript library for building user interfaces

ReactJs är ett open-source bibliotek för att bygga användargränssnitt eller UI-komponenter som är utvecklat av Facebook. Appen har använt React för att bygga front-end delen och user interface som interagerar med användarens inmatade data och sen hanterar data genom att skicka den till back-end [7].

2.2.5. Visual Studio Code

Visual Studio Code är en freeware källkodsredigerare gjord av Microsoft för Windows, Linux och macOS. Front-end delen har utvecklats via denna programvara genom att installera ReactJs bibliotek [8].

2.2.6. Github

Github är en webbplats för mjukvaruutveckling och versionskontroll med Git. Det erbjuder distribuerad versionskontroll och källkodshantering. Applikationen har använt Github så att det finns en huvudbransch (master) och för varje deluppgift får en av gruppmedlemmar en ny

bransch av master för att fortsätta utveckling inom den delen av projektet. När utvecklingen är klar kan de två branscherna mergas ihop till master branschen [9].

2.3. Projekthanteringsverktyg

2.3.1 Scrum

Scrum metoden tillämpades för att implementera Agile processen. Scrum metoden tillämpas genom att anordna möten, avgöra vilka roller som används, tilldela dessa roller, och vilka verktyg som skall användas i projektet. I slutändan får projektet en strukturerad form och gör det lättare att hantera arbetet.

De vanligaste rollerna som används inom scrum metoden:

- Projektledare (den person eller det företag som äger projektet)
- Scrum Master är den person som är ansvarig för att dela upp uppgifter mellan gruppmedlemmar och organisera det som ska göras till nästa möte. Uppgifterna skrivs på papper eller i ett onlineverktyg e.g Trello eller Clubhouse. Scrum Master skriver också upp allt som händer under möten.

2.3.2 Trello

Trello är ett online verktyg för att skapa olika listor och används för att visualisera arbetsbördan inom gruppen. Olika listor kan enkelt skapas som gör det möjligt för gruppmedlemmar att följa nuvarande uppgifter och framstegs nivåer, uppgiftens slutdatum, kommande och avklarade uppdrag [10].

2.3.3 Discord

Discord är en VoIP (Voice over IP) plattform för snabbmeddelanden och röstkommunikation. Användare kan kommunicera via röstsamtal, videosamtal, textmeddelanden och skicka media och filer. Antingen i privata chattar eller som en del av grupper som kallas "servrar" [11].

2.3.4 Designmönstret Modell-Vy-Controller (MVC)

Det här mönstret hjälper till att separera det grafiska användargränssnittet från logik och bearbetning inom applikationen. Med andra ord tillhandahåller MVC-mönstret separationen mellan gränssnittet och själva programmet, så att om logiken ändras förblir vyn oförändrad och vice versa om gränssnittet (vyn) ändras förblir modellen oförändrad. Kontrollern spelar huvudrollen i detta mönster eftersom alla operationer (oavsett om man visar, matar in eller

korrigerar information) styrs av kontrollern innan den skickar datan till modellen. När modellen tar emot information bearbetas den och skickas tillbaka till kontrollern, som i sin tur skickar den till vyn för visningen.

Detta mönster användes i både front-end och back-end där modellen och kontrollen applicerades på back-end delen av programmet och vyn på front-end delen. Front-end delen interagerar med användaren och skickar bearbetnings förfrågningar till kontrollern via API:n där den hanterar data med hjälp av modellen. Sen skickar den resultaten till front-end. Alltså blir kontrollern länken mellan vyn och modellen.

3. Metod

I det här kapitlet redogörs för hur projektet har hanterats och vilka projektmetoder som har använts. Det går också igenom vilka verktyg som använts för utvecklingen av applikationen och hur hela projektet har dokumenterats. Till sist så beskrivs hur applikationen har testats för att se om den uppnått målen.

3.1. Agile Project Management

Projektet utfördes med hjälp av scrum metoden som är en del av agile project management. Man delar upp projektarbetet i sprints där en sprint kan vara en vecka och sen utför man det planerade arbetet under den sprinten. Det kommer att utföras med hjälp av trello som scrum board där uppgifterna läggs upp. Innan varje sprint så tilldelas uppgifterna och under projektets gång så flyttar man uppgifterna mellan olika kategorier. Kategorierna kan vara backlog, att göra, under utveckling och klart. Kommunikation mellan studenterna skedde online, skriftligt och muntligt via Discord och Facebook Messenger. I början testades korta dagliga möten men det ansågs inte nödvändigt eftersom man inte hann med så mycket på bara en dag. Hade det varit en större grupp hade dagliga möten kunnat ge mer men i och med att det bara är två personer var det lätt att hålla koll på vad den andra jobbade med. Istället hölls regelbunden skriftlig kontakt och extrainsatta möten vid behov. I slutet av varje vecka planerades ett muntligt möte. Under det mötet utvärderades sprinten och planering och tilldelning av uppgifter för nästa sprint påbörjades.

3.2. Utvecklingsmiljö

IDE:n Visual Studio Code användes för front-end utvecklingen. Webbapplikationen kodades med React som är ett javascript library. GitHub användes som repo och det kopplades till Visual Studio Code med Git. Varje vecka eller efter behov så skapades nya branscher på Github och sedan mergas de med master igen i slutet av veckan beroende på hur lång tid utvecklingen av en bransch tog. För back-end var planen att använda antingen Node.js eller Spring Boot men efter vidare undersökning valdes istället ASP.NET.

3.3. Dokumentation

All dokumentation under projektets skedde på Google Drive där det finns ett dokument som används som dagbok där det skrevs vad som har jobbats på under en dag eller vecka. Sedan finns det ett dokument för rapporten som skrevs under projektets gång. Google dokument har inte lika många funktioner som till exempel word men används istället för Word online för att

det är lättare och mer effektivt att använda vid samarbeten eftersom Word online kan vara långsamt.

3.4. Testning

För att undersöka om målen för projektet uppnåtts så utfördes flera tester på applikationen för att testa dess funktionalitet. Testerna gjordes så att de efterliknar processen om hur en användare skulle använda applikationen. Testerna som utfördes var att använda navigeringsmenyn, skapa konto, logga in, söka och skapa resor, boka och avboka resor. Hela applikationen testades också med olika skärmstorlekar för att se om den uppfyller målet om att vara responsiv på flera olika plattformar.

4. Genomförande

Arbetet delades in i tre delar, databas, front-end och back-end. I det här kapitlet redogörs hur var och en av dessa delar fungerar, och hur de olika delarna kopplas ihop för att slutföra sändning, lagring och bearbetning av förfrågningar.

4.1. Planering

Målet med projektet är att skapa en lösning för att minska utsläpp från bilar. Lösningen var i form av en webbapplikation där man skapar en resa för att dela de tomma säten med andra. Planen var att göra en webbapplikation med responsivt user interface med hjälp av react för att göra det lättare att öppna applikationen på olika plattformar. Första planen var att utveckla applikationen med de viktigaste egenskaperna: lägg till resa, boka ett säte och avboka. Den andra planen var lägga till fler funktioner med tiden som sign in system, använda google map API och ett notifikations system. På grund av den rådande situationen med Covid-19 så bestämdes att arbetet skulle utföras online genom olika online mjukvaror som angivits tidigare.

I början var planen att utveckla front-end med hjälp av de grundläggande web språken CSS, HTML och Javascript, men frågan diskuterades och React Library valdes istället. Erfarenheten av React var liten och att starta ett relativt stort projekt med detta bibliotek skulle kräva mycket arbete. Men på grund av bibliotekets popularitet, all dokumentation som finns på nätet och en villighet att lära sig React, godkändes det för att utföra arbetet. För back-end identifierades två ramverk med tillhörande programmeringsspråk som var möjliga kandidater för utvecklingen. Kandidaterna var Spring Boot som använder Java och ASP.NET som använder C#. För att hitta det bästa alternativet planerades en utförligare undersökning för att se om det fanns några komplikationer eller fördelar med något av ramverken.

Tiden delades in i två faser, den ena för att utveckla front-end och databastabeller, den andra för att utveckla back-end och ansluta den till front-end. Under de första två veckorna hämtades alla program och bibliotek som krävdes för att slutföra arbetet, sedan skapades databastabellerna ursprungligen med grundläggande relationer mellan dem. Därefter startade processen med att skapa den preliminära designen av front-end, som modifierats under utvecklingens gång.

4.2. Front-end utveckling

Front-end utvecklingen genomfördes i tre delar. Först gjordes en grundläggande design med hjälp av wireframing, sedan implementerades en liknande design i react och tillslut byggdes det på och ändrades i den implementerade designen. I det här avsnittet visas bara den första

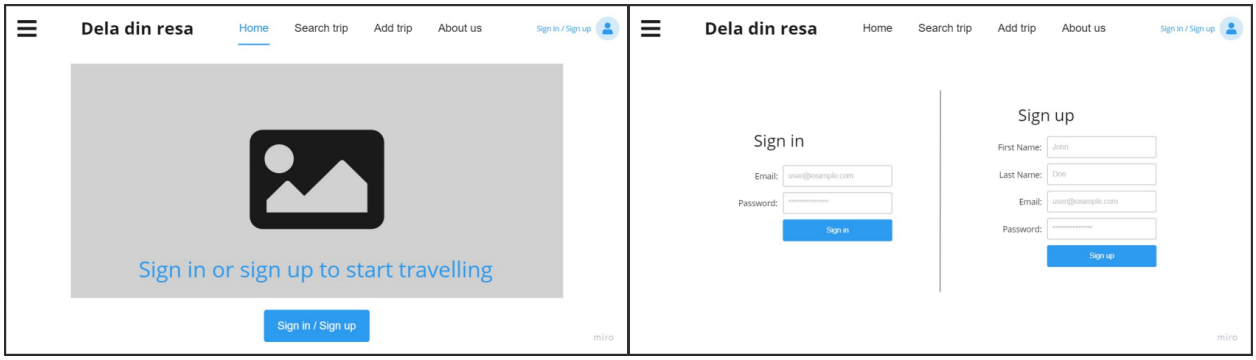
wireframe-designen och den slutgiltiga designen. Den första implementationen var lik den slutgiltiga men med färre funktioner och mindre förfinad.

4.2.1. Första design

En viktig del i projektet är rollerna som användare av applikationen kan ha. En användare kan ha två roller. Den första rollen är förare och är den som skapar en resa. Föraren bidrar med information såsom avgång och destination, datum och tid, antalet platser, pris och övriga kommentarer. Övriga kommentarer kan vara till exempel exakt startplats, typ av bil eller annan eventuell information som kan vara viktig för passagerare att veta. Den andra rollen som en användare kan ha är passagerare. Passagerare kan söka efter resor med filter och antal platser som krävs och sedan boka dem. Utöver det behöver inte passageraren bidra med någon mer information. Det är också viktigt att notera att en användare kan ha både rollen som förare och passagerare för olika resor. En användare kan välja att vara förare för en resa och sedan passagerare för en annan beroende på vilket alternativ som passar bäst för användaren vid varje situation.

Innan utvecklingen började gjordes en grundläggande design med hjälp av wireframing i Miro. Wireframe-designen hjälpte för att få en ungefär idé om hur designen skulle se ut och samtidigt bestämma vilka funktioner som behövdes för att uppnå de mål som sattes för applikationen. Applikationens interface designades på engelska för att inkludera så många potentiella användare som möjligt. Även om applikationen är tänkt att endast användas i Sverige kan den komma att användas av turister eller internationella studenter och arbetare. Ett framtida mål för utvecklingen skulle vara att erbjuda applikationen på både Svenska och Engelska.

Först designades en startsida som välkomnar användare och som hänvisar vidare till en inloggningssida. Navigeringsfältet i toppen är del av alla sidor för att man lätt ska kunna navigera till och från var som helst på webbplatsen. En av de viktigaste funktionerna var att ha ett system för att kunna skapa ett konto och logga in. Dels för identifiering, att användaren kan se sina resor, och för att koppla användare till resor som de skapar eller bokar.



Figur 1. Wireframe design för Startsida och Formulär.

Att kunna söka, boka och skapa resor är den viktigaste funktionen och huvudidén med projektet. Här kan man söka med olika kriterier och sedan få en lista på de resor som matchar, sen kan man boka den resa man vill. Om man istället har en egen resa som man vill dela med andra kan man göra det genom att fylla i detaljerna för resan. För att boka eller skapa resor behöver man ha ett konto.

En skapad resa innebär alltså att en person som ska köra någonstans delar med sig av den resan och användaren är då föraren för den resan. En bokad resa innebär att användaren har bokat en eller flera platser på någon annans resa och är därmed passagerare i den resan.

När man är inloggad ska man kunna se de resor som man antingen har skapat eller bokat på sin sida. Där kan man se all information man behöver såsom datum, tid, antal platser och avgångs- och slutdestination.

Figur 2. Wireframe design för Skapade och Bokade resor.

De sista sidorna från den första designen är “About us” och en kontosida. Under “About us” ska man kunna läsa om applikationen och få svar på eventuella frågor. På sitt konto kan man hitta sina bokade och skapade resor samt se annan information om sitt konto och logga ut.

Full wireframe design kan hittas i Bilaga 1.

4.2.2. App.js (Main)

App.js är main funktionen för applikationen. Den samlar alla komponenter som ska visas och här bestäms också när komponenterna ska visas och vilka sidor som är tillgängliga. De olika sidorna har en url-ändelse som kan användas för att komma åt sidorna. Search trip har till exempel ändelsen /searchtrip. Men eftersom bara någon som är inloggad ska kunna komma åt den och några andra sidor så finns det restriktioner i App.js som begränsar vilka sidor som går att komma åt utan att logga in.

React-router-dom

För att kunna ha flera sidor i samma webbapplikation så används biblioteket react-router-dom. Detta biblioteket gör det möjligt att tilldela url ändelser till olika komponenter. Route används för att tilldela en url och välja vilka komponenter som ska visas för den url:en. När till exempel användaren klickar på länken till “Home” i navigeringsmenyn länkar den till en url och när Route ser att den matchar en url så laddas de komponenterna in, i detta fallet Startsidan.

```
<Route exact path="/home">
  <Home shownotification={setNotificationMessage}/>
</Route>
```

Figur 3. Ett exempel på hur Route används för att tilldela en url ändelse och välja komponenter att ladda in.

4.2.3. Navigation Bar

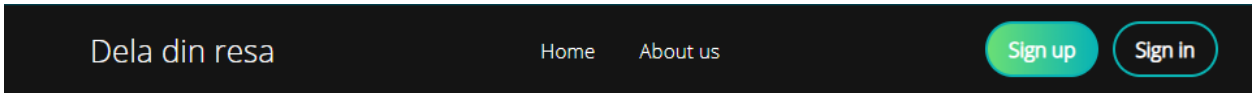
Navigeringsmenyn är alltid synlig längst upp i fönstret för att underlätta navigering från alla olika sidor. Navigeringsmenyn består av flera komponenter. Det finns en komponent för varumärke och den har också en container för att lägga till en eventuell logo. Det finns också en komponent som innehåller knappar som används på navigeringsmenyn. Sedan finns det två komponenter som har hand om navigerings länkarna, en för desktop och en för mobil beroende på storlek av fönstret. För att se storleken på fönstret så används useMediaQuery från react-responsive paketet där man kan specificera vid vilken storlek som fönstret ska gå över till mobilanpassat läge. Alla fönsterstorlekar under 768 pixlar ses som mobil. Sedan används isMobile för att anpassa vad som visas.

```
const isMobile = useMediaQuery({ maxWidth: 768})
```

Figur 4. useMediaQuery hämtar fönsterstorleken och kollar om det matchar argumentet. Om det gör det så är isMobile sant.

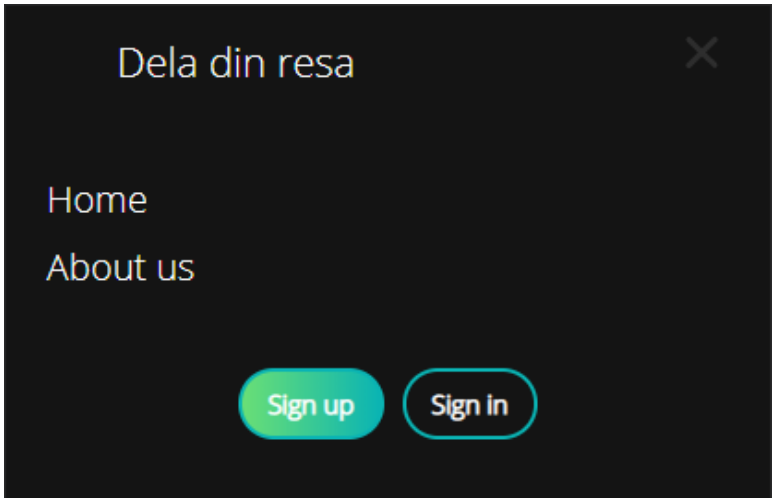
Huvudkomponenten “NavigationBar” samlar alla andra komponenter och placerar dem i containrar så att menyn får en bra struktur och så att komponenterna hamnar på rätt ställe. Det finns tre containers, en till vänster, en i mitten och en till höger. Den vänstra innehåller varumärket och är fäst längst åt vänster. I mitten finns en flexibel container som är centrerad mellan de två yttre och den har hand om de olika länkarna som man kan använda för att navigera på sidan. Om man håller musen över en länk visas ett streck högst upp som en indikation. Den högra containern innehåller knapparna som används för att komma till sidor som relaterar till användarkonton. Knapparna fylls i om man håller musen ovanför.

Om användaren inte är inloggad så visas länkarna “Home” och “About us” och två knappar, en för att registrera sig och en för att logga in. Om användaren är inloggad så finns ytterligare två länkar i menyn, “Search trip” och “Add trip”, som används för att komma till huvud tjänsterna på applikationen. Så man behöver ha ett konto och vara inloggad för att kunna använda tjänsterna som erbjuds eftersom bokningar och resor är kopplade till ett konto. Är man inloggad så byts knapparna ut mot en ny som tar en till sin kontosida.



Figur 5. Navigeringsmeny för desktop.

På navigeringsmenyn för mobil är bara varumärket synligt från början. Med ett knapptryck så kan man expandera menyn och där finns samma innehåll som i den vanliga menyn fast staplat vertikalt.



Figur 6. Navigeringsmeny för mobil.

4.2.4. Card Design

För att ha en sammanhängande design för hela applikationen så använder alla komponenter som kräver input samma grunddesign. Designen består av två javascript filer som innehåller flera komponenter. I “FormBase” finns alla komponenter som relaterar till grunden för kort designen. I “FormComponents” finns de komponenter som designar alla input varianter. Komponenterna är kodade med hjälp av biblioteket “styled components” för react. Sedan kan de komponenterna som behövs importeras för att skapa de olika input formulären.

Styled components möjliggör användningen av CSS direkt i .js filer utan att behöva använda css filer eller react inline styles. Detta görs genom att skapa en const, namnge den och välja vilket element man vill anpassa och sen använda de css egenskaper man vill ha. Så istället för att ha en css fil och sedan använda designen genom att lägga till className till ett element så kan man använda elementet direkt. Antingen kan man ha alla sina styled components i samma fil och använda direkt, eller så kan man importera till andra filer som gjordes med “FormBase” och “FormComponents” eftersom de används på flera olika ställen.

```
export const HeaderContainer = styled.div`
    width: 100%;
    display: flex;
    flex-direction: column;
`;

export const HeaderText = styled.h2`
    font-size: 30px;
    font-weight: 600;
    line-height: 1.24;
    color: white;
    z-index: 10;
    margin: 0;
`;
```

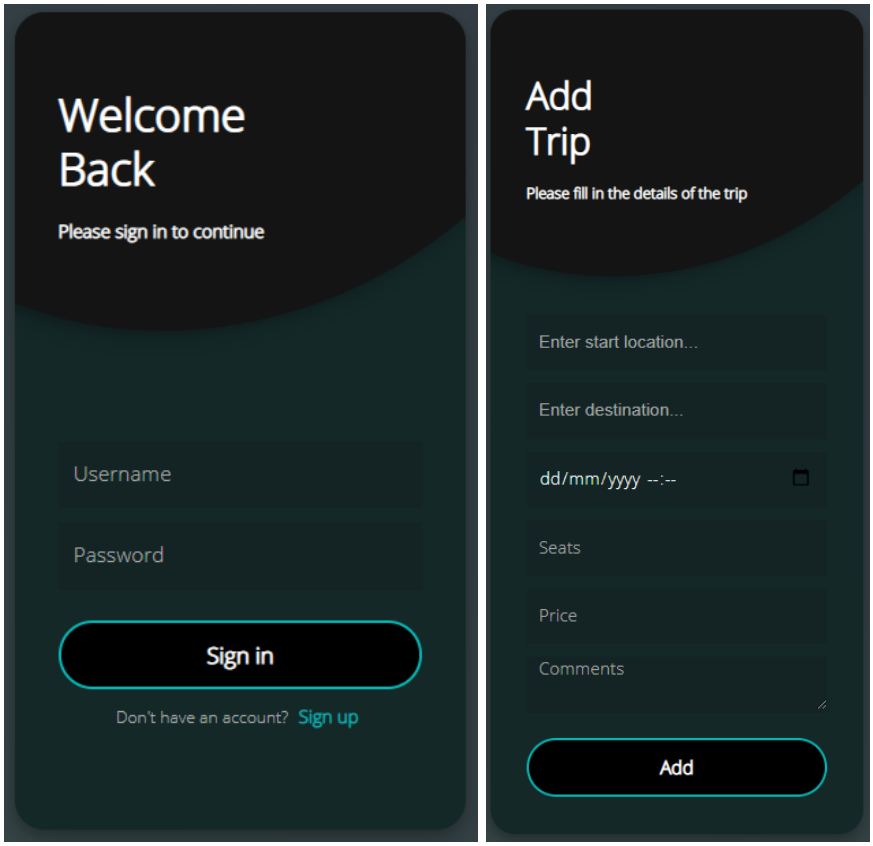
Figur 7. Exempel på hur styled components skapats och designats i applikationen.

```
<HeaderContainer>
    <HeaderText>Welcome</HeaderText>
    <HeaderText>Back</HeaderText>
    <SmallText>Please sign in to continue</SmallText>
</HeaderContainer>
```

Figur 8. Exempel på hur styled components har använts för inloggningssidan i applikationen.

Designen för grunden är flexibel så att den kan användas till alla input formulären. Den har en minimum höjd och kan expanderas på höjden om input fälten kräver mer plats.

FormBase har en header och subheader för titel och eventuell subtitel. Den innehåller också en container för formulären. FormComponents sköter stilen för inputs, till exempel placeholders och effekter om musen hålls över objekt och om ett objekt är i fokus.



Figur 9. Exempel på hur formulären är designade och anpassar sig efter antalet fält.

Sign in/Sign up

För registrering och inloggning finns obligatoriska fält som måste fyllas i. När det är ifyllt och man klickar på knappen skickas informationen till back-end som sparar användarinformation i databasen eller genomför autentisering för inloggning.

Search trip/Add trip

För att söka och lägga till resor fyller man i detaljerna för resan man vill söka eller skapa. Om en resa skapats så processeras det i back-end och den blir tillgänglig för att söka och boka. Om sökfunktionen används skickas detaljerna till back-end som filtrerar resorna och man kommer till nästa sida där man kan välja att boka en resa. När man bokar en resa skickas ett mail till användaren med information om resan.

My page

Konto sidan har profilbilden, förnamn och efternamn i headern. Det finns också fyra knappar för konto relaterade funktioner. Två av knapparna är för att komma till sina bokade eller skapade resor där man sedan kan hantera sina resor, till exempel avbokningar. En knapp är för att

hantera sina kontoinställningar. Denna funktionen är inte utvecklad ännu eftersom den ansågs ha lägre prioritet än många av de andra funktionerna.

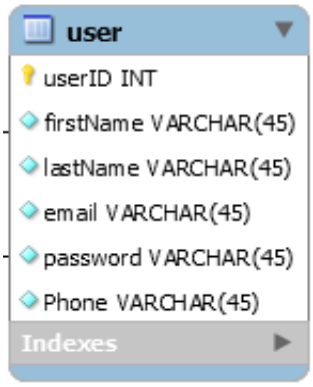
4.2.5. Bokade och skapade resor

Från kontosidan kan man komma till sina bokade och skapade resor. Där kan man se all information om resan såsom datum och destination. Bokade resor kan avbokas genom att klicka på en knapp och användaren avregistreras från resan. Även skapade resor kan tas bort, isåfall skickas ett mail till alla användare som har bokat den resan.

Karta
Innan man bokar en resa så kan man klicka för att se en karta med den snabbaste vägen från start till destination. Kartan visas som en popup och den hämtas med hjälp av googles api.

4.3. Databasutveckling

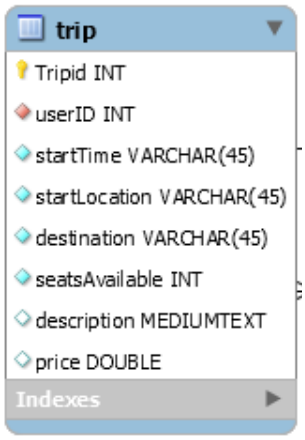
I början av utvecklingsprocessen diskuterades de scheman som krävs för projektet. I princip behöver projektet två bastabeller, användaren och resorna. Användarnas tabell sparar den grundläggande informationen om användarna liksom: first name, last name, email, password, phone och UserId som används för att skapa relationer och förhindra duplicerade fält.



Figur 10. Bilden visar hur första versionen av User tabellen såg ut i databasen.

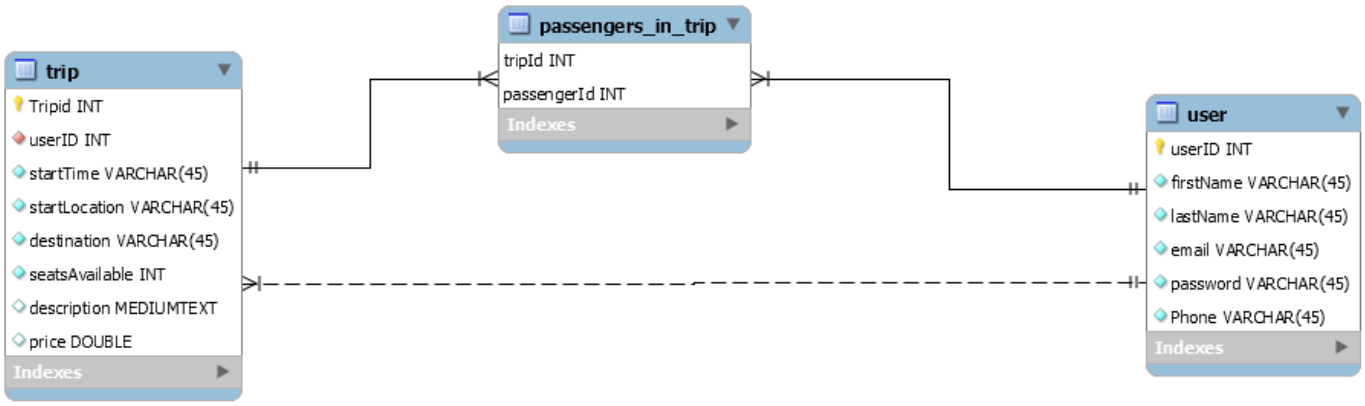
Tabellen för resor sparar följande information:
Start Time: Avgångstid
StartLocation: Avgångsplats

Destination: Resans ändhållplats
Price: Priset för ett säte
Seats available: Antal tomma säten som går att boka. Minskar om ett eller flera säten bokas av användare.
Description: Förarens kommentar om resan liksom, ej djur, rökning eller ingen barnstol i bilen
TripId: Skapa relationer och förhindra duplicerad resa
UserId: Förarens Id



Figur 11. Bilden visar hur första versionen av trip tabellen såg ut i databasen.

Relationen mellan de två tabellerna är “One-to-many”, så att passageraren kan boka flera resor, och resan kan ha många resenärer, men passageraren kan inte boka samma resa två gånger. För att bygga denna relation behövs det en extra tabell för att bryta den cirkulära relationen mellan de två tabellerna och inte få upprepad information.



Figur 12. Bilden visar hur relationerna mellan tabellerna såg ut i den första versionen.

Passengers_in_trip tabellen anses som boknings tabell där det finns två huvudsakliga fält:
TripId: Kopplad till Tripid i trips tabellen
PassengerId: Kopplad till UserId i user tabellen
Så när bokning sker så måste både resenären och resan finnas i tabellerna innan annars går det inte att lägga till bokningen.

När utvecklingsprocessen gick fram många steg upptäcktes några problem i utformningen av databastabellerna. Fälten i tabellerna var otillräckliga eller gjorde inte de nödvändiga uppgifterna. Till exempel kan man helt enkelt se att användarnas tabell inte har någon lösenordskryptering, och att det är möjligt att kapa lösenordet i “plain text” om webbplatsen blir hackad. Därför har många ändringar gjort på user tabellen och många ytterligare fält har lagts till:

Normalizedemail: För att omvandla användarens e-mail till uppercase oavsett om användaren har matat in email i lower eller uppercase. Det här fältet är användbart för att undvika skrivfel när förfrågningar skickas till databasen

EmailConfirmed: Det här fältet deklarerar om användaren har verifierat sin e-post vid registrering eller inte, det har värdena 0 eller 1.

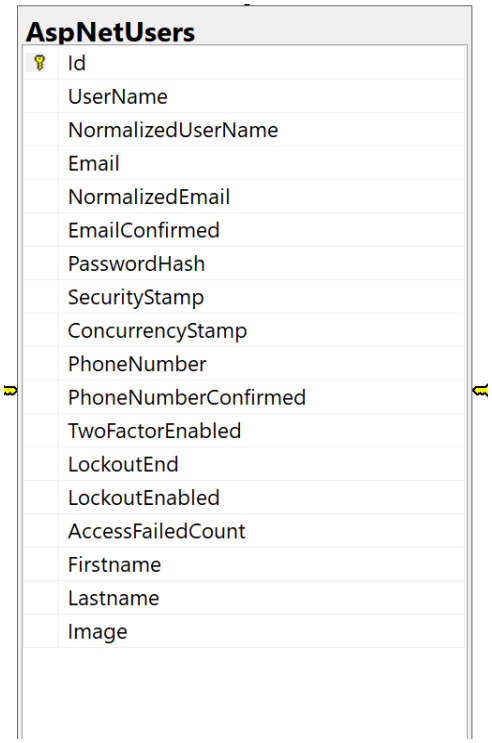
PasswordHash: Det innehåller lösenordskryptering, där ingen kan få lösenordet även om krypteringen fås, eftersom krypteringen är gjord med hjälp av en oerhört komplex funktion som kallas “one way hash”. Det innebär att krypteringen bara kan erhållas från texten och inte tvärtom. Projektet byggdes med hjälp av ASP.NET core v3, vilket innebär att ASP.NET krypterar lösenordet med PBKDF2 algoritm. PBKDF2 använder HMAC-SHA256 med 128-bit salt, 256-bit subkey och 10 000 iterationer [12].

SecurityStamp, ConcurrencyStamp: Dessa två fält läggs till automatiskt av Microsoft-biblioteket vid registreringsprocessen för att öka cybersäkerhet.

PhoneNumberConfirmed: Det här fältet deklarera om användaren har verifierat sitt telefonnummer vid registrering eller inte, det har värdena 0 eller 1.

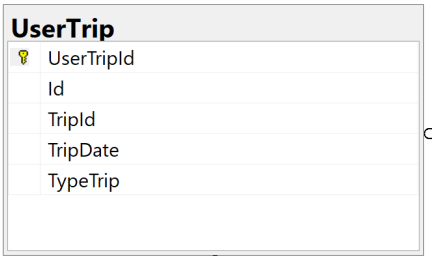
TwoFactorEnabled: Det här fältet anger om den säkra inloggningsprocessen har aktiverats eller inte. Med denna processen måste användaren verifiera sin identitet via e-post vid sidan av en bekräftade åtkomstkoden via telefon.

AccessFailedCount: Antalet felaktiga inloggning bestäms för varje användare, om detta antal överskrids vidtas säkerhetsåtgärder.



Figur 13. Bilden visar hur User tabellen ser ut i databasen.

I "passenger-in-trip" tabell finns det ingen skillnad mellan resenären och föraren och vid utveckling i back-end ledde detta till oväntade resultat när någon filtrering gjorts. Det är därför några ytterligare fält har lagts till för att underlätta filtering i back-end.



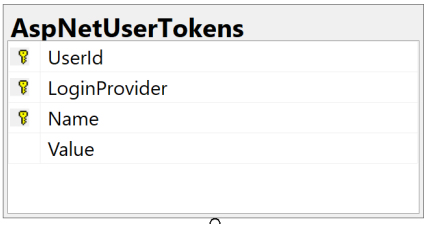
Figur 14. bilden visar hur UserTrip tabell ser ut i databasen

Id: Kopplat till Id i Asp.NetUsers.
TripId: Kopplat till Id i Asp.NetTrips.
Tripdate: Bokningsdatum

TypeTrip: Det är antingen “Driver” eller “Passenger” för att enkelt bestämma om användaren är resans förare eller passagerare.

Utöver det, när inloggningssystem lagts till fanns det ingen tabell som sparar token-koden som används i alla operationer. Token-koden är en speciell kod för användaren som normalt ges vid inloggningsprocessen för att ha befogenhet att utföra alla operationer, oavsett om man söker, lägger till eller går in på den personliga sidan. Utan token-koden kan ingen av de operationerna utföras.

Utgångstiden ställs in på back-end. när någon åtgärd utförs bifogar applikationen token för den inloggade användaren med begäran som ska bearbetas. Begäran avslås när token fattas, så det måste finnas en tabell som sparar token-koder för inloggade användaren för att bifoga den med begäran.

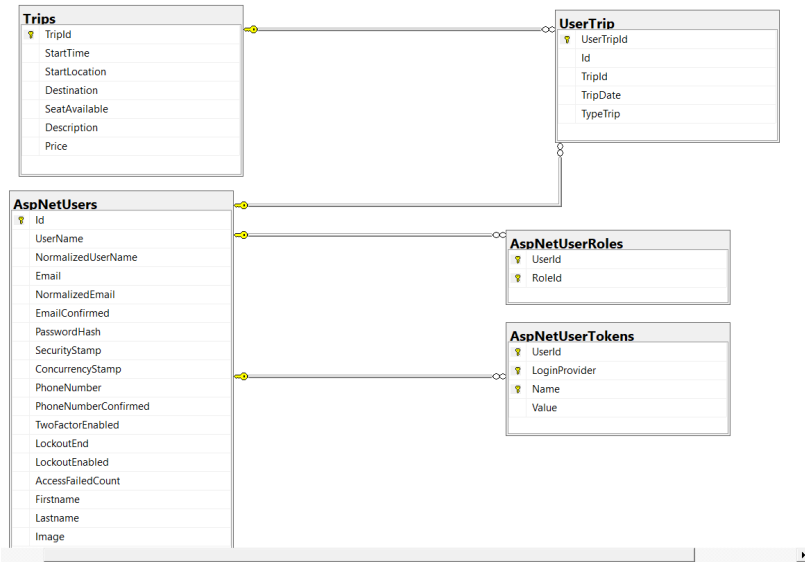


Figur 15. Bilden visar hur UserToken tabellen ser ut i databasen

När det gäller användarroller finns det två huvudroller, antingen en användare eller en administratör. Var och en har sina egna befogenheter. Administratören kan avboka en viss resa för en viss användare, ta bort användarkonton, starta och stoppa vissa tjänster på webbplatsen. Användaren kan bara utföra de normala tjänsterna: lägga till en resa, sittplatsreservation och avboka en resa.

Figur 16. Bilden visar hur UserRoles tabellen ser ut i databasen.

Relationer mellan tabellerna blir enligt följande:

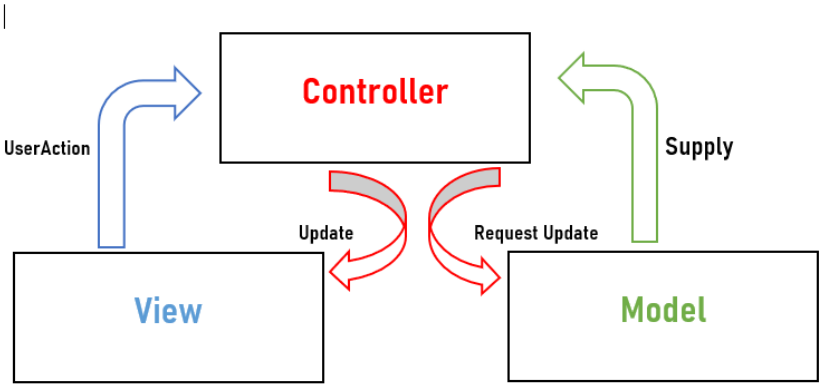


Figur 17. Bilden visar de nya relationerna mellan tabellerna.

4.4. Back-end utveckling

4.4.1. Designmönstret MVC

För att uppnå enklare utveckling och underhåll på webbplatsen används detta mönster som hjälper till att separera användargränssnittet från logiken och modellen som används i applikationen. Detta mönster ger enkel testning, utveckling och underhåll av applikationen i framtiden.



Figur 18. En bild på hur kommunikationen i designmönstret MVC fungerar.

Kontrollen och modellen utvecklades i ASP.NET Core framework och Vyn utvecklades i React-framework med hjälp av JavaScript. Där vyn interagerar med de olika användar instruktionerna och begär en lämplig API från kontrollen, som i sin tur beräknar och hanterar de olika modeller och skickar resultatet till vyn igen.

4.4.2. Modell

I detta projekt finns det åtta modeller. Dessa modeller är beskrivna nedan och används både när användaren begär olika funktioner i front-end.

ApplicationDbContext
Modellen är ansvarig för att länka databastabeller till ASP.NET framework.

```
services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("ConnStr")));
```

Figur 19. Bilden visar hur instanserna av ApplicationDbContext skapas.

I startup.cs filen lägger vi till modellen med den deklarerade databasplatsen i “Connstr”. Connstr definieras i appstring.json där databasplatsen och resten av anslutningsinställningarna inställs.

```
"AllowedHosts": "*",
"ConnectionStrings": {
  "ConnStr": "data source=.;initial catalog=Test;integrated security=True;Connection Timeout=60;
},
```

Figur 20. Bilden visar hur ConnectionString initieras.

ApplicationUser
Den här modellen representerar användarens tabell i DataBas (Asp.NetUsers) för att hämta eller lagra nya användare. Så modellen representerar tabellens fält och relationer till andra tabeller för att lagra det ordentligt.

```
public class ApplicationUser: IdentityUser
{
    public ApplicationUser()
    {
        UserTrips = new HashSet<UserTrip>();
    }
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    [Display(Name="Profile Image")]
    public string Image { get; set; }
    [NotMapped]
    public IFormFile ImageFile { get; set; }
    [NotMapped]
    public string ImageSrc { get; set; }
    [NotMapped]
    public int BookedTrips { get; set; }
    [NotMapped]
    public int CreatedTrips { get; set; }
    public virtual ICollection<UserTrip> UserTrips { get; set; } //Relation med UserTrips
}
```

Figur 21. Kodsnutten visar hur ApplicationUser initieras.

LoginModel

Representerar inloggningsprocessen för användaren.

```
public class LoginModel
{

    [Required(ErrorMessage = "User Name is required")]
    public string Username { get; set; }

    [Required(ErrorMessage = "Password is required")]
    public string Password { get; set; }
}
```

Figur 22. Kodsnutten visar hur LoginModel initieras

UserRole

Den innehåller de roller som användaren kan representera.

```
public static class UserRoles
{
    public const string Admin = "Admin";
    public const string User = "User";
}
```

Figur 23. Kodsnutten visar hur UserRoles initieras

RegisterModel

Den representerar de fält som krävs från användaren för att registrera sig på webbplatsen.

```
public class RegisterModel
{
    [Required(ErrorMessage = "FirstName is required")]
    public string Firstname { get; set; }

    [Required(ErrorMessage = "Last Name is required")]
    public string Lastname { get; set; }

    [Required(ErrorMessage = "User Name is required")]
    public string Username { get; set; }

    [EmailAddress]
    [Required(ErrorMessage = "Email is required")]
    public string Email { get; set; }

    [Required(ErrorMessage = "Password is required")]
    public string Password { get; set; }

    public string Image { get; set; }

    public IFormFile ImageFile { get; set; }
}
```

Figur 24. Kodsnutten visar hur RegisterModel initieras

Trip

Den här modellen representerar tabellen för resor i DataBas (Asp.NetTrips) för att hämta eller lagra nya resor. Så modellen representerar tabellens fält och relationer till andra tabeller för att lagra det ordentligt.

```
public class Trip
{
    public Trip()
    {
        UserTrips = new HashSet<UserTrip>();
    }
    [Key]
    public int TripId { get; set; }
    [Column(TypeName="nvarchar(50)")]
    public string StartTime { get; set; }
    [Column(TypeName = "nvarchar(50)")]
    public string StartLocation { get; set; }
    [Column(TypeName = "nvarchar(50)")]
    public string Destination { get; set; }
    public int SeatAvailable { get; set; }
    [Column(TypeName = "nvarchar(500)")]
    public string Description { get; set; }
    public decimal Price { get; set; }
    public virtual ICollection<UserTrip> UserTrips { get; set; } //Relation till UserTrips

}
```

Figur 25. Kodsnutten visar hur Trip initieras.

UserTrip

Den här modellen representerar UserTrips tabellen i DataBas (Asp.NetUserTrip) för att hämta eller lagra nya bokningar. Modellen representerar tabellens fält och relation till andra tabeller för att boka resor ordentligt.

```
public class UserTrip
{
    [Key]
    public int UserTripId { get; set; }
    public string Id { get; set; }
    public int TripId { get; set; }
    public DateTime TripDate { get; set; }
    public string TypeTrip { get; set; }
    public virtual ApplicationUser User { get; set; }
    public virtual Trip Trip { get; set; }


}
```

Figur 26. Kodsnutten visar hur UserTrip initieras.

Response

Denna modell representerar responsen som skickas från back-end till front-end, bestående av två fält, status och meddelandefältet. Till exempel kan status vara 404(not found) och error meddelandet kan bifogas med till ex “can't retrieve page”.

```
public class Response
{
    public string Status { get; set; }
    public string Message { get; set; }
}
```

Figur 27. Kodsnutten visar hur Response initieras.

4.4.3. Services

I projektet används olika klasser som hjälper till med extra funktioner liksom skicka ett mail, spara inloggade Userid osv. Klasserna är beskrivna nedan som följande:

EmailConfig

Den här klassen används för att ladda upp konfigurationsinformation till e-post, vilket tydligt identifieras i appsettings.json filen. Där definieras avsändarens e-post, ports nummer och användarnamn som följande:

```
"EmailConfiguration": {
  "From": "dimariabashar@gmail.com",
  "SmtpServer": "smtp.gmail.com",
  "Port": 465,
  "Username": "**************",
  "Password": "**************"
},
```

Figur 28. Kodsnutten visar hur Email inställningar konfigureras.

Den informationen laddas bara ner en gång under appens livstid (med hjälp av singelton) via EmailConfig klassen när appen startas för att kunna skicka boknings och avbokningsbekräftelse.

Message

Denna klass används för att definiera meddelandets form, bestående av en avsändare, en grupp av mottagare, titeln och innehållet.

```
public class Messege
{
    public List<MailboxAddress> To { get; set; }
    public string Subject { get; set; }
    public string Content { get; set; }
    public Messege(IEnumerable<string> to, string subject,string content)...
}
```

Figur 29. Kodsnutten visar hur Message initieras.

EmailSender

Klassen är ansvarig för att skicka meddelandet till mottagaren genom att skapa Smtpclient och kontakta avsändarens server med hjälp av EmailConfig klassen. För att sändnings processen ska kunna slutföras rätt måste meddelandet konverteras till rätt format innan det skickas. Den här klassen konverterar det vanliga meddelandet (Message) till ett meddelande som enkelt kan skickas via klienten (MimeMessage) via CreateEmailMessege metoden.

```
public class EmailSender : IEmailSendercs
{
    private readonly EmailConfig _emailComfig;
    public void SendEmail(Messege messege)...
    private void Send(MimeMessage emailMessege)...
    private MimeMessage CreateEmailMessege(Messege messege)...
    public EmailSender(EmailConfig emailConfig)...
}
```

Figur 30. Kodsnutten visar hur EmailSender initieras.

FormatMessage

Den här klassen används för att skapa ett standardiserat innehåll för boknings- eller avbokningsmeddelanden. Klassen är statisk som kan användas direkt i vilken del som helst av programmet, den använder trips informationen och returnerar ett boknings- eller avbokningsmeddelande enligt den kallade metoden.

```
public static class FormatMessage
{
    public static string BookningFormat(Trip trip)...
    public static string CancelTrip(Trip trip)...
}
```

Figur 31. Kodsnutten visar hur Format initieras och vilka metoder som den har.

UserService

För att användaren ska kunna boka, avboka eller utföra någon process inom applikationen måste den användaren länkas till sin information i databasen. Länken skapas helt enkelt när användarnamnet anges. Då erhålls UserId för användaren och operationen utförs men för att underlätta processen för användaren och inte ange användarnamnet vid varje process sparar applikationen UserId under hela sessionen för att använda det i olika operationer. Det betraktas som en session kod i webläsare men för säkerhets skull används och sparas koden i applikationen själv genom denna klass.

```
public static class UserService
{
    public static string UserId { get; set; }
}
```

Figur 32. Kodsnutten visar hur UserServices initieras.

4.4.4. Controller

Kontrollern är en väsentlig grund för applikationen eftersom den tar emot de olika förfrågningarna från front-end och bearbetar dem med hjälp av de olika modellerna som nämnts ovan och skickar resultatet igen, bestående av två vertikala klasser:

4.4.4.1. AuthenticateController

Den här klassen har ansvar för att logga in och registrera ett nytt konto med en användares eller en administratörs befogenheter, och består huvudsakligen av tre metoder:

Login

Denna metod tar inloggningsinformationen från front-end som en json-fil och söker sedan efter användarnamnet i databasen. Här dyker det upp två olika fall, det första är när användarnamnet hittas så kommer metoden att matcha lösenordet och om lösenordet är korrekt börjar den andra processen. Där söker den efter användarens behörigheter, antingen som administratör eller en vanlig användare. Metoden skapar sedan token och ger den en giltighetstid på tre timmar för att tillåta användaren att stanna kvar i applikationen utan att behöva auktorisera sig igen. Sen returnerar metoden token med giltigheten till front-end. Det andra fallet är när användaren inte existerar i databasen, då kommer en felkod skickas till front-end (Unauthorized).

```
[HttpPost]
[Route("login")]
public async Task<IActionResult> Login([FromBody] LoginModel model)...
```

Figur 33. Kodsnutten visar hur Login Metod ser ut och vilka parametrar som den har.

Register

Denna metod tar registrerings informationen från front-end som en json-fil och söker sedan efter användarnamnet i databasen. Här dyker det upp två olika fall. Det första är när användarnamnet hittas så kommer metoden att skicka en felkod till front-end som säger “User already exists!”. Det andra fallet är när användaren inte existerar i databasen. Då kommer metoden att skapa en ny användare med informationen bifogad av front-end och chiffrerar lösenordet. Metoden ger sedan användaren de behörigheter som krävs och lägger till användaren i databasen. Metoden returnerar en bekräftelsekod till fronten när användarens konto är skapat rätt (“User created successfully!”).

```
[HttpPost]
[Route("register")]
public async Task<IActionResult> Register([FromForm] RegisterModel model)...
```

Figur 34. Kodsnutten visar hur Register metoden ser ut och vilka parametrar som den har.

RegisterAdmin

I projektet finns det två huvudroller, användare och administratör. Den föregående klassen används för att registrera ett nytt konto med användarbefogenheter. Den här klassen fungerar precis som den föregående klassen men den ger administratörsbefogenheter istället, där

administratören kan ta bort alla befintliga resor för alla användare, avregistrera användarkonton osv.

```
[HttpPost]
[Route("register-admin")]
public async Task<IActionResult> RegisterAdmin([FromBody] RegisterModel model)...
```

Figur 35. Kodsnutten visar hur RegisterAdmin metoden ser ut och vilka parametrar som den har.

4.4.4.2. TripsController

Den här klassen har ansvaret att utföra olika funktioner på tabellen för resor i databasen liksom hämta tillgängliga resor, boka en resa, avboka en resa osv. Den består av nio metoder och beskrivs som följande:

GetTrips

Den metoden är ganska enkel eftersom den hämtar alla resor i tabellen utan att filtrera resultaten.

```
[HttpGet]
public IEnumerable<Trip> GetTrips()
{
    return _context.Trips;
}
```

Figur 36. Kodsnutten visar hur GetTrips metoden ser ut och vilka parametrar som den har.

GetTrip

Denna metod hämtar en specifik resa enligt resans identifierare (ID). Identifieraren hämtas från länken för den metoden, till exempel resa nummer 5 i tabellen hämtas genom att gå in i länken “TripsController/api/Trips/5”.

```
// GET: api/Trips/5
[HttpGet("{id}")]
public async Task<IActionResult> GetTrip([FromRoute] int id)...
```

Figur 37. Kodsnutten visar hur GetTrip metoden ser ut och vilka parametrar som den har.

PutTrip

Den metoden modifierar en resa som redan finns i tabellen enligt resans identifierare(ID). Identifieraren hämtas från länken för den metoden förutom den modifieringsinformation som tas som Jason-information från själva begäran.

```
// PUT: api/Trips/5
[HttpPut("{id}")]
public async Task<IActionResult> PutTrip([FromRoute] int id, [FromBody] Trip trip)...
```

Figur 38. Kodsnutten visar hur PutTrip metoden ser ut och vilka parametrar som den har.

PostTrip

Denna metod tillåter användare att ladda upp en resa på webbplatsen för att dela den med de andra användarna. Metoden kräver att användaren är inloggad i systemet, så det begär den tydligt innan den utförs.

```
[Authorize]
[HttpPost]
public async Task<IActionResult> PostTrip([FromBody] TripViewModel trip)
```

Figur 39. Kodsnutten visar hur PostTrip metoden ser ut och vilka parametrar som den har.

Därefter tar den användarens identifierare för den inloggade användaren med hjälp av "UserService" klassen för att lägga till resan till rätt användare. Resans information bifogas i begäran som en Json-fil, så skapar applikationen resan och fyller i all information enligt den bifogade informationen. Metoden lägger sedan till resan och ansluter den till användaren genom att skapa de obligatoriska relationerna mellan tabellerna. Metoden tar hänsyn till att märka användaren som resans förare innan anslutningen. Metoden sparar i slutändan ändringarna i databasen.

```
Trip newtrip = new Trip()
{
    Description = trip.Description,
    Destination = trip.Destination,
    Price = trip.Price,
    SeatAvailable = trip.SeatAvailable,
    StartLocation = trip.StartLocation,
    StartTime = trip.StartTime

};
UserTrip usertrip = new UserTrip()
{
    User = user, //inloggade user
    Trip = newtrip,
    TripDate = DateTime.Now,
    TypeTrip = TypeTripUser.Driver
};
```

Figur 40. Kodsnutten visar hur Trip och UserTrip objekt skapas.

Search

Den metoden utför huvudsökprocessen i applikationen. Den tar sökparametrarna från själva begäran som Json-fil, hämtar resultaten, filtrerar dem enligt parametrarna och returnerar resultaten som en lista på tillgängliga resor. Ifall ingen matchande resa hittas, skickas ett förtydligande meddelande till front-end för att visa det till användaren.

```
[Authorize]
[HttpPost]
[Route("search")]
public async Task<IActionResult> Search([FromBody] TripViewModel searchtrip)
```

Figur 41. Kodsnutten visar hur Search metoden ser ut och vilka parameter som den har.

Denna metod tar inte hänsyn till resedatumet, det visar alla tillgängliga resor även om de inte matchar datumet, anledningen till detta är att när alla resor visas kan användaren välja resan dagen efter eller före, därmed undviker användaren att göra många sökningar för att hitta resan. Metoden matchar bara antalet tillgängliga platser, destinationen och avgångsplats.

```
var trips = _context.Trips.Where(x => ((x.SeatAvailable >= searchtrip.SeatAvailable) && (x.StartLocation == searchtrip.StartLocation
      || x.Destination == searchtrip.Destination))).ToList();
```

Figur 42. Kodsnutten visar hur Sql kod utförs för att matcha sökningen.

MyTrips

Den metoden kallas från två olika ställen i front-end, när användaren vill ha sina bokade resor som passagerare eller som förare. Detta bestäms av metodlänken, så om länken “/TripsController/api/trips/Driver” anropas, så hämtas förarens resor, men om länken “/TripsController/api/trips/passenger” anropas är det passagerarens resor som hämtas.

```
[HttpGet]
[Authorize]
[Route("mytrips/{id}")]
public async Task<IActionResult> Mytrips([FromRoute] string id)
```

Figur 43. Kodsnutten visar hur Mytrips metoden ser ut och vilka parametrar som den har.

Metoden tar informationen från själva länken, identifierar typen och sammanfogar sedan två tabeller (Trips, UserTrips) för att extrahera resultatet. Sammanfogningsmetoden är att matcha användar-ID med “typetrip” och konvertera resultaten till en lista.

```
var trips = (from p in _context.Trips
             join e in _context.UserTrips
               p.TripId equals e.TripId
             where (e.Id==UserService.UserId && e.TypeTrip.Equals(id))
```

Figur 44. Kodsnutten visar hur Sql join kod fungerar.

Book

Den metoden reserverar en specifik resa för användaren enligt resan-ID, resan-ID bifogas inuti metod länken.

```
[Authorize]
[HttpGet]
[Route("book/{id}")]
public async Task<IActionResult> Book([FromRoute] int id)
```

Figur 45. Kodsnutten visar hur Book metoden ser ut och vilka parameter som den har.

Den önskade resan söks därefter inom tabellen för resor och antalet tillgängliga platser ändras och minskas med en. En ny bokning görs och läggs till i bokningstabellen med hänsyn till att märka användaren som passagerare.

Med hjälp “emailSenders” klassen skickas ett e-postmeddelande till användaren med information om den bokade resan. I slutändan registreras alla ändringar i databasen.

```
_emailSendercs.SendEmail(messege);
```

Figur 46. Kodsnutten visar hur SendEmail metoden fungerar.

UserData

Den metoden hämtar den personliga informationen för den inloggade användaren från användarens tabell och skickar den till front-end.

```
[Authorize]
[HttpGet]
[Route("userdata")]
public async Task<IActionResult> Userdata()
```

Figur 47. Kodsnutten visar hur Userdata metoden ser ut och vilka parametrar som den har.

DeleteTrip

Den metoden används för att avboka en specifik resa enligt resans-ID, resans-ID är bifogad inuti metod länken.

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTrip([FromRoute] int id)
```

Figur 48. Kodsnutten visar hur DeleteTrip Metoden ser ut och vilka parametrar som den har.

Det finns två typer av avbokningar. Det första är när användaren är en passagerare, då är det relativt enkelt, eftersom metoden tar bort bokningen inifrån bokningstabellen, ökar antalet tillgängliga säten med en i resorstabel och skickar ett bekräftelse till användaren för att bekräfta avbokningen.

```
trip.UserTrips.Remove(usertrip[0]);
trip.SeatAvailable++;
var user = await _userManager.FindByIdAsync(UserService.UserId);
var messege = new Messege(new string[] { user.Email }, "Cancel Confirmation", FormatMessage.CancelTrip(trip));
```

Figur 49. Kodsnutten visar hur avboknings processen utförs.

Det andra fallet är när användaren är förare, där måste alla passagerare inhämtas från boknings tabellen, avboka sina bokningar och skicka en bekräftelse till alla passagerare om att bokningen har avbokats, förutom att ta bort hela resan från tabellen för resor.

```
for(int i =0;i<usertrips.Count;i++)
{
    emails[i] = (await _userManager.FindByIdAsync(usertrips[i].Id)).Email;
}
var messege = new Messege(emails, "Cancel Confirmation", FormatMessage.CancelTrip(trip));
_emailSendercs.SendEmail(messege);
_context.Trips.Remove(trip);
```

Figur 50. Kodsnutten visar hur avboknings görs för alla passagerare.

GetTrip

Den metoden är relativt enkel, den hämtar bara en specifik resa enligt bifogat resans-ID inuti länken.

```
// GET: api/Trips/5
[HttpGet("{id}")]
public async Task<IActionResult> GetTrip([FromRoute] int id)
```

Figur 51. Kodsnutten visar hur GetTrip Metoden ser ut och vilka parametrar som den har.

5. Resultat

Projektet har resulterat i en applikation som kan förenkla delningen av olika resor mellan användare och hjälpa miljön genom att minska bilutsläpp. Genom denna applikation kan användaren båda välja att addera nya resor för att dela dem med andra och boka en befintlig resa för att spara lite pengar.

Med “Addera “ alternativet kan användaren lägga till sin resa från vilken stad som helst i Sverige till vilken destination som helst och lägga till antalet tillgängliga säten med varsitt bestämt pris. Därefter läggs resan till automatiskt i applikationens databas. Användaren kan komma åt sin personliga sida och se alla skapade resor som då hämtas från databasen.

Med “Boka/Avboka” alternativet kan användaren boka eller avboka sin resa. Om användaren är en passagerare annulleras bokningen/resan omedelbart och ett bekräftelsemeddelande skickas till de två parterna för att informera dem om förändringen. Om användaren är föraren kommer hela resan att annulleras och e-post skickas till alla passagerare för att informera dem.

Applikationens första version innehåller följande sidor och en navbar:

Splash Page: En informationssida, där man ser applikationens namn och logga med ett välkomstmeddelande som uppvisas i form av en notifikation. Denna sida är den första sidan som användaren ser vid uppstarten.

Navbar: Navigation Bar som delegerar användaren till rätt sida. Navigation Bar förblir fast under hela applikationens livstid för att underlätta navigering mellan applikationens olika sidor.

Sign Up Page: På denna sida får användaren mata in sin personliga information utöver fotot för att registrera sig på webbplatsen och använda dess olika funktioner. Utan registrering på webbplatsen är det inte möjligt att se andras resor eller lägga till någon resa.

Sign In Page: Sidan tillåter användaren logga in på applikationen för att kunna använda de olika funktionerna. Varje användare kännetecknas av ett användarnamn och lösenord.

Add Trip page: Den tillåter användaren att lägga till en resa och dela den med andra. Användaren matar in reseinformation som avgångsplats, destination, antal tomma säten, och priset. Genom att klicka på en knapp bekräftar man informationen och resan sparas i databasen för andra att söka efter.

Search page: Tillåter användaren att söka efter tillgängliga resor på webbplatsen där användaren matar in sökinformationen, liksom avgångsstation, destination, och datum. Befintliga resor visas på skärmen enligt de önskade sökfiltren. Man har då möjlighet att antingen boka eller ta en blick över resvägen genom en popup karta.

My Account page: Detta är sidan som ger användaren möjlighet att se sitt konto på webbplatsen, och få en snabb blick över sin bokade eller skapade resor.

Bokade/Skapade resor: Bokade och skapade resor kommer man åt från sin kontosida där det finns en knapp för respektive alternativ. Där kan man se resan och dess information och även avboka resor.

About Us Page: Denna sidan består endast av en rubrik och textruta. Ännålänge är det bara en mall men det enda som krävs är att det skrivs text för About us.

Alla sidorna kan hittas som bilder i Bilaga 2.

6. Diskussion

Design

Vid början av utvecklingen så användes .css filer för att designa komponenterna. Det var först senare under utvecklingen som styled components hittades och började användas. Så vissa komponenter använder styling från .css filer och vissa från styled components. Det har ingen direkt påverkan på själva applikationen men för koden och strukturen hade det varit bättre om allt använde styled components.

Val av ramverk

I början av projektet var det oklart vilket ramverk som skulle användas för utveckling av back-end. Valet låg mellan Spring Boot i Java och ASP.NET i C#. Eftersom vi både hade tidigare erfarenhet av Java lutade det mot Spring Boot men efter ytterligare undersökning upptäcktes det att ASP.NET passade bättre för projektet. Detta kan man se i faktumet att LINQ stöds i ASP.NET, dock inte i Java.

LINQ (Language Integrated Query) är enhetlig frågesyntax (query syntax) i C# och VB.NET för att hämta data från olika källor och format. Den är integrerad både i C# och VB, vilket eliminerar ojämnheten mellan programmeringsspråk och databaser, samt tillhandahåller ett enda fråga gränssnitt för olika typer av datakällor och låter programmerare komma åt data på ett sätt som liknar SQL.

Den mest synliga delen av LINQ för en utvecklare som skriver queries är query expression som kan enkelt skrivas med hjälp av LINQ(query syntax). Genom att använda LINQ kan det underlätta filtrerings-, hämtning- och grupperings operationer på datakällor med ett minimum av kod. LINQ är grundläggande (query expression) för att hämta och omvandla data från SQL-databaser. Till exempel följande kod kan skrivas omedelbart i C#:

```
var temp = from p in db.resor
           where p.startLocation == “Göteborg”
           select p;
```

Nyckelordet “var” lades till samtidigt som LINQ, eftersom den typen som fås tillbaka är förmodligen komplex. Kompilatorn upptäcker automatiskt typen och skapar internt resultatvariabler med lämplig typ.

Därför valdes istället ASP.NET för utvecklingen av back-end Eftersom ASAP i hög grad underlättar processen för att hämta och bearbeta information från databasen jämfört med Java [13].

Resor med delmål
Anledningen till att Resor med delmål inte implementerats är att när man skapar ett utvecklarkonto på Google får utvecklare en gratis initial summa på två tusen sek, anledningen är att Google Map-API förfrågningar kostar lite pengar för behandling. Så när kartan begärs från Google är det inte gratis och varje rendering av kartan innebär en begäran som skickas till Google API med en liten kostnad.

Huvudproblemet är att det finns ingen API som kan skicka information om städerna som ligger mellan två utsedda punkter. Antag till exempel att resan är från punkt A till punkt B och passerar genom två punkter D och H. Det finns ingen API som kan skicka information om punkterna D och H. Det enda sättet som finns för att veta om resan passerar genom punkterna D och H är att skicka många kontinuerliga förfrågningar till Google Map-API Decoder för att informera om de olika platser som finns på vägen. Google Map-API tillhandahåller en tjänst genom att skicka koordinaterna till ett område och fråga om det är en stad, en restaurang, en gata etc.

Till exempel, resan från A till B passerar hundra punkter som kan vara en stad, en restaurang eller ett kafé. För att veta om punkten är en stad eller inte måste hundra förfrågningar skickas till Google Map-API och bearbeta responsen för att ta reda på om det är en stad eller inte. Detta sätt kostar inte bara mycket pengar men det förhindras också av Google. Google tillåter inte utvecklare att skicka många behandlingsförfrågningar på en gång, så om hundra förfrågningar skickas per sekund kommer endast mellan 10 och 20 förfrågningar att behandlas. Därför implementerades inte denna tjänst i projektet, i väntan på att en annan lösning som skulle vara mer lämplig.

6.1. Miljö

En av grundtankarna med projektet är att minska miljöpåverkan genom att minska bilar i trafiken. Om applikationen skulle lanseras så resulterar det förhoppningsvis i att fler reser tillsammans och därmed minskar utsläppen från bilar. Det som kan vara värt att tänka på ifall applikationen blir verklighet är hur applikationen och databasen “hostas” eftersom sådant kan kräva mycket energi. Så en bra fråga att ta sig an är hur man kan driva själva systemet på ett hållbart sätt.

6.2. Etik

Det finns vissa etiska problem med att samåka med personer man inte känner. Därför skulle det krävas att alla konton är kopplade till en person via någon form av identifikation och att alla resor med information lagras i en databas för eventuella problem. Detta lyfter också ett nytt

problem med att lagra användares information och resvanor. Vid full utveckling av detta systemet skulle det därför krävas att all data som lagras är krypterad eller lagrat säkert så att ingen utomstående kan ta del av den. En annan sak att tänka på är att alla har olika “regler” för sina bilar, till exempel om de tillåter mat eller husdjur i sina bilar. Allergier är också viktigt att tänka på med mat och djur.

7. Slutsats och Vidareutveckling

Syftet med den här studien var att konstruera en applikation med hjälp av ramverket React bibliotek (Javascript) och visual studio community (ASP.NET entityframework).

En viktig lärdom som går att dra från resultaten är att det är möjligt att koppla olika plattformar trots skillnaden i programmeringsspråk eller utvecklingsmiljö och att det finns flera sätt att utveckla webbprojektet. Det är inte nödvändigt att programmera back-end med hjälp av ASP.NET, det kan göras med många andra plattformar liksom php och spring boot.

En annan viktig lärdom är att under utvecklingsstadiet rekommenderas att utvecklingen utförs på en lokal server och inte på Internet. Det kommer att leda till en snabbare utvecklingsprocess, minskad kostnad och man undviker serveranslutnings problem.

Data- och informationssäkerhet är en viktig del av utvecklingsprocessen och bör inte tolereras. Cybersäkerhet bör utvecklas och integreras separat under projektets gång. Under det projektet hanterades vissa säkerhetsproblem, till exempel lösenordskryptering med “one way hash function”, men användarkonton säkerhet måste förstärkas ytterligare med mer sofistikerade metoder som “two factor authentication“ .

Idag finns det ett stort antal sociala nätverk, såsom Facebook, Twitter, LinkedIn, Instagram och Pinterest och liknande, och de bör användas positivt med applikationen. En ide är att kunna koppla applikationen till alla sociala nätverk och underlätta registrering och inloggningsprocessen för användaren, men också säkerhet bör övervägas och hur enkla lösenord hanteras i detta fall.

7.1. Vidareutveckling

Det finns ett par delar, som till exempel startsidan, som är fungerande på olika skärmstorlekar men inte helt mobilanpassade. Med fortsatt utveckling hade det varit bra att se till så hela applikationen är anpassad för både dator och mobil.

I sitt nuvarande skick fungerar inte applikationens kontoinställningar eftersom prioriteringen låg på appens huvudfunktioner. Tanken var att man skulle kunna se sina personliga uppgifter och ändra till exempel email eller lösenord.

En stor funktion som var planerad att implementera är att kunna lägga till delmål på skapade resor eller boka delar av resan om man bara ville åka med en bit. På grund av tidsbrist och

svårigheter med att implementera den funktionen tillsammans med en API så ansågs det inte möjligt att utföra. För vidareutveckling är detta en av de viktigaste punkterna att implementera eftersom det skulle öka applikationen och tjänstens värde och användbarhet signifikant. Som det är nu skulle man tekniskt sett kunna boka en resa och fråga föraren om det är möjligt att bara åka en del av resan men för klarhet bland annat så skulle det vara bättre om funktionen finns i applikationen. Man skulle till exempel kunna lägga till alternativ där föraren kan kryssa i en ruta om den vill att man ska kunna boka delar av resan eller ej.

Det nuvarande inloggningssystemet är fungerande och har vissa säkerhetsfunktioner såsom password hash men det finns fortfarande utrymme för förbättring. Ett mer avancerat inloggningssystem och förbättrad säkerhet är ett krav för att kunna lansera produkten.

En sista punkt som inte är absolut nödvändig men kan vara bra att ha i åtanke är att utföra användartester och få feedback från utomstående personer. Dels för att för att få ett annat perspektiv och åsikter om design och funktioner men också för att det är lätt som utvecklare att missa fel eller småsaker som kan optimeras eftersom man är väldigt bekant med systemet och har en mycket bättre koll än en ny användare.

Referenser

[1] ”Chalmers,” Ensamma förare använder mest energi, Jun. 2020. [Online]. Tillgänglig: https://www.chalmers.se/sv/institutioner/see/nyheter/Sidor/Ensamma-forare-anvander-mest-energi.aspx. [Hämtad: Maj. 22, 2021].

[2] ”Trafikverket,”Vägtrafikens utsläpp 2020, Feb. 2020. [Online]. Tillgänglig: https://www.trafikverket.se/contentassets/ac3df8a1b07a49f09230c64189ccba4b/pm-vagtrafikens-utslapp-210224.pdf [Hämtad: Maj. 22, 2021].

[3] Biao Yin, Liu Liu, Nicolas Coulombel, Vincent Viguie. Appraising the environmental benefits of ridesharing: The Paris region case study. Journal of Cleaner Production, Elsevier, 2018, 177, pp.888-898. ff10.1016/j.jclepro.2017.12.186ff. ffhal-01695082 [Online]. Tillgänglig: https://hal.archives-ouvertes.fr/hal-01695082/document [Hämtad: Juni. 14, 2021].

[4]. ” Visual Studio” May. 2020. [Online]. Tillgänglig: https://visualstudio.microsoft.com/ [Hämtad: Maj. 22, 2021].

[5]. ” What is ASP.NET” May. 2020. [Online]. Tillgänglig: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet [Hämtad: April. 22, 2021].

[6]. ”What is SQL Server Management Studio” Sep. 2019. [Online]. Tillgänglig: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15 [Hämtad: April. 22, 2021].

[7].”React” Nov. 2020. [Online]. Tillgänglig: https://reactjs.org/ [Hämtad: Maj. 26, 2021].

[8] ”Why did we build Visual Studio Code?” Maj. 2021. [Online]. Tillgänglig: https://code.visualstudio.com/docs/editor/whyvscode [Hämtad: Maj. 20, 2021].

[9] ”What is GitHub?” Jul. 2020. [Online]. Tillgänglig: https://guides.github.com/activities/hello-world/ [Hämtad: April. 26, 2021].

[10] ”Trello” Maj. 2021. [Online]. Tillgänglig: https://trello.com/en [Hämtad: Maj. 20, 2021].

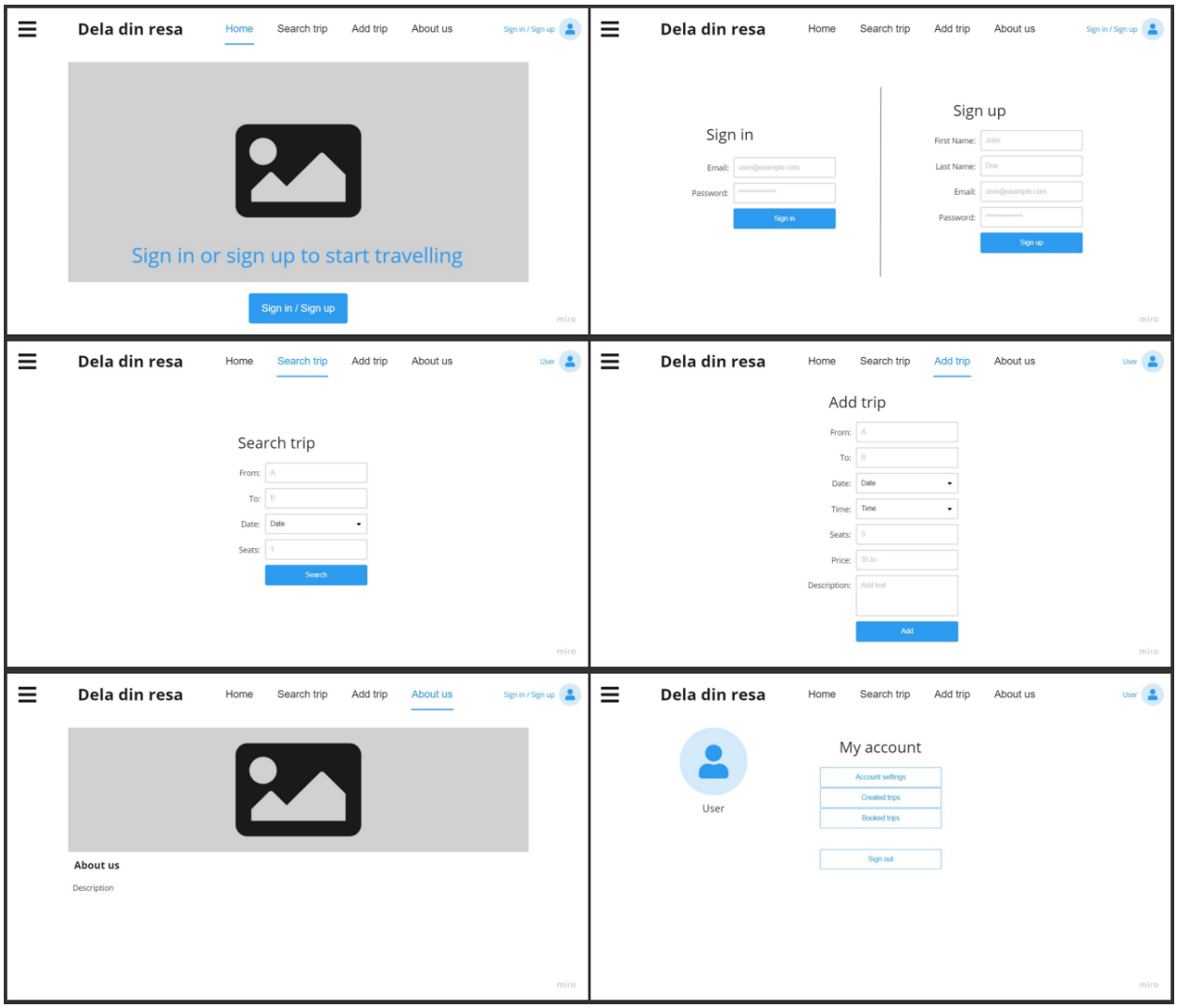
[11] ”WHAT IS DISCORD? ” Feb. 2021. [Online]. Tillgänglig: https://discord.com/safety/360044149331-What-is-Discord [Hämtad: Maj. 21, 2021].

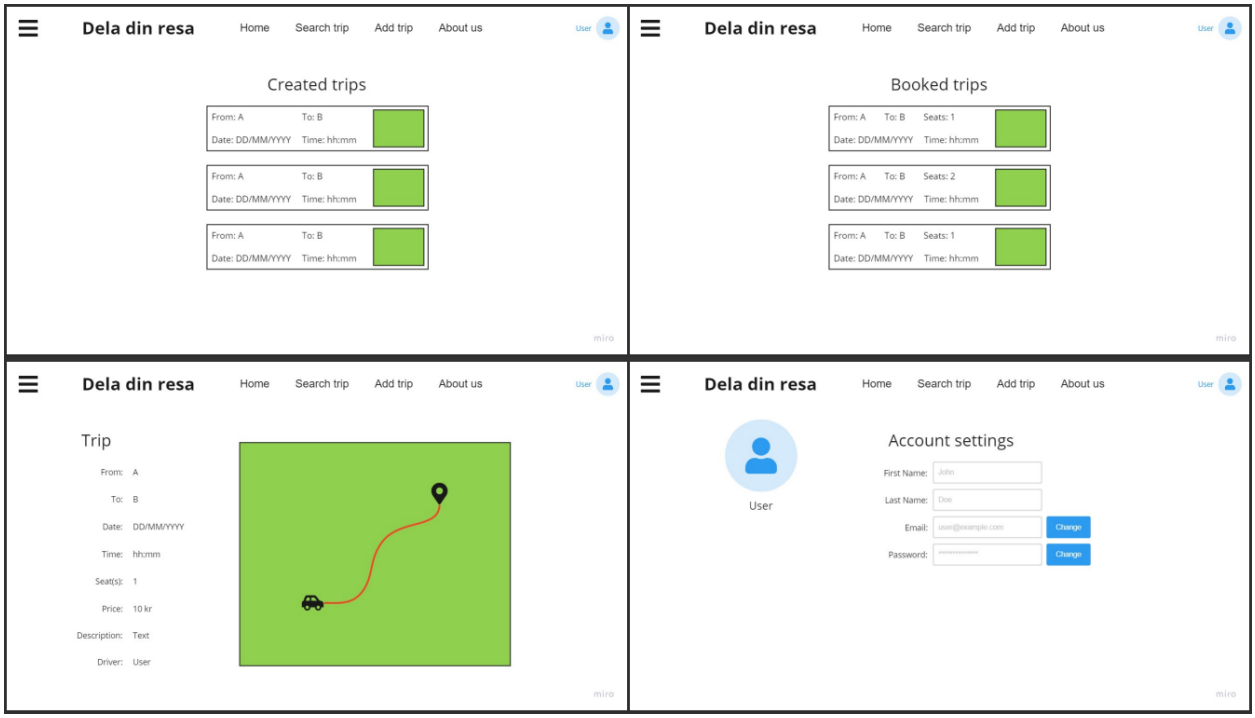
[12] “Hash passwords in ASP.NET Core” October.2016.[Online] Tillgänglig: https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-5.0 [Hämtad: Juni. 10, 2021].

[13] “Language Integrated Query (LINQ) (C#)” Feb.2017 [Online] Tillgänglig https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ [Hämtad: Juni. 11, 2021].

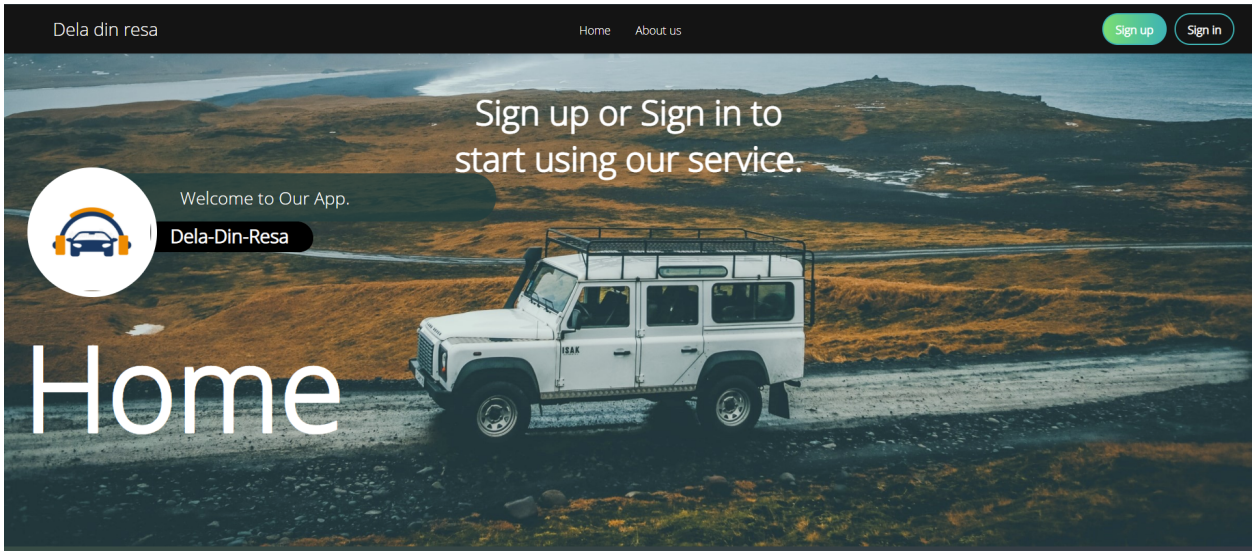
Bilagor

Bilaga 1 - Wireframe Design

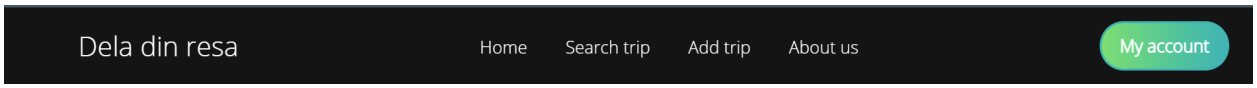



Dela din resa
Home
Search trip
Add trip
About us
User
Created trips
From: A
To: B
Date: DD/MM/YYYY
Time: hh:mm
Booked trips
Seats: 1
Seats: 2
Trip
From: A
To: B
Date: DD/MM/YYYY
Time: hh:mm
Seat(s): 1
Price: 10 kr
Description: Text
Driver: User
Account settings
First Name: John
Last Name: Doe
Email: user@example.com
Password:
Change
miro

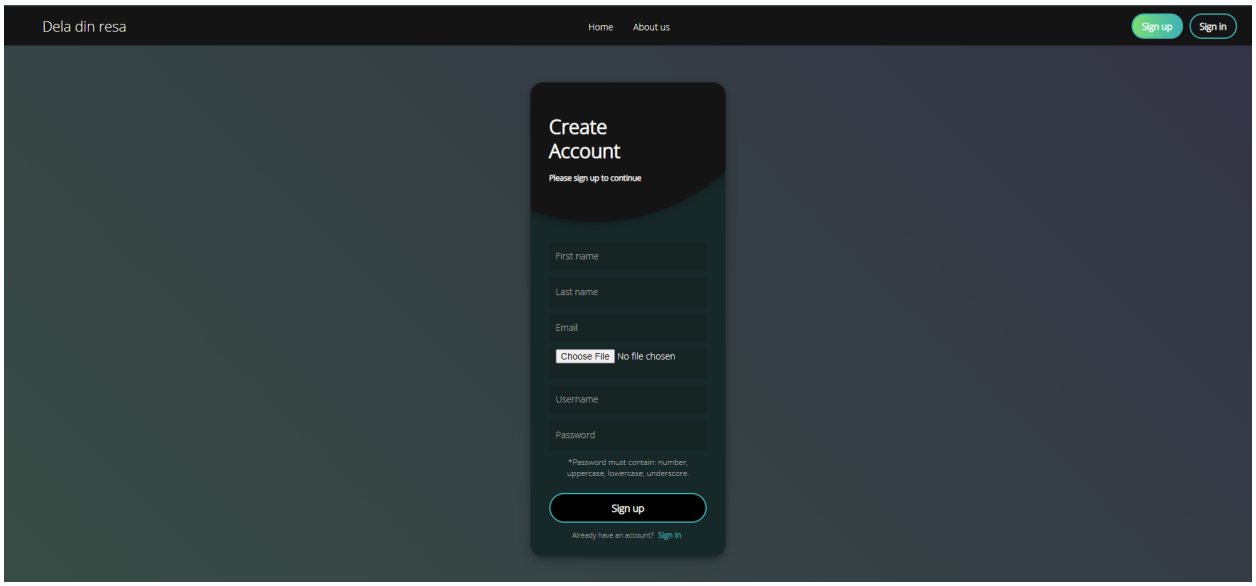
Bilaga 2 - Färdigutvecklad Webbapplikation



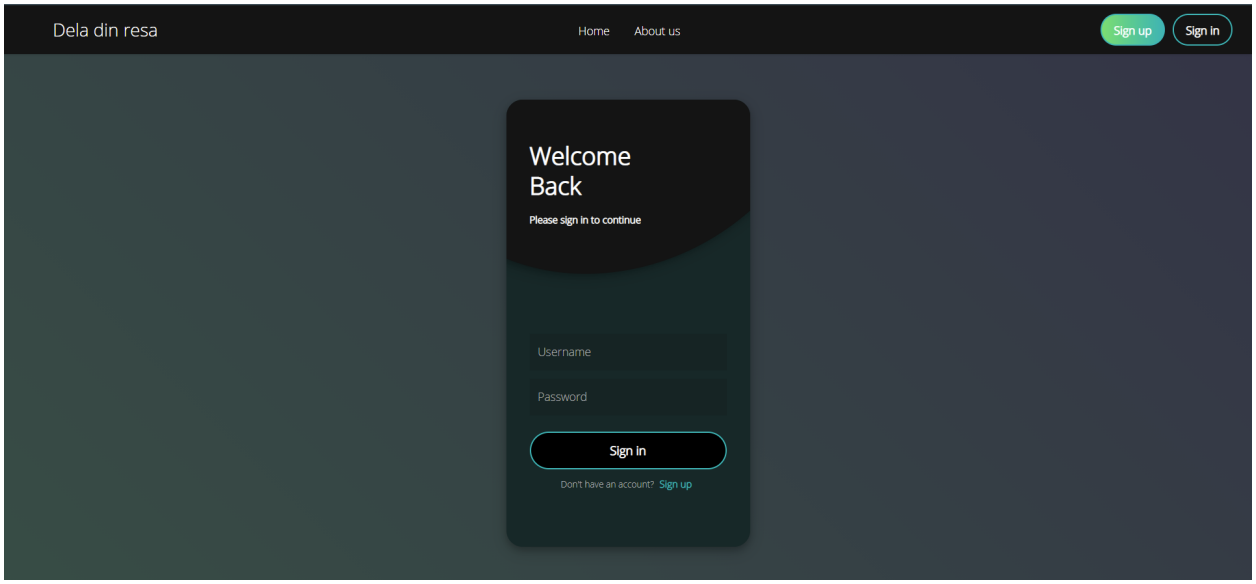
Landing Sida



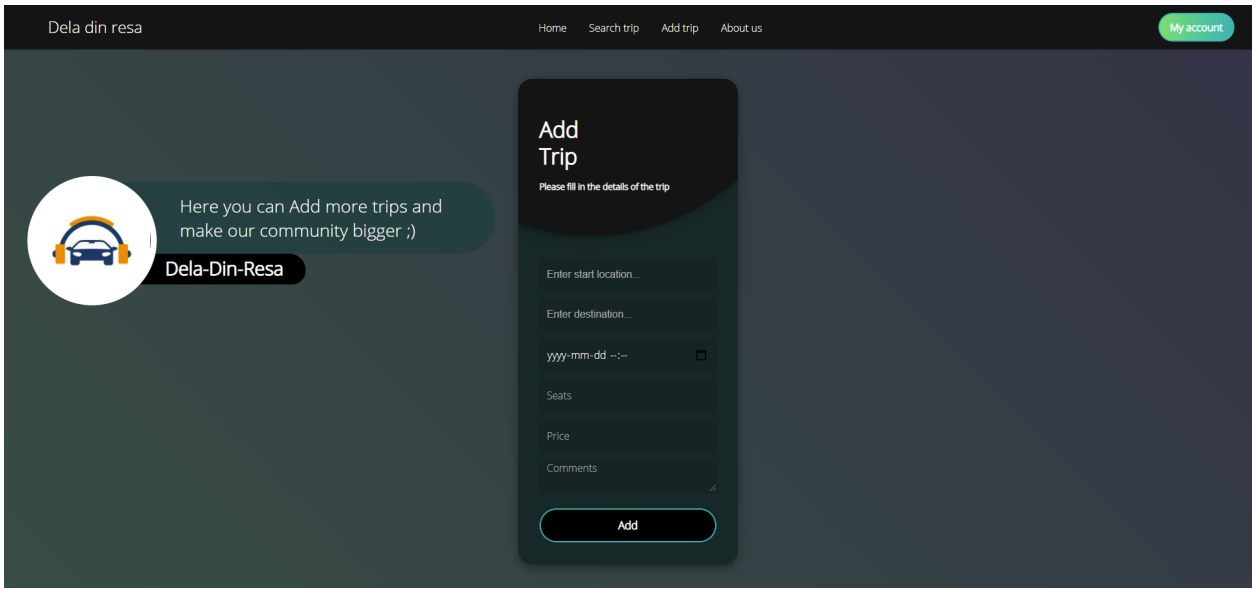
Navbar



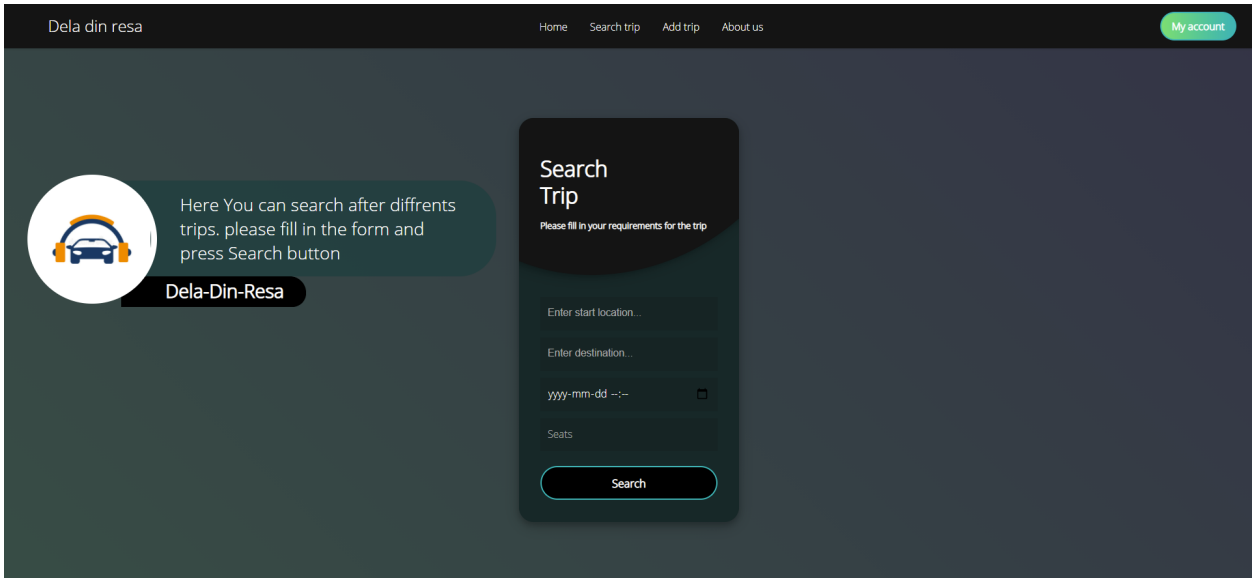
Sign up Sida



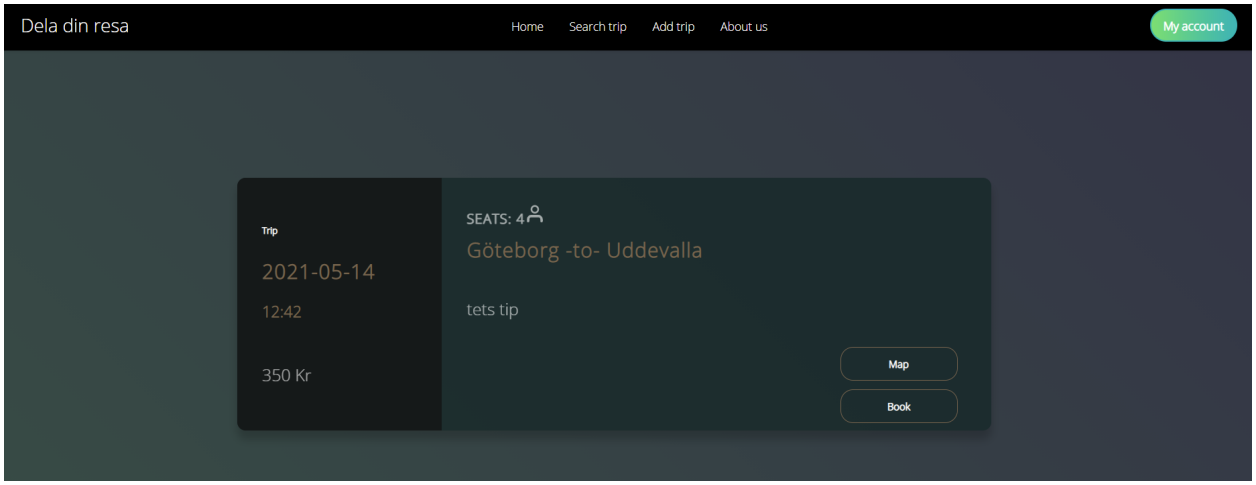
Sign in Sida



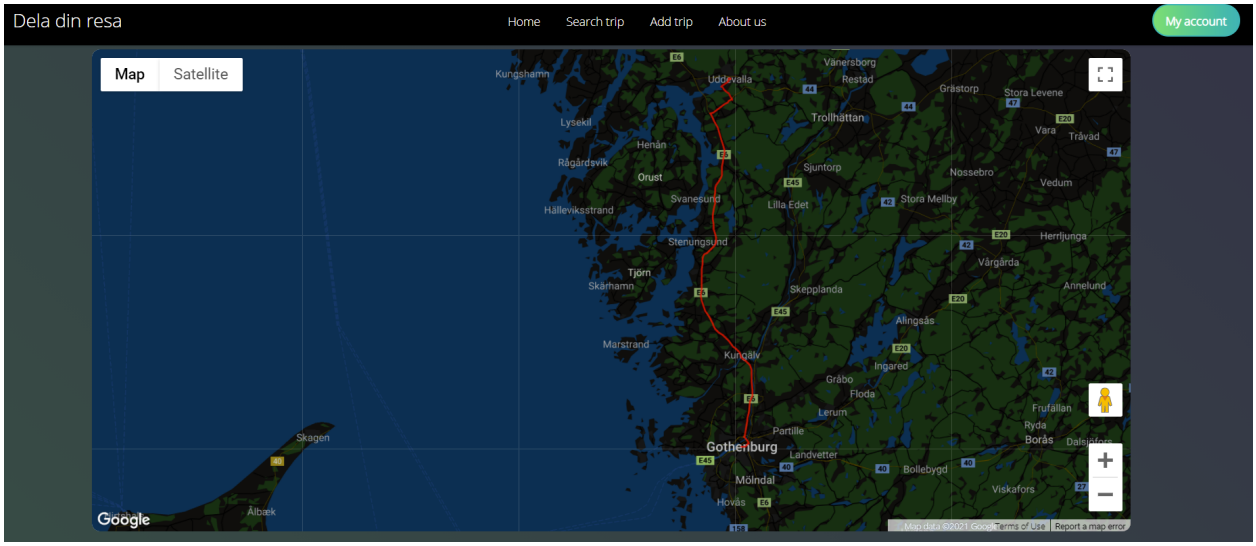
AddTrip Sida



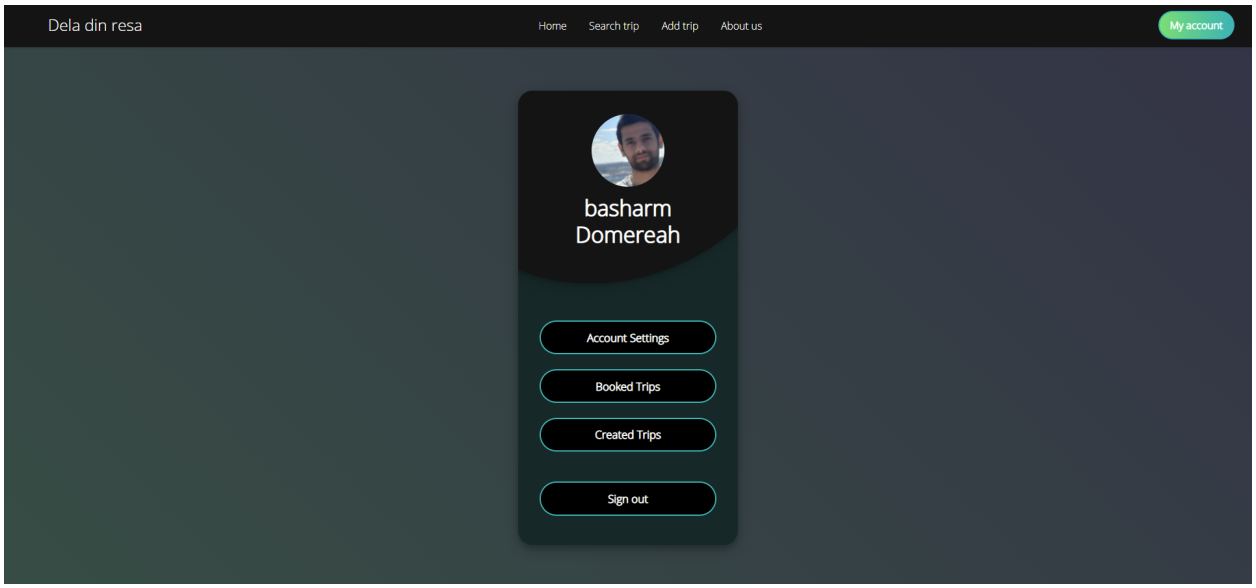
SearchTrip Sida



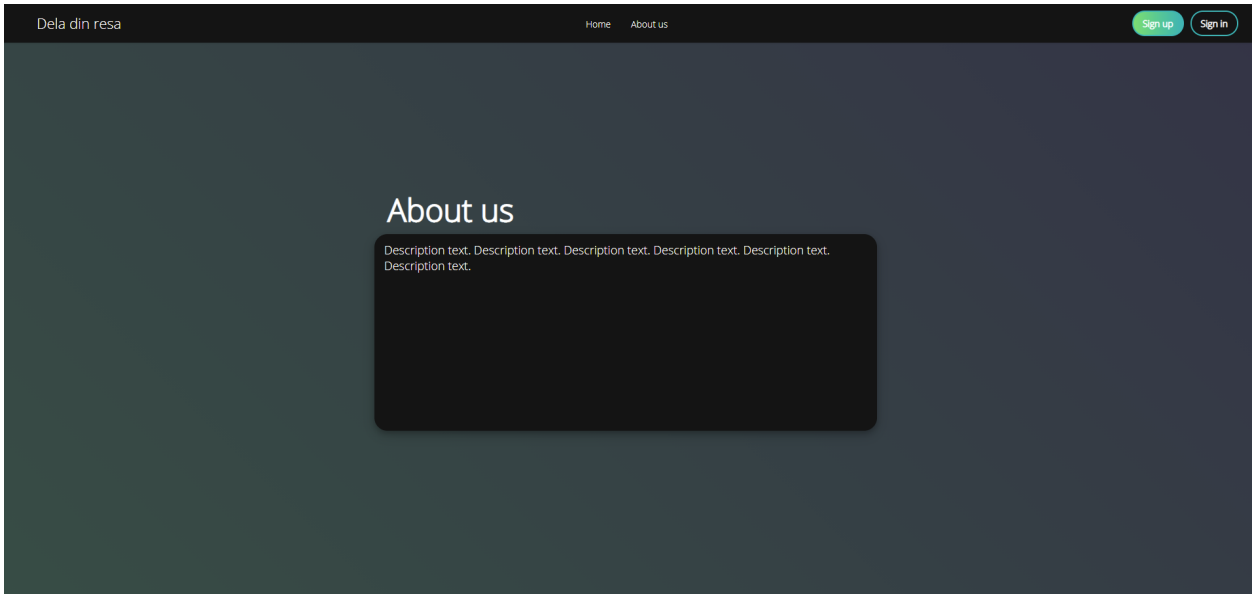
TripKort



Karta



Min Sida


Dela din resa
Home
About us
Sign up
Sign in
About us
Description text. Description text. Description text. Description text. Description text. Description text.

About us