Природо-математическа гимназия

„Васил Друмев“

гр. Велико Търново

ДИПЛОМЕН ПРОЕКТ

за придобиване на трета степен на професионална квалификация

Професия: „Приложен програмист“

Специалност: „Приложно програмиране“

Тема: Изграждане на комплекс от методи, средства и подходи при разработване на приложение „Споделено пътуване“

Ученик: Ръководител консултант:

Боян Росенов Божанов Милен Василев

Велико Търново 2025

Съдържание

Увод

С нарастващата нужда от устойчиви и икономически ефективни начини за пътуване, съвременните платформи за споделено пътуване предлагат удобни решения за свързване на шофьори и пътници с общи маршрути. В този контекст представям документацията за проекта „СПОДЕЛЕНО ПЪТУВАНЕ“ – информационна система, предназначена да улесни организацията и участието в съвместни пътувания с автомобил.

Целта на дипломната работа е да се разработи и документира цялостна уеб базирана система, която да обслужва както обикновени потребители (пътници), така и шофьори, желаещи да споделят разходите по своите пътувания. Чрез платформата потребителите могат да разглеждат налични пътувания, да резервират място като спътници или да създадат собствено пътуване като шофьори.

В основата на системата е двукомпонентна архитектура, състояща се от база данни и уеб приложение със слой за бизнес логика, който осигурява комуникация между потребителския интерфейс и данните. Проектът предлага разнообразни функционалности, включително управление на потребителски профили, създаване и преглед на пътувания, кандидатстване за пътуване и администраторски контрол.

В настоящата документация ще бъде представена подробна информация за проектирането, функционалностите и реализацията на „СПОДЕЛЕНО ПЪТУВАНЕ“, както и взаимодействието между различните компоненти на системата. Ще бъде разгледано използването на валидация на данни, падащи менюта, потребителска регистрация и автентикация, съобразено с нуждите и изискванията на проекта.

Тази документация има за цел да предостави ясен преглед на проекта „СПОДЕЛЕНО ПЪТУВАНЕ“ и да подпомогне разбирането на неговата структура, предназначение и възможности. В следващите раздели ще бъдат описани в детайли отделните модули на системата, използваните технологии и постигнатите резултати.

За реализацията на този проект са използвани HTML, CSS, Bootstrap, JavaScript, C#, ASP.NET Core и Entity Framework Core.

Теоритична част - Технологии за реализиране на уеб-базирани информационни системи

1.1 Особености на уеб-базираните информационни системи

Уеб информационна система, наричана също и уеб-базирана информационна система, е софтуерна система, която използва уеб технологии за доставяне на информация и услуги до потребители или други информационни системи и приложения. Основната цел на тази система е да публикува и поддържа данни, като използва хипертекстови принципи.

Типичната уеб информационна система се състои от едно или повече уеб приложения, които имат специфични функции, ориентирани към компоненти, заедно с информационни елементи и други не-уеб компоненти. Обикновено front-end на системата е уеб браузър, докато back-end се състои от база данни.

С развитието на уеб технологиите и нарастващия интерес на потребителите и разработчиците, понятието за уеб сайт се развива от прост набор от HTML страници към уеб-базирани информационни системи (WIS). WIS предоставят средства за достъп до комплексни данни и интерактивни услуги чрез Мрежата. Примери за WIS включват е-бизнес приложения, CRM и приложения за веригата на доставки.

Въпреки че традиционните и уеб-базирани информационни системи имат много прилики, съществуват и значителни различия. Уеб-базираните информационни системи представляват различни предизвикателства за разработчика, който трябва да се справя с неизвестни условия на мрежата и потребителите, различни типове данни и други въпроси, като управление на съдържанието, представяне и използваемост. Въпреки че уеб-базираните информационни системи са платформено независими по отношение на информацията за доставка, което е една от причините, поради които те са толкова популярни, уеб разработчиците трябва да се справят с много различни мрежи и да ги вземат под внимание по време на процеса на разработване на системата.

В контраст, потребителите на традиционните информационни системи често са служители, работещи в рамките на една организация или служба. Анализът

на изискванията на потребителите за WIS е далеч по-голямо предизвикателство за разработчиците.

1.2 Езици за реализиране на уеб приложения

Скриптовите езици се превръщат в програмни езици с общо предназначение поради тяхната гъвкавост и функционалност. Те се използват широко там, където времето на разработчиците е ценно и е необходимо да се избегне дълъг процес на компилация. С тях се пишат програми, предназначени за множество цели, като например:

- Вграждане в уеб страници. Тези програми не се компилират, а се интерпретират от браузъра при зареждането на страницата на клиентския компютър. С помощта на стандартните средства на скрипт езика и на обектите, дефинирани от DOM (Document Object Model), програмистът може да получи достъп до всеки елемент от уеб страницата и динамично да промени неговото съдържание и стилово оформление.
- Реализиране на различни динамични ефекти като показване и скриване на съдържание, придвижване и позициониране на обекти, анимация и т.н.
- Съединяване в едно цяло различни пакети приложни програми.
- Създаване на големи интернет приложения.
- Автоматизиране на различни процеси.

За да се разбере по-добре същността на скриптовите езици, е необходимо да се изходи от второто им наименование - езици за сценарии. Сценарий е последователност от операции, които потребителят може да извършва на компютъра. Сценариите обикновено се интерпретират, а не компилират, което позволява бързо и лесно изпълнение.

Скриптовите езици са програмни езици, които автоматизират определени задачи, които би трябвало да се изпълняват ръчно. Те са особено полезни в случаи, когато времето за разработка на програми е по-ценно от времето за изпълнение на програмата. Скриптовите езици могат да се използват масово в голям брой приложения, като например вграждане в уеб-страници, реализиране на динамични ефекти и създаване на големи Интернет приложения.

Скриптовите езици могат да бъдат динамично изпълними или предварително компилирани. Динамично изпълнимите скриптови езици четат инструкциите от програмния файл на минимални функционални блокове и изпълняват тези блокове без да четат останалия код. Предварително компилираните скриптови езици прочитат програмата в началото, компилират я цялата в машинен код или някакъв вътрешен формат и след това я изпълняват.

Скриптовите езици имат няколко предимства пред готовите програми, транслирани в машинен код. Те имат по-сложен инструментариум и поддържат по-прогресивни техники на програмиране, което съкращава времето за разработка на програми и ги прави по-ефективни. Скриптовите езици позволяват бързо да се откриват грешки в програмата и да се реализират доработки на кода без да се изчаква приключване на компилацията. Освен това, скриптовите езици са сравнително лесни за изучаване, дори и без съответната подготовка.

1.2.1 HTML (HyperText Markup Language)

HTML (HyperText Markup Language) е основният език за описание на уеб страници, който се използва за създаване на графични и текстови елементи на уеб страниците. HTML е стандарт в Интернет, като правилата за него се определят от международния консорциум W3C. Най-новата версия на стандарта е HTML5.

Описание на документа става чрез специални елементи или маркери, които се състоят от етикети или тагове (HTML tags). HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница, като заглавия, цитати, раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки и имат отварящ таг <h1> и затварящ таг </h1>.

HTML е основната технология, която контролира това, което уеб браузърът показва на екрана.

Основните категории команди в езика HTML, включени във версиите 3.2 и 4.01, са предназначени за специфициране на стилови формати и управление на текстовия поток (Flow Control), включване на графични изображения (Images), създаване на хипервръзки (Links), интегриране на аудио с външни графични обекти (Sound and Maps), вмъкване на видео таг (Video) и на youtube клипове директно от

тяхната страница чрез тага (Iframe), създаване на интерактивни формуляри (Forms), разделяне на документа на отделни полета (Frames), включване на външни приложения (Applet), написани на езика Java, и осъществяване на връзка с външни информационни структури (CGI-script).

HTML е език за маркиране, който се използва за създаване на уеб страници и документи. Той позволява на уеб браузърите да интерпретират и показват съдържанието на уеб страници, включително текст, изображения и други материали. HTML таговете се използват за оформление на съдържанието, като се определят характеристики като заглавия, параграфи, линкове и други елементи.

В допълнение към HTML, дизайнерите на уеб страници използват Cascading Style Sheets (CSS), за да задават стилове на съдържанието, като например цветове, шрифтове и разположение. CSS се използва за засилване на дизайна и визуалния аспект на уеб страници.

HTML файловете обикновено се пишат в текстови редактори и се хостват на сървъри, свързани към Интернет. Те съдържат текстово съдържание с маркери, които инструктират браузърите как да показват съдържанието на страницата. Уеб браузърите прочитат HTML документите и ги превръщат в уеб страници, като скриват HTML таговете и показват само съдържанието на страницата.

Едно от основните предимства на HTML е, че документите, създадени по този начин, могат да се разглеждат на различни устройства, включително компютри, таблети и мобилни устройства. Създаването на HTML-базирани уеб страници може да се извършва с помощта на обикновен текстов редактор или специализирани инструменти за създаване на уеб страници, като Microsoft FrontPage, Macromedia Dreamweaver, Notepad и други.

```
<form asp-action="Create" method="post">
    <input type="hidden" name="tripId" value="@ViewData["TripId"]" />
    <input type="hidden" name="returnUrl" value="@ViewBag.ReturnUrl" />
    <input type="hidden" name="returnUrlOriginal" value="@ViewBag.ReturnUrlOriginal" />
    <input type="hidden" asp-for="Score" id="rating-value" />

    <div class="form-group rating-group">
        <label class="rating-label">Рейтинг</label>
        <div class="star-rating">
            <div class="star-container">
                <i class="far fa-star star-item" data-rating="1"></i>
                <i class="far fa-star star-item" data-rating="2"></i>
                <i class="far fa-star star-item" data-rating="3"></i>
                <i class="far fa-star star-item" data-rating="4"></i>
                <i class="far fa-star star-item" data-rating="5"></i>
            </div>
            <div class="rating-text">Изберете рейтинг</div>
        </div>
        <span asp-validation-for="Score" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Comment" class="comment-label">Коментар</label>
        <div class="comment-container">
            <textarea asp-for="Comment" class="form-control comment-area" placeholder="Споделете вашето мнение за пътуването..."></textarea>
            <div class="comment-footer">
                <div class="comment-tips">
                    <i class="fas fa-lightbulb"></i> Съвет: Опишете какво ви хареса и какво може да се подобри
                </div>
                <div class="character-count">0/500</div>
            </div>
        </div>
        <span asp-validation-for="Comment" class="text-danger"></span>
    </div>

    <div class="review-actions">
        <a href="javascript:history.back()" class="btn btn-outline-secondary">
            <i class="fas fa-arrow-left"></i> Върни се обратно
        </a>
        <button type="submit" class="btn btn-primary submit-review">
            <i class="fas fa-paper-plane"></i> Изпрати ревю
        </button>
    </div>
</form>
```

Фигура 1 Пример за HTML форма за създаване на модел “Rating“

1.2.2 CSS (Cascade style sheets)

CSS (Cascading Style Sheets) е език за описание на стилове, който се използва основно за определяне на представянето на документи, написани на езици за маркиране (като HTML и XML). Официалната спецификация на CSS се поддържа от W3C (World Wide Web Consortium).

CSS позволява на уеб дизайнерите да определят визуални стилове на HTML елементите, като например шрифтове, размери, цветове, фонове и други. Кодът на CSS се състои от последователност от стилови правила, като всяко правило се състои от селектор, последван от декларации, като всяка декларация се състои от свойство и стойност.

CSS е много удобен за форматиране на текст, шрифтове, списъци, таблици и други елементи на уеб страницата. Комбинацията от селектор и декларации определя стиловете, които ще се приложат върху избраните HTML елементи.

Синтаксисът на CSS е много прост и лесен за разбиране. Всяко правило започва със селектор, който определя върху кой елемент ще се приложат стиловете.

След това следват декларации, като всяка декларация е комбинация от свойство и стойност, разделени с двоеточие и заключени в скоби.

Фигура 2 Пример за CSS код

```
<span class="trip-detail-value">
    @if (item.FreeSeats > 0)
    {
        <span style="color: var(--secondary);">@item.FreeSeats</span>
    }
    else
    {
        <span style="color: var(--danger);">@item.FreeSeats</span>
    }
</span>
```

Фигура 3 CSS код приложен чрез “style” атрибут

Ако има повече от един стил, специфициран за един HTML елемент, ще се използва този с най-висок приоритет. Приоритетът е както следва:

1. Стиловете, вградени в HTML елементите с атрибута style.
2. Вътрешните стилове, зададени във <style> елемент в същия HTML файл.
3. Външните стилове, зададени в отделен CSS файл.
4. Стилът по подразбиране на браузъра.

HTML етикетите са първоначално създадени, за да определят съдържанието на документа, като използват синтаксис като <h1> за големи заглавия и <p> за параграфи. С добавянето на нови спецификации към HTML и увеличаването на изискванията към визията на сайта, стана все по-трудно да се създават страници само с HTML. Тук влиза в игра CSS, който ни позволява да разделим стиловете от съдържанието и да създадем по-сложни дизайни. CSS ни позволява да съхраняваме повтарящите се форматирания във външен файл и така да не пишем всеки път

атрибутите за тях. Това ни спестява работа и ни позволява да правим по-лесни промени в изгледа на уебсайта.

Синтаксисът на CSS е съставен от три части: селектор (selector), характеристика (property) и стойност (value). (фиг. 2)

Селекторът е нормален HTML елемент (таг), който трябва да се промени, характеристиката е атрибутът за смяна, като всяка характеристика си има стойност. Характеристиката и стойността са отделени с двоеточие и са заградени с фигурни скоби:

```
div {color: red}
```

Ако стойността представлява низ съдържащ няколко думи –необходимо е да се огради с кавички:

```
span {font-family: "sans serif"}
```

Ако трябва да се специфицира повече от една характеристика, енеобходимо да се разделят с “;”.

```
H1 {text-align:center; color:blue}
```

Селекторите с еднакви характеристики могат да се групират по между си и така да се спести писане на код. Това става като се разделят съсзапетая. В следващия пример са групирани всички header елементи. След прилагането на стила съответния текст ще бъде оцветен в зелено:

```
h1,h2,h3,h4,h5,h6 { color: red }
```

CSS притежава още едно важно свойство, така наречения клас Селектор (class selector). С него може да се зададат различни стилове за един и същи HTML елемент в уеб страницата. Ако например имаме два параграфа, единият от тях е с дясно подравняване а другият с централно, с клас селектор това би изглеждало така:

```
p.right {text-align: right}

p.center {text-align: center}
```

Друга полезна опция в CSS е id селекторът. Той се различава от класа

селектор, по това че последния може да се приложи върху няколко елемента върху страницата, а id селекторът е приложим само за един. Той трябва да е уникален в рамките на уеб страницата. Долу е дефиниран стил, който свързва елемента <h2> с id стойността “heading2”

```
h2# heading2{text-align: start; color: blue }
```

Добра практика е като се пише код да се оставят коментари, такава възможност притежава и CSS. Коментарът започва с "/*" и завършва с "*/". Всичко между тях се игнорира от браузъра.

1.2.3 Bootstrap

Bootstrap е библиотека с отворен код, която съдържа дизайн шаблони за HTML и CSS, включително компоненти за интерфейса на уеб-приложения и/или уеб-страници, като типография, форми, бутони, навигация и други. Въпреки това, има няколко вградени файлове, които не е препоръчително да се редактират, като файлове с разширения .css и .js. Библиотеката включва няколко JavaScript компонента във формата на jQuery плъгини, които предоставят допълнителни елементи на потребителския интерфейс, като диалогови прозорци и пояснения, и разширяват функционалността на някои от съществуващите елементи на интерфейса.

1.2.4 JavaScript

JavaScript е интерпретируем език за програмиране, разпространяван с повечето уеб браузъри. Поддържа обектно ориентиран и функционален стил на програмиране. Създаден е в Netscape през 1995 г. Най-често се прилага към HTML кода на интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. Javascript е стандартизиран под името EcmaScript.

JavaScript е програмен език, който позволява динамична промяна на поведението на браузъра в рамките на дадена HTML страницата. JavaScript се зарежда, интерпретира и изпълнява от уеб браузъра, който му осигурява достъп до Обектния модел на браузъра. JavaScript функции могат да се свържат със събития

на страницата (например: движение/натискане на мишката, клавиатурата или елемент от страницата, и други потребителски действия). JavaScript е най-широко разпространеният език за програмиране в интернет. Прието е JavaScript програмите да се наричат скриптове.

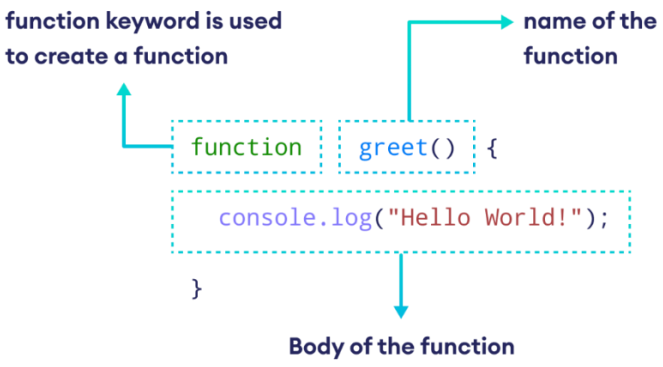
JavaScript може да влияе на почти всяка част от браузъра. Браузъра изпълнява JavaScript кода в цикъла на събития т.е. като резултат от действия на потребителя или събития в браузъра (например document.onLoad).

Основни задачи в повечето JavaScript приложения са:

- Зареждане на данни чрез AJAX.
- Ефекти с изображения и HTML елементи: скриване/показване, пренареждане, влачене, слайд шоу, анимация и много други.
- Управление на прозорци и рамки.
- Разпознаване на възможностите на браузъра.
- Използване на камерата и микрофона.
- Създаване на 3D графики WebGL.
- По-добър и гъвкав потребителски интерфейс

Какво не може да се прави с помощта на JavaScript:

- Не може да се записва информация на потребителския компютър или отдалечения сървър.
- Не може да се запазва информация директно в отдалечена база данни.
- Не може да се стартират локални приложения.



Фигура 4 Пример за JavaScript функция

1.2.5 Езикът C#

C# е език за програмиране, разработен от Microsoft през 2000 година. Той е предназначен да бъде силно типизиран, обектно-ориентиран език, който може да се използва за различни цели - създаване на Windows приложения, уеб приложения, игри, мобилни приложения и други видове софтуер. C# е основен елемент от платформата .NET, която позволява на програмистите да създават приложения, които работят на различни операционни системи.

Езикът C# е синтаксисът на програмиране, който е много подобен на C и C++. Той е дизайниран за да бъде по-модерен и сигурен от тези езици, като същевременно запазва много от техните мощни възможности. Един от ключовите аспекти на C# е, че той е език със строго типизирани променливи, което означава, че програмата може да се компилира само ако типовете на променливите в програмата са коректни.

Освен това C# е езикът, който се използва за създаване на приложения, които се базират на архитектурата на Microsoft Windows. Той е много подходящ за създаване на Windows десктоп приложения и уеб приложения, които използват технологии като ASP.NET и Silverlight. Също така, C# е използван в голяма степен за разработване на игри в платформата Unity, която е популярна за създаване на мобилни игри.

Един от главните достойнства на C# е, че той има много добра интеграция с други езици за програмиране, което позволява на програмистите да използват различни езици и технологии, когато разработват софтуер. Например, C# може да се използва за създаване на програми, които използват бази данни, с помощта на технологии като LINQ (Language Integrated Query).

Също така, C# има богата библиотека от класове, която програмистите могат да използват за да улеснят програмирането. Библиотеката включва класове за работа с графични интерфейси, мултимедийни файлове, мрежови връзки и много други. Освен това, C# разполага с много добра документация и поддръжка от страна на Microsoft, която предоставя на програмистите широк избор от ресурси за учене, като курсове, уроци и форуми.

В последните години, C# се е развивал много бързо и предоставя нови функционалности за програмистите. Например, C# 8.0 представи нова функционалност като nullable типове, изключения от шаблони и асинхронни поточни методи. C# 9.0, който е пуснат през 2020 година, включва нови възможности като записи, модели на образци, улеснена употреба на null безопасност и много други.

C# е изключително мощен език за програмиране, който се използва в много различни области на софтуерното инженерство. Той е изключително полезен за програмисти, които работят върху създаването на мобилни приложения, игри, уеб приложения и Windows приложения. C# има силна поддръжка и развитие от Microsoft, което го прави един от най-предпочитаните езици за програмиране в индустрията.

1.2.6 Технологията ASP.NET

ASP.NET е технология за уеб разработка, разработена от Microsoft, която позволява на програмистите да създават динамични уеб приложения и услуги. Технологията е базирана на програмния език C# или VB.NET и използва Microsoft .NET Framework, който предоставя много библиотеки и функционалности за програмиране на уеб приложения.

ASP.NET предлага различни подходи за програмиране на уеб приложения. Един от тях е ASP.NET Web Forms, който е бил първоначално предназначен да подобри процеса на програмиране на уеб приложения. Web Forms използва модел на програмиране, който е близък до този на програмирането на Windows Forms, където програмистът може да използва графичен потребителски интерфейс и компоненти. Сега, обаче, ASP.NET MVC е по-популярен подход за програмиране на уеб приложения и се състои от модел-изглед-контролер архитектура, която е подобна на модела на програмиране на Ruby on Rails.

ASP.NET MVC позволява на програмистите да разделят приложението на три слоя: модел, който управлява данните и извършва операции с тях, изглед, който представя данните на потребителя и контролер, който се грижи за приемането на заявките от потребителите и извършва съответните операции. Тази архитектура прави приложението лесно поддържаемо и просто за разбиране.

ASP.NET Core е нова версия на ASP.NET, която е направена да бъде по-бърза и лека от предишните версии. Тя е съвместима със множество операционни системи, включително Windows, Linux и macOS, и може да се използва за създаване на уеб приложения, уеб услуги, микросервизи и други.

С помощта на ASP.NET, програмистите могат да създават различни видове уеб приложения, като например онлайн магазини, социални мрежи, уеб услуги, уеб приложения за управление на данни и много други. ASP.NET предоставя множество вградени функционалности, като например автоматично управление на сесии и кеширане на данни. Това улеснява работата на програмистите, тъй като те не трябва да се занимават с тези задачи и могат да се концентрират върху разработването на бизнес логиката на приложението.

Технологията е съвместима с много бази от данни, като например Microsoft SQL Server, MySQL, Oracle и други. Това позволява на програмистите да изберат базата от данни, която най-добре отговаря на нуждите на приложението.

ASP.NET има голяма общност от програмисти, които споделят знания и опит в разработката на уеб приложения. Microsoft предоставя множество ресурси и инструменти, които могат да бъдат използвани за учене и подобряване на уменията в програмирането на уеб приложения.

С помощта на ASP.NET програмистите могат да създадат много гъвкави и мащабируеми уеб приложения, които могат да се използват от малки стартъп компании до големи корпорации. Технологията предоставя много инструменти за сигурност и оптимизация на уеб приложенията, които могат да се използват за защита на данните и оптимизация на производителността на приложението.

Съществуват множество уеб хостинг услуги, които поддържат ASP.NET приложения, което прави деплоя на приложенията много лесен и удобен.

В заключение, ASP.NET е мощна технология за уеб разработка, която предоставя на програмистите голям брой инструменти и функционалности, които им помагат да създават гъвкави, мащабируеми и сигурни уеб приложения. ASP.NET е подходяща за разработка на уеб приложения от различни отрасли, като например онлайн магазини, социални мрежи, уеб услуги и много други.

Практическа част - Концепция

Задание

Целта на дипломния проект е създаването на уеб базирана система за споделено пътуване, която да улесни потребителите в намирането на спътници за пътуване в страната. Системата предлага възможност за публикуване, търсене и резервиране на споделени пътувания между различни градове в България, като по този начин улеснява транспорта, намалява разходите и допринася за опазване на околната среда чрез по-ефективно използване на превозните средства.

Задачи, които трябва да бъдат решени:

- В системата има четири вида потребители: администратор, шофьори, пътници и гости. Само регистрираните потребители (шофьори и пътници) имат достъп до основната функционалност на приложението, като създаване на пътувания и резервиране на места. Гостите могат да разглеждат списъка с налични пътувания, но нямат възможност да взаимодействат активно със системата.
- Администраторът е първоначално зададен и притежава достъп до административен панел, от който може да управлява потребителите, да следи активността им и да променя техните роли. Той може да търси потребители по потребителско име, имейл или роля, както и да ги филтрира. Не може да създава или назначава други администратори.
- Шофьорите могат да публикуват нови пътувания, като посочват начален и краен град, дата и час на тръгване, цена, свободни места и друга допълнителна информация. Те могат да редактират и премахват свои пътувания, както и да виждат списък с пътници, заявили участие.
- Пътниците могат да разглеждат наличните пътувания и да резервират място в избрано от тях пътуване. При потвърждение, те получават информация за шофьора, мястото на тръгване и други детайли. Един потребител не може да се запише на повече от едно пътуване със същия маршрут и час.
- Всеки маршрут се характеризира с начален и краен град, както и със средни спирки (ако има такива). Градовете се избират от предварително зададен списък, за да се избегнат правописни грешки и несъответствия.

- Формите за създаване на пътувания и резервации преминават през валидация – не се приемат празни полета, отрицателни стойности или нелогични дати (напр. минали дати или крайна точка, съвпадаща с началната).
- Всички пътувания имат статус – „предстоящо“, „в процес“, „завършено“ или „отменено“. Статусите се променят автоматично от системата според текущата дата и състоянието на резервациите.
- Приложението включва възможност за търсене и филтриране на пътувания по градове, дата, цена и налични места, което улеснява потребителите при намиране на най-подходящия вариант.

Допълнителни разработки

Като допълнително към основната функционалност е разработена начална страница, която цели да подтикне посетителите да се регистрират, като визуализира ползите от използването на системата за споделено пътуване.

Регистрираните потребители разполагат със собствен потребителски профил, в който могат да видят историята на пътуванията си – както в ролята на шофьор, така и в ролята на пътник. След приключване на пътуването, пътниците имат възможност да оставят оценка и коментар за шофьора, което подпомага изграждането на репутация и доверие в системата.

Допълнително е реализирана страница, която визуализира всички регистрирани шофьори, като към всеки от тях е достъпен списък с предстоящи и минали пътувания. Това улеснява пътниците при избора на шофьор и предоставя по-голяма прозрачност относно активността на всеки потребител в системата.

Шофьорите и пътниците имат достъп до специализирани панели, от които могат да управляват и следят своите пътувания – шофьорите виждат всички направени заявки към техните пътувания, а пътниците могат лесно да управляват резервациите си.

Реализация

Модели и езици за бази данни и СУБД

База данни представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. Терминът база данни се дефинира като записи, организирани в систематичен ред, който позволява на компютърни програми да извличат информация по зададени критерии. За по-добро извличане и сортиране на данните, всеки запис се организира като множество от елементи на данни (факти). Извлечените елементи в отговор на въпросите (запитванията) стават информация, която може да се използва за вземане на решения.

Системата за управление на бази данни (СУБД) е компютърна програма, която се използва за управление и задаване на въпроси в базите данни. От потребителска гледна точка СУБД трябва да притежава следните свойства:

- Устойчива памет - данните трябва да се съхраняват независимо от процесите; базите данни трябва да позволяват ефективен достъп до много големи обеми от данни - времето на достъп до данните трябва да е независимо от техния обем.
- Програмен интерфейс - СУБД предоставя мощен език за заявки на потребителя или на приложната програма, която използва данните.
- Управление на транзакции - СУБД поддържа едновременен достъп до данните; той се осъществява чрез множество процеси, наречени транзакции; всеки потребител, който работи с данните трябва да инициира такъв процес; СУБД поддържа изолация на транзакциите - всяка транзакция е атомарна, т.е. тя или завършва изцяло или бива изцяло отхвърлена; ако по време на изпълнение на една транзакция възникне проблем тя се отменя, заедно с всички изменения, които е причинила върху базата от данни.

От съществено значение за ефективното използване на базите данни е оптимизацията на заявките към тях. Това се постига чрез изграждане на индекси върху най-често използваните полета в таблиците, които ускоряват търсенето на записи по тези полета.

Освен това, в съвременните бази данни се използват различни технологии за съхранение на данни, като релационните, нерелационните и графовите бази данни. Всеки от тези видове бази данни има свои предимства и недостатъци и е подходящ за различни типове приложения.

В заключение, базите данни са от съществено значение за съвременните информационни системи и играят важна роля във вземането на решения в бизнеса и други области. Оттам и необходимостта от постоянно развитие на технологиите за управление на бази данни и оптимизация на тяхната работа.

2.1 Модели и езици за бази данни

Базите данни са съвкупност от организирани данни, които се използват за съхраняване, манипулиране и извличане на информация. Един от най-важните аспекти при проектирането на бази данни е изборът на подходящ модел и език за реализация на базата данни.

2.1.1 Релационни бази данни (RDBMS)

Този модел е най-често използваният модел за създаване на бази данни. Релационните бази данни работят върху таблици, които имат редове и колони. Всяка таблица има уникален идентификатор, наречен "ключ", който свързва редовете с други таблица чрез "връзки". RDBMS използва SQL език за манипулиране на данните и създаване на заявки.

```
var overlappingTrips = await context.Trips
    .Where(t => t.DriversId == trip.DriversId &&
                (t.StatusTrip == TripStatus.Upcoming || t.StatusTrip == TripStatus.Ongoing))
    .ToListAsync();
```

Фигура 5 LINQ заявка (преобразува се в SQL заявка от Entity Framework Core)

Microsoft SQL Server е релационна база данни, която използва SQL езика. Тя е налице само за Windows операционните системи и може да се използва с Microsoft Azure за облачно съхранение на данни. Microsoft SQL Server е използвана в големи корпоративни приложения и включва функционалности за управление на данните, включително сигурност и репликация на данните.

2.1.2 Нерелационни бази данни

Нерелационните бази данни (NoSQL бази данни) са алтернатива на традиционните релационни бази данни. Те се използват за съхранение на големи

масиви от структурирани и/или неструктурирани данни, които не могат да бъдат ефективно съхранени и обработвани с традиционни релационни бази данни. В следващите редове ще представим някои от най-популярните видове нерелационни бази данни:

Документни бази данни: Тези бази данни съхраняват данните в документи, които са обикновено в JSON или XML формат. Те са много гъвкави и удобни за съхранение на неструктурирани данни като текст, снимки, видео и други.
Ключ-стойностни бази данни: Тези бази данни съхраняват данни в стойности, които могат да бъдат достъпвани чрез ключове. Те са много бързи и ефективни при работа с много голям обем данни.

Колонни бази данни: Тези бази данни съхраняват данни в колони, вместо в редове, което ги прави много ефективни при работа с големи обеми данни и търсене на конкретна информация.

Графови бази данни: Тези бази данни съхраняват данни като графи, като връзките между данните са важни. Те са много полезни за съхранение на данни за социални мрежи, транспортни мрежи, връзки между хора и други.

Нерелационните бази данни са много гъвкави и удобни за съхранение на различни видове данни, които не могат да бъдат съхранени ефективно в релационни бази данни. Те са много полезни за приложения, които работят с големи обеми данни, които се променят често и изискват бърз достъп и търсене на информация.

В заключение, изборът на база данни зависи от специфичните нужди на приложението, като се вземат предвид също така и неговите мащабируемост, сигурност и производителност. Различните модели и езици за бази данни предоставят различни функционалности, които могат да се използват за различни видове приложения.

2.2 СУБД

СУБД (Система за Управление на Бази от Данни) е програмен продукт, който се използва за управление на данните в една или няколко свързани бази от данни. Тези бази от данни съхраняват информация, която може да бъде много разнообразна - от текстови документи и изображения до структурирани данни като таблици и списъци. СУБД-тата осигуряват удобен интерфейс за работа с данните,

който може да бъде използван от различни потребители - от начинаещи до професионалисти.

Най-известните СУБД-та са MySQL, PostgreSQL, Oracle, Microsoft SQL Server, MongoDB, SQLite и др. Всяко от тях има свои предимства и недостатъци и е подходящо за различни сценарии на използване.

Microsoft SQL Server е СУБД, която е разработена специално за използване на Windows операционни системи. Тя е платена, но се предлага в различни пакети в зависимост от нуждите на потребителите. Microsoft SQL Server е много бърза и мощна, като има вградени инструменти за управление и администриране на бази от данни. Тя е също така съвместима с различни програмни езици и платформи, като .NET и Java.

MongoDB е NoSQL СУБД, която се използва за съхранение на голямо количество неструктурирани данни като документи, изображения и видео файлове. MongoDB е много гъвкава и позволява много бързо и лесно мащабиране на приложенията. Тя е идеална за уеб приложения, които работят с големи количества неструктурирани данни.

СУБД-тата играят много важна роля в съвременните информационни технологии и са необходими за управление на големи количества данни. В зависимост от нуждите на потребителя, той може да избере най-подходящото СУБД за неговите нужди - от малки приложения до големи корпоративни системи. Въпреки че СУБД-тата могат да бъдат доста сложни, те предлагат изключително голяма гъвкавост и функционалност, което ги прави важен инструмент в информационната технология.

Реализиране на разработка на приложение „Споделено Пътуване“

Фирма „Рион“ извършва всякакви видове ремонтни дейности за дома в гр. Велико Търново. Управителния съвет взема решение дейността на фирмата да се разшира и в други градове на страната. За релаизиране на тази задача, фирмата се нуждае от приложение, което освен да рекламира дейността ѝ, трябва да осигурява връзка с клиенти.

3.1 Design pattern

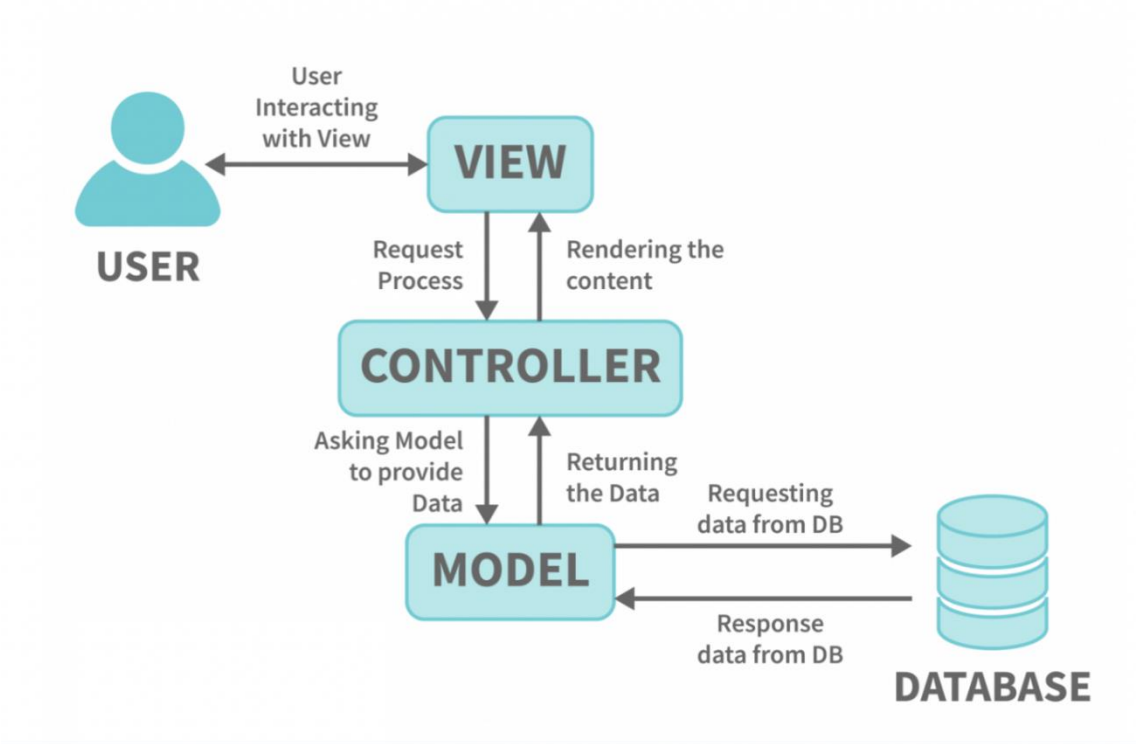
Design pattern (дизайн шаблон) е един препоръчителен начин за решаване на често срещани проблеми в софтуерното проектиране. Те представляват решения на проблеми, които са се появили многократно в различни контексти на даденото софтуерно приложение.

3.1.1 MVC design pattern

MVC е съкращение от "Model-View-Controller" (Модел-Изглед-Контролер) и е един от най-популярните дизайн патерни за програмиране на уеб приложения.

Той разделя приложението на три основни компонента: Модел, Изглед и Контролер.

- Моделът представлява данните и бизнес логиката на приложението. Той обработва всички заявки за данни от Контролера и осъществява комуникацията с базата данни.
- Изгледът е отговорен за визуализацията на данните и общуването с потребителя. Той използва Модела, за да взема необходимата информация и да я представи на потребителя по желания начин.
- Контролерът посредничи между Изгледа и Модела, обработва заявките на потребителя и връща съответните отговори. Той работи с Изгледа, за да осигури правилното визуализиране на информацията и с Модела, за да извършва операции с данните.



Фигура 6 Структурата MVC

MVC е особено полезен за големи приложения, тъй като разделя логиката на приложението на отделни компоненти, което улеснява тестването и поддръжката на кода.

В C# програмирането може да се използва MVC с помощта на ASP.NET Core фреймуърка, който предоставя готова инфраструктура за работа с Модела, Изгледа и Контролера, както и други полезни функционалности за разработката на уеб приложения.

За да се използва ASP.NET Core фреймуъркът, е необходимо да се създаде проект на Visual Studio и да се избере шаблон за ASP.NET Core Web App.

След това може да се започне работа върху Модела, Изгледа и Контролера.

Моделът може да бъде съставен от класове, които представят съответните обекти в приложението и съдържат методи за работа с тях.

Изгледът може да бъде съставен от HTML файлове и Razor шаблони, които използват C# код за визуализиране на данните от Модела.

Контролерът може да бъде съставен от класове, които обработват заявките от Изгледа и извършват нужните операции върху Модела, като връщат подходящи отговори на Изгледа.

ASP.NET MVC фреймуъркът използва маршрутизация за да определи кой Контролер да се използва за обработка на всяка заявка.

MVC е много гъвкав и може да се използва за разработка на уеб приложения от всякакъв тип и сложност. Освен това, той се използва широко в индустрията и предоставя стандартизиран начин за организиране на кода и поддръжка на приложенията.

3.1.2 Service design pattern

Service design pattern (Шаблон за проектиране "Сервиз") е дизайн патерн, който се използва за изграждане на сложни приложения, където е необходима комуникация между различни компоненти или модули.

Service pattern се основава на идеята за изграждане на независими сервизи, които предоставят определени функционалности на останалите части на приложението. Тези сервизи имат ясно дефинирани интерфейси, които други части на приложението могат да използват, за да получат необходимите им данни.

Service pattern обикновено се състои от следните елементи:

- Интерфейс - дефинира методите, които сервизът предлага на другите компоненти на приложението.
- Реализация - включва кода, който извършва необходимите операции и имплементира методите от интерфейса.
- Инжектиране на зависимости - това е механизмът, чрез който приложението може да свърже различни компоненти помежду си и да ги интегрира със сервизите.

Service pattern се използва за улесняване на комуникацията между компонентите на приложението и за повторно използване на кода. Той улеснява тестването и поддръжката на приложението, като осигурява стандартизиран начин за достъп до различни функционалности на приложението.

В C# програмирането Service pattern се използва често в комбинация със Dependency Injection (DI). Dependency Injection е механизъм, който автоматично

свързва различните компоненти на приложението помежду си и ги инжектира в нужните места в кода.

Така Service pattern и DI заедно помагат за постигане на добра архитектура на приложението, улесняват мащабируемостта му и увеличават гъвкавостта и поддръжката на кода.

За да бъде успешно използван Service pattern, трябва да бъде правилно дефиниран интерфейсът на сервизите и да се спазват добрите практики за организиране на кода. Това включва използването на правилни имена на класовете и методите, разделянето на кода на подходящи слоеве, дефинирането на подходящи модели на данните и други.

3.1.3 Service в приложението “Споделено Пътуване“

3.1.3.1 TripSchedulerService

Фоновата услуга в системата има за цел автоматичното създаване на нови пътувания въз основа на предварително дефинирани повтарящи се шаблони. Това осигурява удобство за шофьори, които извършват регулярни курсове, тъй като не е необходимо ръчно да създават всяко пътуване. Основната логика на услугата се изпълнява от метода ExecuteAsync, който работи във фонов режим и се активира периодично на всеки 30 секунди.

При всяко изпълнение, услугата проверява за шаблони на повтарящи се пътувания, при които моментът за следващо изпълнение вече е настъпил. За всеки намерен шаблон системата анализира дали шофьорът, свързан с него, вече има насрочени пътувания, които биха се припокрили по време с новото, предстоящо пътуване. Тази проверка се извършва чрез метода UserHasNoOverlappingTrips.

```
0 references
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    while (!stoppingToken.IsCancellationRequested)
    {
        _logger.LogInformation("Checking for recurring trips at {Time}", DateTime.Now);

        using (var scope = _services.CreateScope())
        {
            var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
            var Now = DateTime.Now;

            var allRecurringTrips = await context.Trips
             .Where(t => t.IsRecurring && t.RecurrenceInterval != "00:00:00" && t.NextRunDate.HasValue)
             .ToListAsync();

            var recurringTrips = allRecurringTrips
                .Where(t => TimeSpan.TryParse(t.RecurrenceInterval, out TimeSpan interval)
                 && t.tripSchedule.Add(interval) <= Now)
                .ToList();

            foreach (var trip in recurringTrips)
            {
                _logger.LogInformation("Processing recurring trip {TripId}", trip.Id);

                var newDepartureTime = trip.NextStart;
                TimeSpan reccurenceInterval = TimeSpan.Parse(trip.RecurrenceInterval);

                if (await UserHasNoOverlappingTrips(trip))
                {
                    var newTrip = new Trip
                    {
                        Driver = trip.Driver,
                        DriversId = trip.DriversId,
                        StartPosition = trip.StartPosition,
                        Destination = trip.Destination,
                        DepartureTime = newDepartureTime,
                        ReturnTime = newDepartureTime.Add(trip.ReturnTime - trip.DepartureTime),
                        Price = trip.Price,
                        TotalSeats = trip.TotalSeats,
                        FreeSeats = trip.TotalSeats,
                        CarModel = trip.CarModel,
                        PlateNumber = trip.PlateNumber,
                        ImagePath = trip.ImagePath,
                        StatusTrip = TripStatus.Upcoming,
                        CreatedDate = Now,
                        IsRecurring = false
                    };
                    context.Trips.Add(newTrip);
                }

                trip.NextStart = trip.NextStart.Add(reccurenceInterval);
                trip.tripSchedule = trip.tripSchedule.Add(reccurenceInterval);

                await context.SaveChangesAsync();
            }
        }
        await Task.Delay(TimeSpan.FromSeconds(30), stoppingToken);
    }
}
1 reference
```

Фигура 7 Методът “ExecuteAsync“

Ако методът установи, че няма припокриващи се пътувания, се създава ново пътуване, базирано на данните от шаблона, с актуализирани дати и часове. След успешното създаване на пътуването, системата обновява времето за следващо изпълнение на съответния шаблон, така че да е готов за следващото автоматично създаване. Всички промени, включително новите записи за пътуванията и актуализираните стойности в шаблоните, се запазват в базата данни.

Методът UserHasNoOverlappingTrips играе ключова роля в този процес. Неговата задача е да провери дали даден шофьор вече има пътувания, които се припокриват с времето на новосъздаваното пътуване. Ако няма такова припокриване, методът връща стойност true, което позволява на системата да създаде новото пътуване. В случай на припокриване, създаването се пропуска, за да се избегнат конфликти в графика на шофьора.

```
private async Task<bool> UserHasNoOverlappingTrips(Trip trip)
{
    using (var scope = _services.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

        var overlappingTrips = await context.Trips
            .Where(t => t.DriversId == trip.DriversId &&
                        (t.StatusTrip == TripStatus.Upcoming || t.StatusTrip == TripStatus.Ongoing))
            .ToListAsync();

        var NewDepartureTime = trip.NextStart;
        var NewReturnTime = NewDepartureTime.Add(trip.ReturnTime - trip.DepartureTime);

        bool hasOverlap = overlappingTrips.Any(t =>
            (NewDepartureTime >= t.DepartureTime && NewDepartureTime <= t.ReturnTime) ||
            (NewReturnTime >= t.DepartureTime && NewReturnTime <= t.ReturnTime) ||
            (NewDepartureTime <= t.DepartureTime && NewReturnTime >= t.ReturnTime)
        );

        return !hasOverlap;
    }
}
```

Фигура 8 Методът “UserHasNoOverlappingTrips“

3.1.3.2 TripStatusUpdaterService

TripStatusUpdaterService е фонова услуга, която работи автоматично и има за цел да актуализира статуса на пътуванията според текущото време. Тя се изпълнява периодично, като на всеки 30 секунди проверява базата данни за пътувания, чието състояние трябва да бъде променено.

Методът в услугата е ExecuteAsync, който съдържа логиката за актуализиране. Ако е достигнато времето за тръгване на дадено пътуване, неговият статус се променя от „Предстоящо“ (Upcoming) на „В процес“ (Ongoing). Съответно, ако е достигнато времето за връщане, статусът се обновява от „В процес“ на „Завършено“ (Completed).

```
0 references
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    while (!stoppingToken.IsCancellationRequested)
    {
        try
        {
            using (var scope = _scopeFactory.CreateScope())
            {
                var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

                var now = DateTime.Now;

                var ongoingTrips = context.Trips
                    .Where(t => t.DepartureTime <= now && t.StatusTrip == TripStatus.Upcoming)
                    .ToList();

                foreach (var trip in ongoingTrips)
                {
                    trip.StatusTrip = TripStatus.Ongoing;
                }

                var finishedTrips = context.Trips
                    .Where(t => t.ReturnTime <= now && t.StatusTrip == TripStatus.Ongoing)
                    .ToList();

                foreach (var trip in finishedTrips)
                {
                    trip.StatusTrip = TripStatus.Finished;
                }

                if (ongoingTrips.Any() || finishedTrips.Any())
                {
                    await context.SaveChangesAsync();
                    _logger.LogInformation("Статусът на пътешествието е сменен успешно.");
                }
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Промяната на статуса на пътешествието не беше успешна.");
        }

        await Task.Delay(TimeSpan.FromSeconds(30), stoppingToken);
    }
}
```

Фигура 9 Методът "ExecuteAsync"

След всяка такава актуализация, промените се записват в базата данни. Услугата също така логва информация за успешно извършените промени или за възникнали грешки по време на процеса.

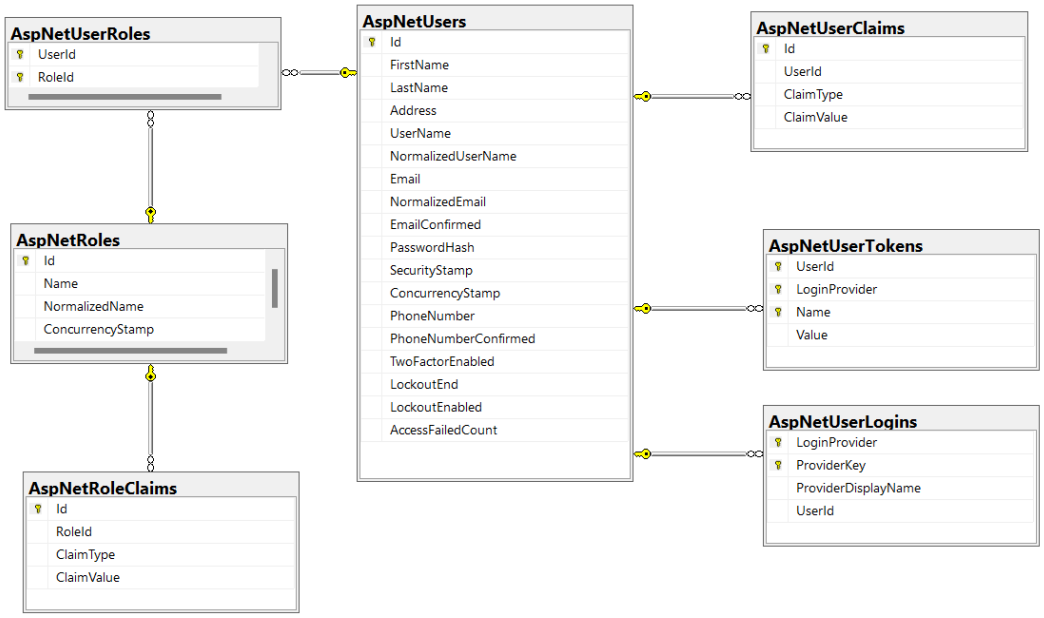
Service pattern е мощен инструмент за управление на сложни приложения и осигуряване на стабилно и мащабируемо приложение. Правилното му използване може да подобри архитектурата на приложението, да улесни тестването и поддръжката му и да ускори разработката на нови функционалности.

3.2 Модели в приложението „Споделено пътуване“

Модел (Model) - това е компонентът, който управлява данните в приложението. Той се грижи за извличането, обработката и запазването на данните. Моделът може да предоставя данните на контролера или директно на изгледа. За тази цел е използван Microsoft SQL Server в комбинация с Entity Framework Core, в контекста на ASP.NET Core приложение. Това позволява създаване на модел на данните, управление на базата данни и извършване на операции като създаване, четене, актуализиране и изтриване на данни по лесен и удобен начин. Entity Framework Core предоставя мощен ORM (Object-Relational Mapping) инструмент, който улеснява взаимодействието с базата данни, като позволява манипулирането на данните с помощта на обекти и класове, вместо да се пишат SQL заявки директно. Базата от данни се състои общо от 11 таблици. 7 от тях са автоматично генерирани и 6 създадени в процеса на разработка.

Седемте автоматично генерирани таблици:

- AspNetUsers – Съдържа информация за потребителите в системата.
- AspNetRoles – Съхранява ролите на потребителите.
- AspNetUserRoles – Установява връзката между потребителите и техните роли.
- AspNetRoleClaims – Съдържа допълнителна информация за ролите.
- AspNetUserClaims – Съдържа допълнителна информация за потребителите.
- AspNetUserLogins – Съхранява информация за външните идентификатори на потребителя.
- AspNetUserTokens – Съхранява токени за потвърждаване на идентичността на потребителите.



Фигура 10 Автоматично създадените таблици

3.2.1 Потребители (AspNetUsers)

Потребителите, които ASP.NET създава автоматично съдържат следните полета:

- Id
- UserName
- NormalizedUserName
- EmailConfirmed
- PasswordHash
- SecurityStamp
- ConcurrencyStamp
- PhoneNumber
- PhoneNumberConfirmed
- TwoFactorEnabled
- LockoutEnd
- LockoutEnabled
- AccessFailedCount

Ние искаме потребителите да имат следните допълни полета:

- Собствено име
- Фамилно име
- Снимка (ако е шофьор)
- Дата на приемане (когато е станал шофьор)
- Роля (за нея отговаря автоматично създадена таблица)
- Позиция
- Участници в пътуването - Списък с всички потребители, записани за участие в пътуването
- Заявки– Списък със заявки от потребители за участие в пътуването
- Оценки – Списък с оценки, дадени за това пътуване
- Заявки за шофьори – Списък със заявки от потребители за повдигане към роля шофьор

За целта създаваме клас ApplicationUser, в който добавяме допълнителните полета. Класът ApplicationUser наследява класа IdentityUser, който е дефаултния клас за потребители от ASP.NET.

```
96 references
public class ApplicationUser : IdentityUser
{
    48 references
    public string FirstName { get; set; } = string.Empty;

    48 references
    public string LastName { get; set; } = string.Empty;
    48 references
    public string Position { get; set; } = string.Empty;

    39 references
    public string? ImagePath { get; set; }
    18 references
    public DateTime? DateOfDriverAcceptance { get; set; }

    1 reference
    public ICollection<TripParticipant>? TripParticipants { get; set; }

    1 reference
    public ICollection<Request>? Requests { get; set; }
    2 references
    public ICollection<RequestDriver>? RequestDrivers { get; set; }
    1 reference
    public ICollection<Rating>? Ratings { get; set; }
}
```

Фигура 11 Моделът “Потребители“

3.2.2 Пътешествие

Пътешествието се характеризира с:

```
26 references
public class Trip
{
    [Key]
    59 references
    public int Id { get; set; }

    50 references
    public string? DriversId { get; set; } = string.Empty;

    [ForeignKey(nameof(DriversId))]
    15 references
    public ApplicationUser? Driver { get; set; }

    [Required(ErrorMessage = "Началната позиция е задължителна")]
    31 references
    public string StartPosition { get; set; }

    [Required(ErrorMessage = "Дестинацията е задължителна")]
    31 references
    public string Destination { get; set; }

    [Required(ErrorMessage = "Часът на заминаване е задължителен")]
    55 references
    public DateTime DepartureTime { get; set; }

    [Required(ErrorMessage = "Часът на връщане е задължителен")]
    49 references
    public DateTime ReturnTime { get; set; }

    [Required(ErrorMessage = "Цената е задължителна")]
    [Range(1, int.MaxValue, ErrorMessage = "Цената трябва да е положително число")]

    28 references
    public int Price { get; set; }

    [Required(ErrorMessage = "Общият брой места е задължителен")]
    [Range(1, int.MaxValue, ErrorMessage = "Броят на местата трябва да е положително число")]

    37 references
    public int TotalSeats { get; set; }

    46 references
    public int FreeSeats { get; set; }

    [Required(ErrorMessage = "Моделът на автомобила е задължителен")]
    26 references
    public string CarModel { get; set; }

    [Required(ErrorMessage = "Регистрационният номер е задължителен")]
    26 references
    public string PlateNumber { get; set; }

    [Url(ErrorMessage = "Невалиден URL за изображение")]
    29 references
    public string? ImagePath { get; set; }

    95 references
    public TripStatus StatusTrip { get; set; }

    9 references
    public DateTime CreatedDate { get; set; } = DateTime.Now;

    5 references
    public DateTime tripSchedule { get; set; } = DateTime.Now;
    8 references
    public DateTime NextStart { get; set; } = DateTime.Now;

    22 references
    public bool IsRecurring { get; set; }

    13 references
    public String? RecurrenceInterval { get; set; } = "00:00:00";

    8 references
    public DateTime? NextRunDate { get; set; } = DateTime.Now;

    16 references
    public ICollection<TripParticipant>? TripParticipants { get; set; }
    15 references
    public ICollection<Request>? Requests { get; set; }
    8 references
    public ICollection<Rating>? Ratings { get; set; }
}
```

Фигура 12 Моделът “Пътешествие“

- Ключ
- Външен ключ към “Потребители“
- Шофьор
- Начална позиция

- Дестинация
- Час на заминаване
- Час на връщане
- Цена
- Общ брой места
- Свободни места
- Модел на автомобила
- Регистрационен
- Снимка
- Статус на пътуването
- Дата на създаване
- График на пътуване
- Следващо тръгване
- Следващо изпълнение
- Повтарящо се пътуване
- Интервал на повторение
- Участници в пътуването - Списък с всички потребители, записани за участие в пътуването
- Заявки– Списък със заявки от потребители за участие в пътуването
- Оценки – Списък с оценки, дадени за това пътуване

3.2.3 Заявка

Заявката се характеризира с:

- Ключ
- Външен ключ към “Пътешествие“
- Външен ключ към “Потребители“
- Дата на подаване
- Брой места
- Статус

```
29 references
public class Request
{
    [Key]
    18 references
    public int Id { get; set; }

    27 references
    public int TripId { get; set; }
    [ForeignKey(nameof(TripId))]
    24 references
    public Trip Trip { get; set; }

    31 references
    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    27 references
    public ApplicationUser User { get; set; }

    34 references
    public RequestStatus StatusRequest { get; set; }

    19 references
    public DateTime Date { get; set; }

    [Range(1, int.MaxValue, ErrorMessage = "Броят места трябва да бъде поне 1")]
    15 references
    public int NumberOfSeats { get; set; } = 1;
}
```

Фигура 13 Моделът “Заявка“

3.2.4 Участници в пътешесвие

Характеризира се с:

- Ключ
- Външен ключ към “Пътешествие“
- Външен ключ към “Потребители“
- Брой места

```
22 references
public class TripParticipant
{
    [Key]
    12 references
    public int Id { get; set; }

    27 references
    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    9 references
    public ApplicationUser User { get; set; }

    16 references
    public int TripId { get; set; }
    [ForeignKey(nameof(TripId))]
    32 references
    public Trip Trip { get; set; }

    [Range(1, int.MaxValue, ErrorMessage = "Броят места трябва да бъде поне 1")]
    6 references
    public int NumberOfSeats { get; set; } = 1;
}
```

Фигура 14 Моделът “ Участници в пътешесвие“

3.2.5 Заявка за шофьори

Характеризира се с:

- Ключ
- Външен ключ към “Потребители“
- Дата на подаване
- Статус

```
10 references
public class RequestDriver
{
    [Key]
    3 references
    public int Id { get; set; }

    // [Required]
    11 references
    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    8 references
    public ApplicationUser User { get; set; }

    12 references
    public RequestStatus StatusRequest { get; set; }

    5 references
    public DateTime Date { get; set; }
}
```

Фигура 15 Моделът “Заявка за шофьори“

3.2.6 Рейтинг

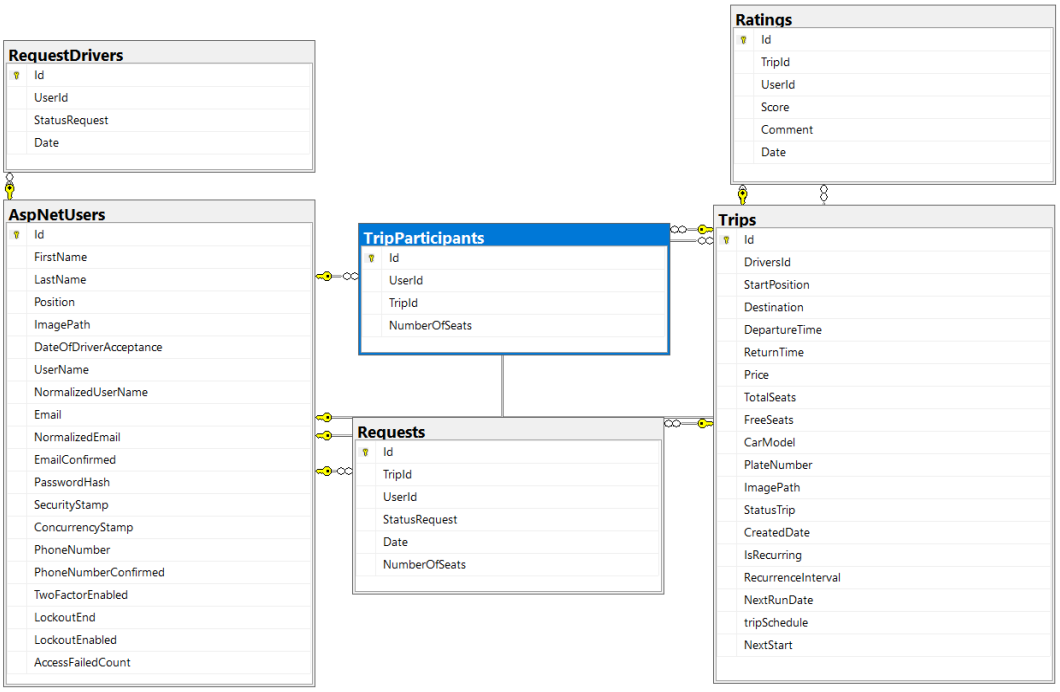
Характеризира се с:

- Ключ
- Външен ключ към “Потребители“
- Външен ключ към “Пътешествие“
- Дата на подаване
- Коментар
- Оценка

```
18 references
public class Rating
{
    [Key]
    9 references
    public int Id { get; set; }

    8 references
    public int TripId { get; set; }
    [ForeignKey(nameof(TripId))]
    12 references
    public Trip Trip { get; set; }

    8 references
    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    11 references
    public ApplicationUser User { get; set; }

    [Range(1,5, ErrorMessage =("Стойността трябва да е от 1 до 5"))]

    16 references
    public int Score { get; set; }

    15 references
    public string? Comment { get; set; }

    12 references
    public DateTime Date { get; set; } = DateTime.Now;

}
```

Фигура 16 Моделът "Рейтинг"

3.2.7 Общо свързване



Фигура 17 Общо свързване на таблиците

3.3 Изгледи в приложението „Споделено пътуване“

Изгледът (View) е компонентът от архитектурата на ASP.NET Core MVC, който отговаря за визуалното представяне на данните пред потребителя. Той съдържа HTML, в комбинация с Razor синтаксис, което позволява динамично вграждане на данни и логика за визуализация директно в HTML страниците.

В този проект изгледите служат за показване на информацията, обработена от контролера и модела, както и за събиране на потребителски вход чрез формуляри. Всеки изглед обикновено е свързан с конкретно действие от контролер и може да използва ViewModel-и за по-прецизно предаване на данни между слоевете.

ASP.NET Razor View Engine позволява лесно интегриране на C# логика в HTML шаблоните, като същевременно поддържа добра разделеност между бизнес логиката и потребителския интерфейс. Това улеснява поддръжката и развитието на приложението.

Чрез изгледите потребителят може да:

- Вижда списъци с пътувания, профили, заявки и оценки;
- Попълва форми за създаване или редакция на пътувания;
- Получава потвърждения за извършени действия;
- Визуализира данни според различни критерии и статуси.

3.3.1 Trip

3.3.1.1 Index

Предназначение: Показва списък с пътувания с възможности за филтриране и сортиране.

Основни компоненти:

- - Заглавие на страницата с информация за шофьора (ако е филтрирано по шофьор)
- - Бутон за създаване на ново пътуване (за шофьори и администратори)
- - Форма за търсене и филтриране с полета:
- - Начална позиция

- - Дестинация
- - Шофьор (падащо меню)
- - Падащо меню за сортиране по различни критерии
- - Списък с пътувания, показан чрез частичен изглед `_TripTablePartial`
- - Странициране чрез частичен изглед `_PaginationPartial`

```
@using TestProject.Extentions
@model PaginatedList<TestProject.Models.Trip>

@{
    ViewData["Title"] = "Пътешествия";

}

<div class="container">
    <div class="page-header">
        <h1>
            @if (!string.IsNullOrEmpty(ViewBag.DriverName))
            {
                <span>Пътешествия от @ViewBag.DriverName @* - @ViewBag.TripCount *@</span>
            }
            else
            {
                <span>Пътешествия @* - @ViewBag.TripCount *@</span>
            }
        </h1>

        @if (!string.IsNullOrEmpty(ViewBag.ReturnUrl) && Url.IsLocalUrl(ViewBag.ReturnUrl))
        {
            <a href="@ViewBag.ReturnUrl" class="btn btn-secondary">
                <i class="fas fa-arrow-left"></i> Върни се обратно
            </a>
        }
        else
        {
            @if (User.IsInRole("Driver"))
            {
                <a asp-action="Create" asp-route-returnUrl="@Context.Request.Path" class="create-trip-btn">
                    <i class="fas fa-plus-circle"></i> Създай ново пътешествие
                </a>
            }
            else
            {
                <a href="/Identity/Account/Manage" class="create-trip-btn">
                    <i class="fas fa-id-card"></i> Стани Шофьор
                </a>
            }
        }
    </div>

    <div class="search-form">
        <div class="search-form-header">
            <h2 class="search-form-title" style="margin-top: -20px;">Търсене и филтриране </h2>

            <div class="form-group" style="min-width: 200px; margin-left: 10px;">
                <form asp-action="Index" method="get" id="sortForm">
                    <input type="hidden" name="startPosition" value="@ViewBag.CurrentStartPosition" />
                    <input type="hidden" name="destination" value="@ViewBag.CurrentDestination" />
                    <input type="hidden" name="driverId" value="@ViewBag.CurrentDriverId" />
                    <input type="hidden" name="returnUrl" value="@ViewBag.ReturnUrl" />
                    <select name="sortOrder" asp-items="@(new SelectList(ViewBag.SortOptions as IEnumerable<SelectListItem>, "Value", "Text", ViewBag.CurrentSortOrder))"
                        onchange="this.form.submit()" class="form-control">
                    </select>
                </form>
            </div>
        </div>
```

Фигура 18 изглед "Index" част 1

```
        <form asp-action="Index" method="get" id="tripSearchForm">
            <input type="hidden" name="returnUrl" value="@ViewBag.ReturnUrl" />
            <input type="hidden" name="sortOrder" value="@ViewBag.CurrentSortOrder" />

            <div class="search-form-row" style="margin-bottom: -5px;">
                <div class="form-group">
                    <label>Стартираща позиция:</label>
                    <input type="text" name="startPosition" value="@ViewBag.CurrentStartPosition" class="form-control" placeholder="Въведете начална позиция" />
                </div>

                <div class="form-group">
                    <label>Дестинация:</label>
                    <input type="text" name="destination" value="@ViewBag.CurrentDestination" class="form-control" placeholder="Въведете дестинация" />
                </div>
                <div class="search-buttons" style="padding-bottom: 16px">
                    <button type="submit" class="btn btn-primary">
                        <i class="fas fa-search"></i> Потърси
                    </button>
                    <button type="button" class="btn btn-secondary" onclick="clearSearch()">
                        <i class="fas fa-times"></i> Изчисти
                    </button>
                </div>
            </div>
        </form>
    </div>

    @if (!Model.Any())
    {
        <div class="empty-state">
            <i class="fas fa-clipboard-check"></i>
            <p>Няма намерени пътешествия в този момент.</p>
        </div>
    }
    else
    {
        <partial name="_TripTablePartial" model="Model" />
        <partial name="_PaginationPartial" model="Model" />
    }
</div>

@section Scripts {
    <script>
        function clearSearch() {
            $('input[name=startPosition]').val('');
            $('input[name=destination]').val('');

            document.getElementById("tripSearchForm").submit();
        }
    </script>
}
```

Фигура 19 изглед "Index" част 2

3.3.1.2 Details

Предназначение: Показва подробна информация за конкретно пътуване.

Основни компоненти:

- - Заглавие и бутон за връщане назад
- - Изображение на пътуването и статус
- - Подробна информация, организирана в секции:
- - Основна информация (маршрут, време, цена)
- - Информация за шофьора
- - Информация за автомобила
- - Информация за местата
- - Информация за повторение (ако е приложимо)
- - Секция за действия:
- - Заявяване на място (за пътници)
- - Управление на заявки
- - Редактиране и изтриване (за шофьора)

Функционалност:

- - Показва всички детайли за пътуването
- - Позволява на пътниците да заявят място с избор на брой места
- - Показва статуса на заявката на текущия потребител
- - Позволява на шофьора да редактира или изтрие пътуването

```
<div class="trip-details-info">
    <div class="trip-details-header">
        <div class="trip-details-route">
            <h2>@Model.StartPosition → @Model.Destination</h2>
            <div class="trip-details-times">
                <div class="trip-time">
                    <i class="fas fa-hourglass-start"></i> Тръгване: @Model.DepartureTime.ToString("dd-MM-yyyy HH:mm")
                </div>
                <div class="trip-time">
                    <i class="fas fa-hourglass-end"></i> Връщане: @Model.ReturnTime.ToString("dd-MM-yyyy HH:mm")
                </div>
            </div>
        </div>
        <div class="trip-details-price">
            <span>@Model.Price лв</span>
        </div>
    </div>

    <div class="trip-details-sections">
        <div class="trip-details-section">
            <h3 class="section-title"><i class="fas fa-user"></i> Информация за шофьора</h3>
            <div class="details-grid">
                <div class="details-item">
                    <span class="details-label">Име</span>
                    <span class="details-value">@Model.DriverName</span>
                </div>
                <div class="details-item">
                    <span class="details-label">Телефон</span>
                    <span class="details-value">@Model.DriverPhoneNumber</span>
                </div>
                <div class="details-item">
                    <span class="details-label">Шофьор от</span>
                    <span class="details-value">@(Model.Driver?.DateOfDriverAcceptance?.ToString("dd-MM-yyyy") ?? "Вече не е шофьор")</span>
                </div>
            </div>
        </div>
```

Фигура 20 Част от изгледа "Details"

3.3.1.3 Create

Предназначение: Предоставя формуляр за създаване на ново пътуване.

Основни компоненти:

- - Заглавие и бутон за връщане назад
- - Формуляр с три основни секции:
- - Детайли за пътуването (начална позиция, дестинация, време, цена, места)
- - Детайли за автомобила (модел, регистрационен номер, изображение)
- - Настройки за повторение (интервал в минути, часове, дни)
- - Предварителен преглед на пътуването в реално време

Функционалност:

- - Валидация на въведените данни
- - Качване и предварителен преглед на изображение
- - Настройка на повтарящи се пътувания
- - Предварителен преглед на пътуването в реално време
- - Проверка за припокриващи се пътувания

```
<div class="form-section">
    <h2 class="section-title">
        <i class="fas fa-car"></i> Детайли за автомобила
    </h2>

    <div class="form-row">
        <div class="form-group">
            <label asp-for="CarModel" class="control-label">
                <i class="fas fa-car-side"></i> Модел на автомобила
            </label>
            <input asp-for="CarModel" class="form-control" placeholder="Въведете модел" />
            <span asp-validation-for="CarModel" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="PlateNumber" class="control-label">
                <i class="fas fa-id-card"></i> Регистрационен номер
            </label>
            <input asp-for="PlateNumber" class="form-control" placeholder="Въведете рег. номер" />
            <span asp-validation-for="PlateNumber" class="text-danger"></span>
        </div>
    </div>
```

Фигура 21 Част от изгледа "Create"

3.3.1.4 Edit

Предназначение: Предоставя формуляр за редактиране на съществуващо пътуване.

Основни компоненти:

- - Подобен на Create View, но с предварително попълнени данни
- - Допълнителна секция за статус на пътуването
- - Показва текущото изображение с възможност за промяна
- - Предупреждение при намаляване на броя места
- Функционалност:
- - Валидация на въведените данни
- - Качване и предварителен преглед на ново изображение
- - Настройка на повтарящи се пътувания
- - Предварителен преглед на пътуването в реално време
- - Проверка за припокриващи се пътувания

- - Предупреждение при промени, които биха засегнали съществуващи заявки

```
<div class="form-group">
    <label asp-for="ImageFile" class="control-label">
        <i class="fas fa-image"></i> Изображение
    </label>
    <div class="custom-file-upload">
        <input asp-for="ImageFile" type="file" class="form-control" accept="image/*" id="imageUpload" />
        <label for="imageUpload" class="file-upload-label">
            <i class="fas fa-cloud-upload-alt"></i> Изберете файл
        </label>
        <span id="file-chosen">Не е избран файл</span>
    </div>
    <span asp-validation-for="ImageFile" class="text-danger"></span>

    @if (!string.IsNullOrEmpty(Model.ImagePath))
    {
        <div class="current-image-container">
            <p class="current-image-label">Текущо изображение:</p>
            <img src="@Model.ImagePath" alt="Current Trip Image" class="current-image" />
        </div>
    }

    <p>Ново изображение:</p>
    <div id="image-preview-container" class="image-preview-container" style="display: none;">
        <img id="image-preview" src="#" alt="Preview" class="image-preview" />
        <button type="button" id="remove-image" class="remove-image-btn">
            <i class="fas fa-times"></i>
        </button>
    </div>
```

Фигура 22 Част от изгледа "Edit"

3.3.2 TripViewModel

Моделът на изгледа (ViewModel) представлява специален клас, използван за предаване на данни между контролера и изгледа. За разлика от основния модел (Model), който е обвързан директно с базата данни, ViewModel се използва за представяне на специфични за изгледа нужди – комбиниране на данни от различни източници, форматиране на стойности и предоставяне на допълнителна логика, нужна за визуализация или потребителски вход.

ViewModel-ите изпълняват ролята на посредник между слоевете „контролер“ и „изглед“, като помагат да се постигне:

- По-добра организация на кода;
- По-голяма сигурност чрез скриване на чувствителни данни;
- По-гъвкаво представяне на данни, различно от структурата на базата;
- Валидиране на потребителски вход чрез атрибути за валидация (Data Annotations).
- В този проект ViewModel-ите се използват за:
- Създаване и редакция на пътувания (напр. с полета като дати, начален/краен адрес, цена, места и т.н.);

- Филтриране и търсене на пътувания;
- Показване на заявки, оценки и участници;
- Обединяване на данни от няколко свързани модела за нуждите на конкретен изглед.

Този подход осигурява по-ясна логика, по-добра структура на кода и улеснява поддръжката и разширяването на приложението.

```
12 references
public class TripViewModel : Trip
{
    [NotMapped]
   // [Required(ErrorMessage = "Please upload an image.")]
    14 references
    public IFormFile? ImageFile { get; set; }
    4 references
    public string? DriverName { get; set; }

    2 references
    public string? DriverPhoneNumber { get; set; }

    5 references
    public int? RecurrenceIntervalMinutes { get; set; } = 0;
    5 references
    public int? RecurrenceIntervalDays { get; set; } = 0;
    5 references
    public int? RecurrenceIntervalHours { get; set; } = 0;

}
```

Фигура 23 Моделът на изгледа “Пътешествие“

3.4 Контролери в приложението „Споделено пътуване“

Контролерите са ключов компонент от архитектурата на ASP.NET Core MVC и играят ролята на посредник между потребителския интерфейс (изгледите) и бизнес логиката (моделите и услугите). Те получават HTTP заявки от клиента, обработват ги, извикват нужната логика и връщат подходящ отговор – най-често изглед или JSON данни.

Контролерите:

- Получават входни данни от потребителя чрез заявки;
- Валидират тези данни с помощта на модели и анотации;
- Извикват бизнес логиката (услуги/сервиси, repository и др.);
- Манипулират или извличат информация от базата данни чрез моделите;
- Връщат изгледи с ViewModel-и или API отговори.

В този проект, контролерите са разделени според функционалността на приложението.

Един от основните контролери е “TripsController“ – управлява създаване, редактиране, изтриване, заявяване и визуализиране на пътувания;

3.4.1 TripsController

В ASP.NET MVC, HTTP методите GET и POST играят ключова роля за начина, по който потребителят взаимодейства с приложението. Методите от тип GET се използват за извличане и показване на информация – като например формуляри за попълване или списъци с данни. Те не променят състоянието на сървъра. Обратно, методите от тип POST се използват за обработка на въведени от потребителя данни – например за създаване, редактиране или изтриване на записи в базата данни. В тази секция ще се съсредоточим върху POST логиката на основните методи, която съдържа реалната обработка и бизнес логика на приложението.

3.4.1.1 Index

Методът Index обработва параметрите, подадени от потребителя за сортиране и филтриране на списък с пътувания. Приемат се стойности като начална позиция, дестинация, ID на шофьор и избран критерий за сортиране. След това се

прилага филтрация и сортиране върху заявката към базата данни. Резултатът се разделя на страници чрез странициране, което улеснява навигацията в по-големи списъци. След обработката се връща изглед със списъка от пътувания, отговарящи на зададените условия.

```
ViewBag.Drivers = new SelectList(drivers, "Value", "Text", driverId);

IQueryable<Trip> trips = _context.Trips.Include(t => t.Driver);

if (!string.IsNullOrEmpty(driverId))
{
    trips = trips.Where(t => t.DriversId == driverId);

    var driver = await _context.ApplicationUsers.FindAsync(driverId);

    if (driver != null)
    {
        ViewBag.DriverName = $"{driver.FirstName} {driver.LastName}";
    }
}

var tripsList = await trips.ToListAsync();

if (!string.IsNullOrEmpty(startPosition))
{
    tripsList = tripsList.Where(t => t.StartPosition.Contains(startPosition, StringComparison.OrdinalIgnoreCase)).ToList();
}

if (!string.IsNullOrEmpty(destination))
{
    tripsList = tripsList.Where(t => t.Destination.Contains(destination, StringComparison.OrdinalIgnoreCase)).ToList();
}

IOrderedEnumerable<Trip> orderedTrips = tripsList.OrderBy(t => t.StatusTrip != TripStatus.Upcoming);
orderedTrips = orderedTrips.ThenBy(t => t.StatusTrip);
```

Фигура 24 Част от методът "Index"

3.4.1.2 Details

Методът Details зарежда подробна информация за конкретно пътуване чрез неговото ID. Освен основните данни за самото пътуване, се зареждат и свързани

```
0 references
public async Task<IActionResult> Details(int? id, string? returnUrl)
{
    if (id == null)
    {
        return NotFound();
    }

    var trip = await _context.Trips
        .Include(t => t.Ratings!)
            .ThenInclude(r => r.User)
        .Include(t => t.Driver)
        .Include(t => t.TripParticipants)
        .Include(t => t.Requests!)
            .ThenInclude(r => r.User)
        .FirstOrDefaultAsync(m => m.Id == id);

    if (trip == null)
    {
        return NotFound();
    }

    var tripViewModel = new TripViewModel
    {
        Id = trip.Id,
        Driver = trip.Driver,
        DriversId = trip.DriversId,
        DriverName = trip.Driver.UserName,
        DriverPhoneNumber = trip.Driver.PhoneNumber,
        StartPosition = trip.StartPosition,
        Destination = trip.Destination,
        DepartureTime = trip.DepartureTime,
        ReturnTime = trip.ReturnTime,
        Price = trip.Price,
        TotalSeats = trip.TotalSeats,
        FreeSeats = trip.FreeSeats,
        CarModel = trip.CarModel,
        PlateNumber = trip.PlateNumber,
        ImagePath = trip.ImagePath,
        StatusTrip = trip.StatusTrip,
        Ratings = trip.Ratings,
        TripParticipants = trip.TripParticipants,
        IsRecurring = trip.IsRecurring,
        RecurrenceInterval = trip.RecurrenceInterval,
        NextRunDate = trip.NextRunDate,
        Requests = trip.Requests
    };
```

Фигура 25 Методът "Details"

обекти като рейтинги, шофьор, участници и заявки. След това информацията се преобразува в TripViewModel, за да бъде удобно подадена на изгледа. Така потребителят може да прегледа цялата налична информация за избраното пътуване на едно място.

3.4.1.3 Create

Методът Create отговаря за създаването на ново пътуване въз основа на данни, въведени от потребителя. Извършва се цялостна проверка (валидация) на формуляра, включително и на каченото изображение. Ако изображението е налично, то се записва в локалната файлова система, а пътят към него се съхранява в базата. След това се създава нов обект от тип Trip, който се запазва чрез Entity Framework. Ако потребителят е избрал опция за повтаряемост, се генерират допълнителни копия на пътуването в бъдещи дати. След успешно създаване потребителят се пренасочва към страницата със списък на пътуванията или друга, зададена чрез returnUrl.

```
// For more details, see http://go.microsoft.com/fwlink/?LinkID=317598.
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin,Driver")]
0 references
public async Task<IActionResult> Create(TripViewModel tripViewModel, string? returnUrl)
{
    tripViewModel.DriversId = User.Id();
    tripViewModel.StatusTrip = TripStatus.Upcoming;

    var recurrenceInterval = TimeSpan.FromDays(tripViewModel.RecurrenceIntervalDays ?? 0)
      .Add(TimeSpan.FromHours(tripViewModel.RecurrenceIntervalHours ?? 0))
      .Add(TimeSpan.FromMinutes(tripViewModel.RecurrenceIntervalMinutes ?? 0));

    if (tripViewModel.IsRecurring && recurrenceInterval == TimeSpan.Zero)
    {
        ModelState.AddModelError("", "Интервалът трябва да е по-голям от нула.");
    }

    if (ModelState.IsValid)
    {
        if (tripViewModel.ImageFile != null && tripViewModel.ImageFile.Length > 0)
        {
            string uploadsFolder = Path.Combine("wwwroot", "images", "trips");

            if (!Directory.Exists(uploadsFolder))
            {
                Directory.CreateDirectory(uploadsFolder);
            }

            string uniqueFileName = Guid.NewGuid().ToString() + Path.GetExtension(tripViewModel.ImageFile.FileName);
            string filePath = Path.Combine(uploadsFolder, uniqueFileName);

            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                await tripViewModel.ImageFile.CopyToAsync(stream);
            }

            tripViewModel.ImagePath = $"/images/trips/{uniqueFileName}";
        }
```

Фигура 26 Част от методът "Create"

3.4.1.4 Edit

Методът Edit обработва заявката за редакция на съществуващо пътуване. Валидират се подадените стойности, включително датите и местата. Ако

потребителят е качил ново изображение, старото се премахва и новото се запазва на негово място. След това съответният запис в базата се актуализира с новите стойности. Ако броят на свободните места е променен, се извършва допълнителна логика за управление на вече одобрени заявки и промяна в статуса на пътуването. След приключване на редакцията потребителят се пренасочва към оригиналната страница, от която е започнал.

```
    trip.CarModel = tripViewModel.CarModel;
    trip.PlateNumber = tripViewModel.PlateNumber;
    trip.IsRecurring = tripViewModel.IsRecurring;
    trip.RecurrenceInterval = recurrenceInterval.ToString();
    trip.NextRunDate = tripViewModel.DepartureTime.Add(recurrenceInterval);
    trip.NextStart = tripViewModel.DepartureTime.Add(recurrenceInterval);


    if (trip.FreeSeats == 0 && trip.StatusTrip != tripViewModel.StatusTrip && tripViewModel.StatusTrip != TripStatus.Upcoming)
    {
        trip.StatusTrip = tripViewModel.StatusTrip;
    }
    else if (trip.FreeSeats == 0 && trip.StatusTrip == tripViewModel.StatusTrip && trip.StatusTrip != TripStatus.Upcoming)
    {

    }
    else if (trip.FreeSeats == 0)
    {
        trip.StatusTrip = TripStatus.Booked;
    }
    else if (trip.FreeSeats > 0 && trip.StatusTrip == tripViewModel.StatusTrip && trip.StatusTrip == TripStatus.Booked)
    {
        trip.StatusTrip = TripStatus.Upcoming;
    }
    else
    {
        trip.StatusTrip = tripViewModel.StatusTrip;
    }

        _context.Update(trip);
    await _context.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException)
{
    if (!TripExists(id))
    {
        return NotFound();
    }
    else
    {
        throw;
    }
}
```

Фигура 27 Част от методът "Edit"

3.4.1.5 Delete

Методът DeleteConfirmed (Delete) обработва действителното изтриване на дадено пътуване. След като бъде намерено по ID, се премахва и свързаното с него изображение от файловата система, ако съществува такова. След това записът се изтрива от базата данни чрез Entity Framework. Накрая потребителят се пренасочва обратно към изгледа със списък на пътуванията или към друга предефинирана страница.

```
// POST: Trips/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> DeleteConfirmed(int id, string? returnUrl)
{
    var trip = await _context.Trips.FindAsync(id);
    if (trip != null)
    {
        if (!string.IsNullOrEmpty(trip.ImagePath))
        {
            string filePath = Path.Combine("wwwroot", trip.ImagePath);
            if (System.IO.File.Exists(filePath))
            {
                System.IO.File.Delete(filePath);
            }
        }
        _context.Trips.Remove(trip);
    }
    await _context.SaveChangesAsync();
    if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }

    return RedirectToAction("MyDriverTrips", "DriverTrips");

}
```

Фигура 28 Методът "Delete"

3.4.1.6 GetUserOverlappingTrips

Приватният метод GetUserOverlappingTrips подпомага функционалността за създаване и редакция, като проверява дали потребителят има други пътувания, които времево се припокриват с текущото. Методът проверява както участието на потребителя като шофьор, така и като пътник, и изключва текущото пътуване от резултата, когато е необходимо. Връща списък с конфликти, който може да се използва за предупреждение към потребителя или предотвратяване на създаването.

```
2 references
private async Task<List<object>> GetUserOverlappingTrips(string userId, int? excludeTripId = null)
{
    var driverTripsQuery = _context.Trips
        .Where(t => t.DriversId == userId &&
               (t.StatusTrip == TripStatus.Upcoming || t.StatusTrip == TripStatus.Ongoing));

    if (excludeTripId.HasValue)
    {
        driverTripsQuery = driverTripsQuery.Where(t => t.Id != excludeTripId.Value);
    }

    var driverTrips = await driverTripsQuery
        .Select(t => new {
            id = t.Id,
            start = t.DepartureTime,
            end = t.ReturnTime,
            startPosition = t.StartPosition,
            destination = t.Destination
        })
        .ToListAsync();

    var passengerTripsQuery = _context.TripParticipants
        .Where(tp => tp.UserId == userId)
        .Include(tp => tp.Trip)
        .Where(tp => tp.Trip.StatusTrip == TripStatus.Upcoming || tp.Trip.StatusTrip == TripStatus.Ongoing);

    if (excludeTripId.HasValue)
    {
        passengerTripsQuery = passengerTripsQuery.Where(tp => tp.Trip.Id != excludeTripId.Value);
    }

    var passengerTrips = await passengerTripsQuery
        .Select(tp => new {
            id = tp.Trip.Id,
            start = tp.Trip.DepartureTime,
            end = tp.Trip.ReturnTime,
            startPosition = tp.Trip.StartPosition,
            destination = tp.Trip.Destination
        })
        .ToListAsync();

    var userTrips = driverTrips.Concat(passengerTrips).Cast<object>().ToList();
    return userTrips;
}
```

Фигура 29 Методът "GetUserOverlappingTrips"

3.4.1.7 TripExists

Проверява дали пътешествието съществува

```
1 reference
private bool TripExists(int id)
{
    return _context.Trips.Any(e => e.Id == id);
}
```

Фигура 30 Методът "TripExists"

Заключение

Приложението „Споделени пътувания“, разработено на C# с използване на ASP.NET MVC, представлява съвременен и практичен софтуерен продукт, който има за цел да улесни хората в намирането и организирането на съвместни пътувания между различни населени места. Идеята на системата е да събере на едно място шофьори, които разполагат със свободни места в автомобила си, и пътници, търсещи удобен, икономичен и екологичен начин за придвижване. Приложението осигурява достъпна и удобна платформа за комуникация между двете страни, като по този начин насърчава споделянето на ресурси и намалява трафика и вредните емисии.

Системата е изградена върху архитектурния модел Model-View-Controller (MVC), което осигурява ясна структура на кода, улеснява неговата поддръжка и дава възможност за лесно бъдещо надграждане. Моделите в приложението съдържат информация за потребителите, пътуванията, заявките за участие, маршрутите, датите и оценките на пътуванията. Контролерите обработват потребителските действия и управляват логиката на приложението, като създаване на ново пътуване, подаване на заявка за участие, одобрение или отхвърляне на пътници и други. Изгледите, създадени чрез Razor синтаксис, предоставят интуитивен и лесен за използване потребителски интерфейс, който позволява на потребителите да навигират и използват приложението без затруднения.

Приложението е разработено с помощта на .NET фреймуърк и C#, което гарантира надеждност, бързодействие и сигурност. Използвана е базата данни MS SQL Server, която се управлява чрез Entity Framework, осигурявайки лесна и ефективна работа с данните. За по-добро потребителско изживяване са използвани HTML, CSS (Bootstrap) и JavaScript, които допринасят за по-модерен и отзивчив дизайн.

Основните възможности на приложението включват създаване и редактиране на пътувания от страна на шофьори, търсене и филтриране на пътувания по дата, маршрут и брой свободни места, кандидатстване на пътници за участие в дадено пътуване, одобрение или отказ от страна на шофьора, оставяне на обратна връзка след приключване на пътуването, както и възможност за повтарящи се пътувания – например всяка седмица по определен маршрут.

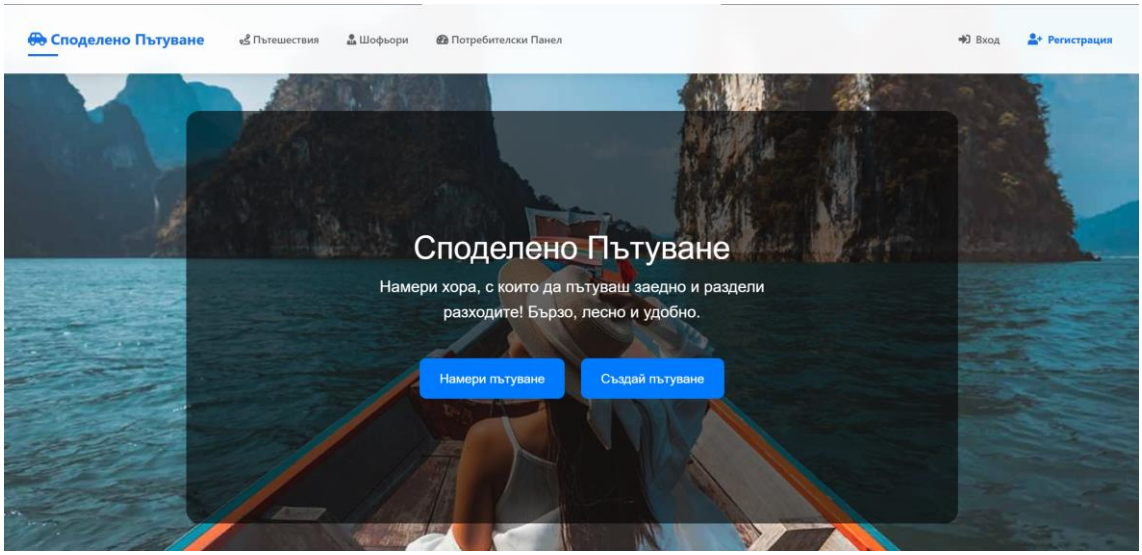
Приложението има потенциал за допълнително развитие. Възможни подобрения включват качване на проекта и базата му данни в облачно пространство, което ще осигури по-лесен достъп и скалируемост; създаване на мобилна версия или прогресивна уеб апликация (PWA), която ще улесни потребителите в движение; интеграция с картови и маршрутни услуги като Google Maps за по-добра визуализация на пътуванията; въвеждане на система за известия и съобщения между потребителите за по-добра комуникация; както и възможност за сътрудничество с реални транспортни фирми за разширяване на обхвата и възможностите на услугата.

Проектът „Споделени пътувания“ демонстрира силен практически подход към съвременен проблем и предлага дигитално решение, което е както удобно, така и от полза за околната среда и обществото. Разработката следва добрите практики в софтуерното инженерство и предоставя стабилна основа за бъдещо надграждане и комерсиализация.

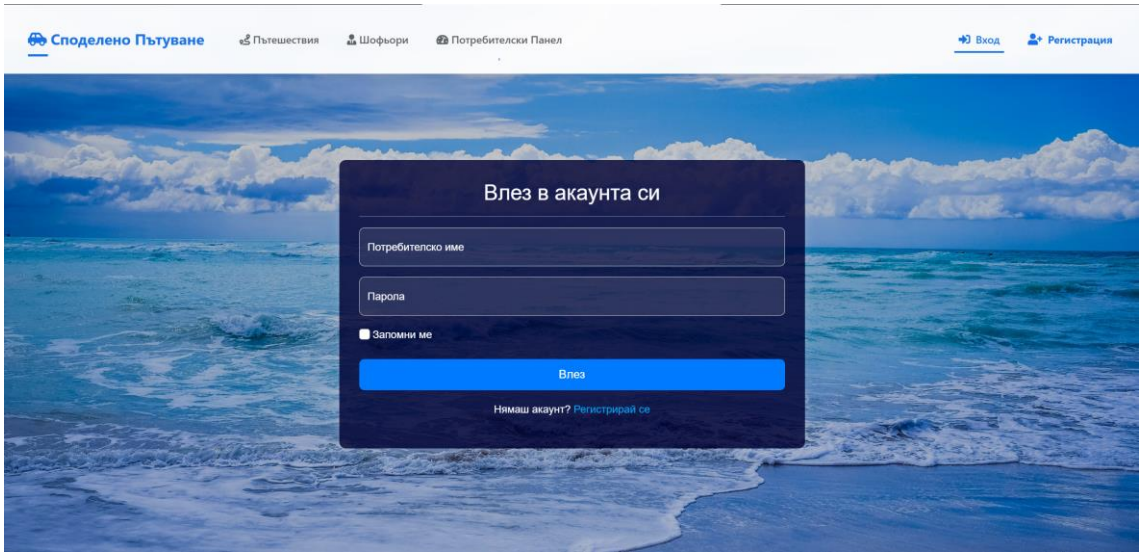
Използвана литература

1. Джон Дъкет, „HTML & CSS: Design and Build Web Sites“
2. Петров П., „Създаване на уеб приложение с ASP.NET MVC“, Варна, 2020
3. Светлин Наков, „Основи на уеб програмирането със C# и ASP.NET“
4. https://getbootstrap.com/
5. http://www.w3schools.com/
6. https://learn.microsoft.com/en-us/aspnet/core/
7. https://www.microsoft.com/en-us/sql-server

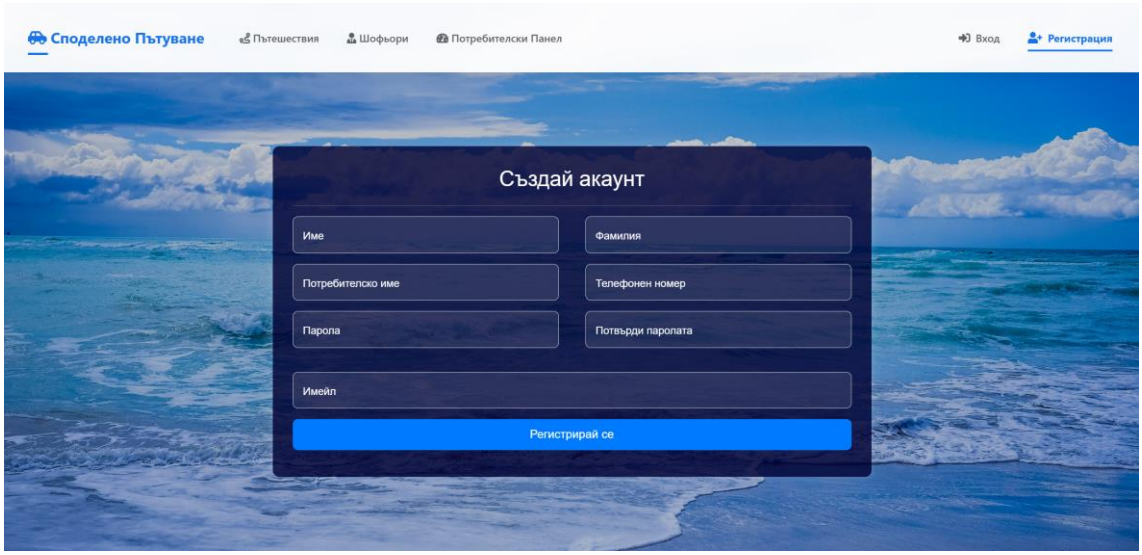
Приложението „Споделено пътуване“



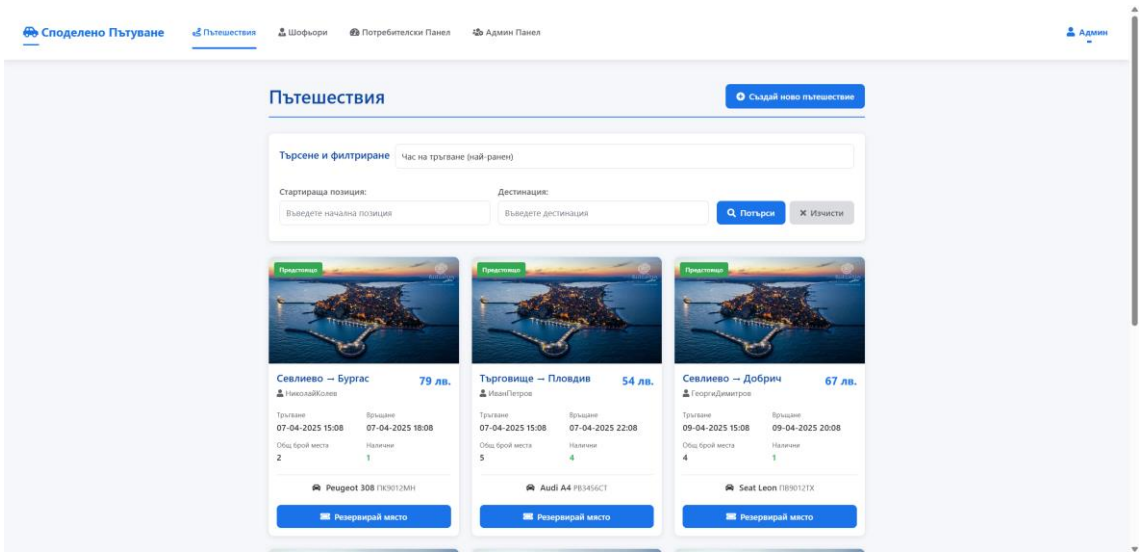
Фигура 31



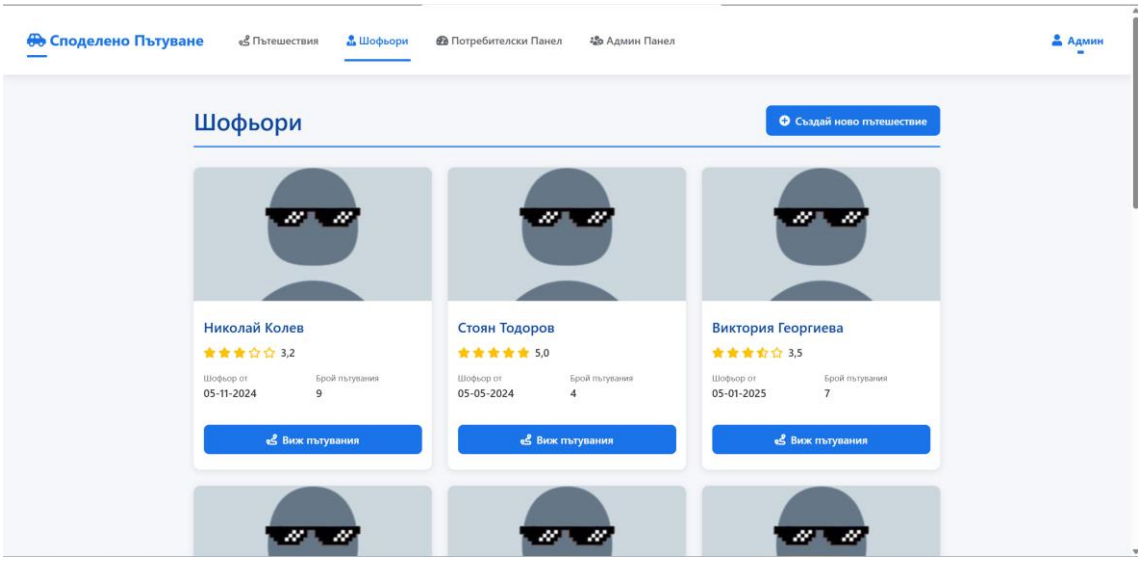
Фигура 32



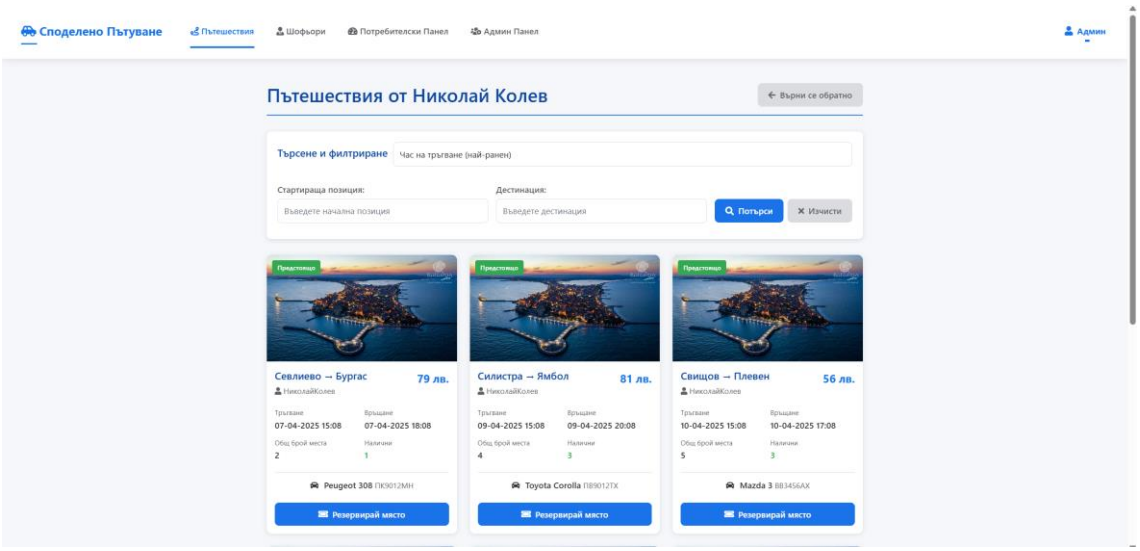
Фигура 33



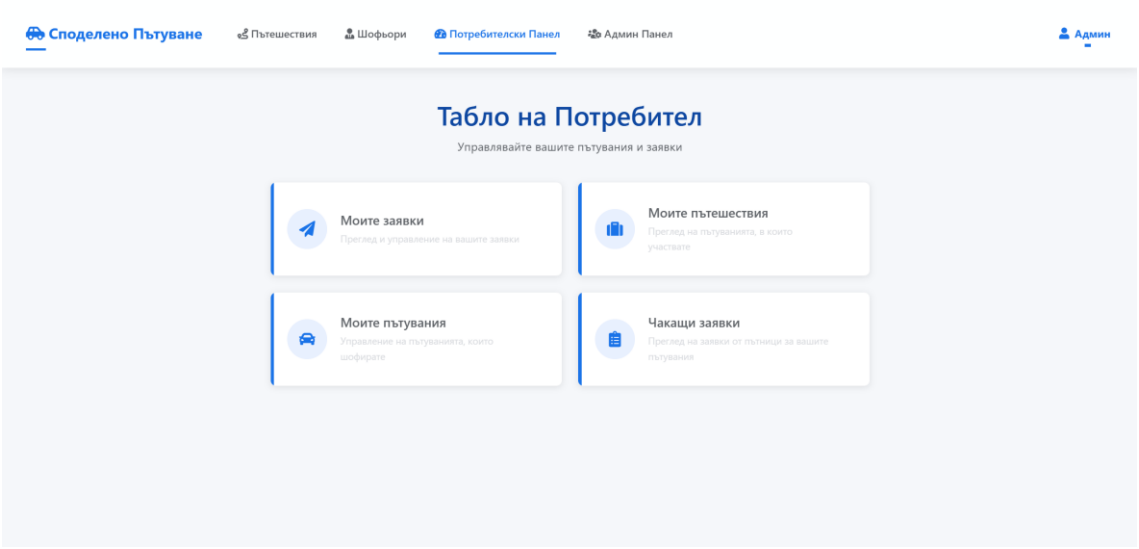
Фигура 34



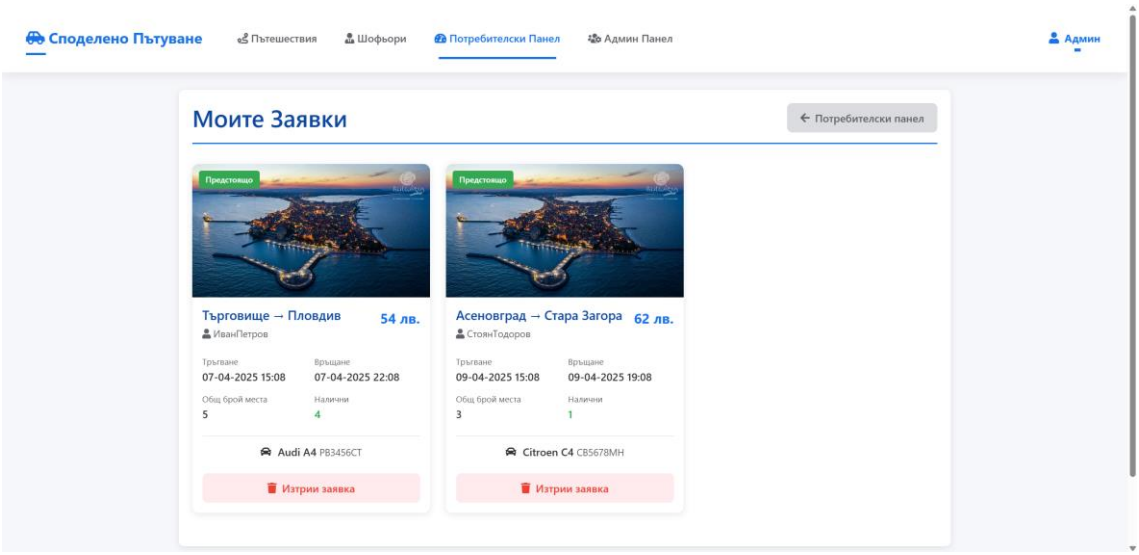
Фигура 35



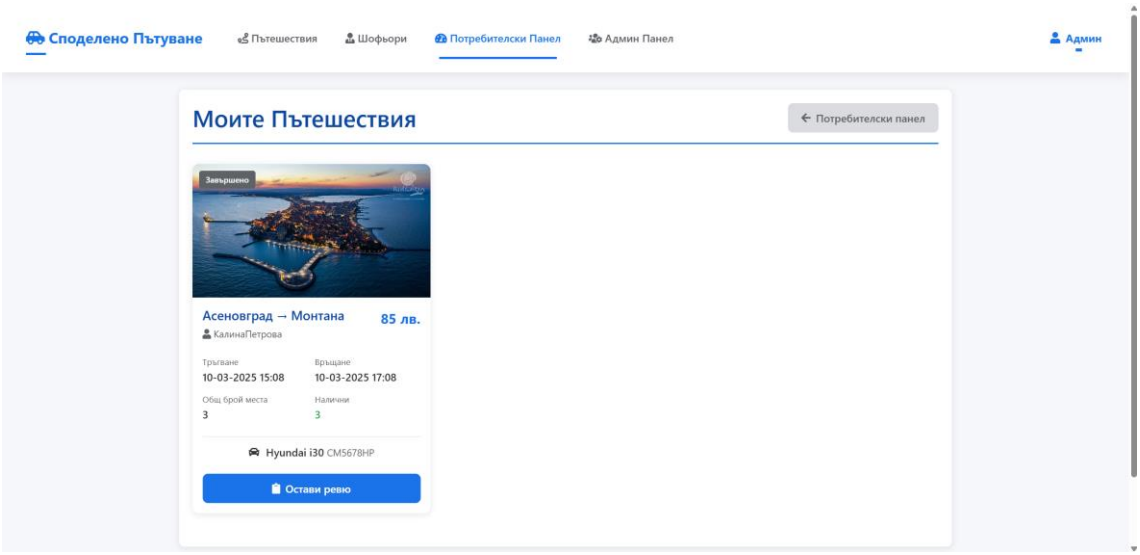
Фигура 36



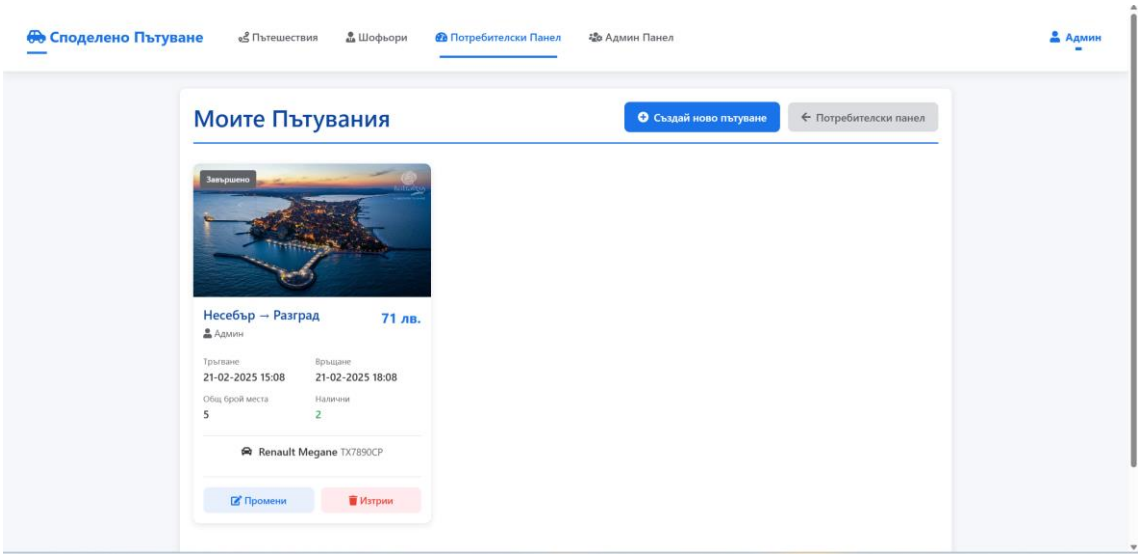
Фигура 37



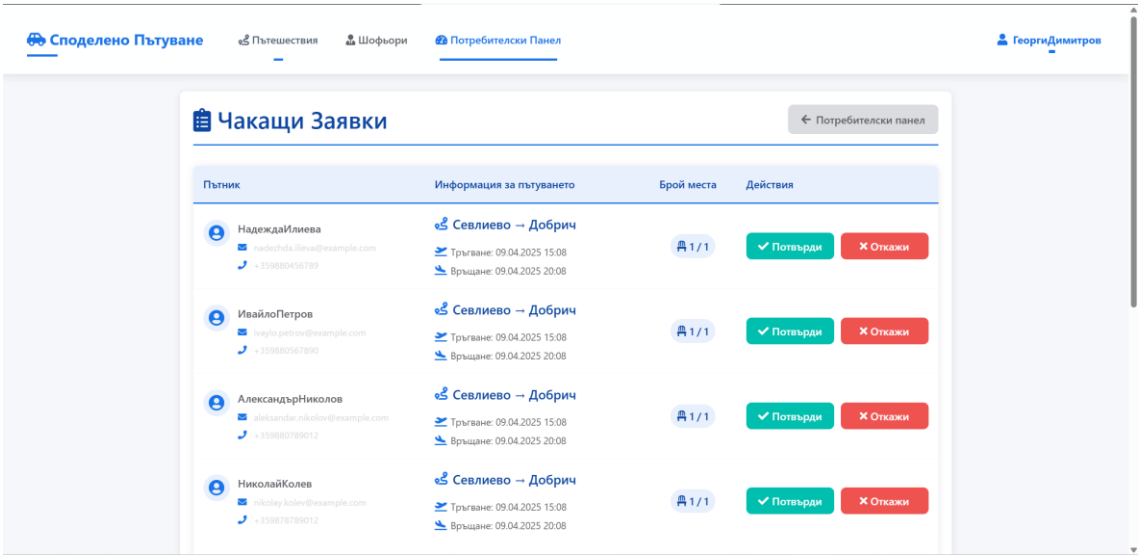
Фигура 38



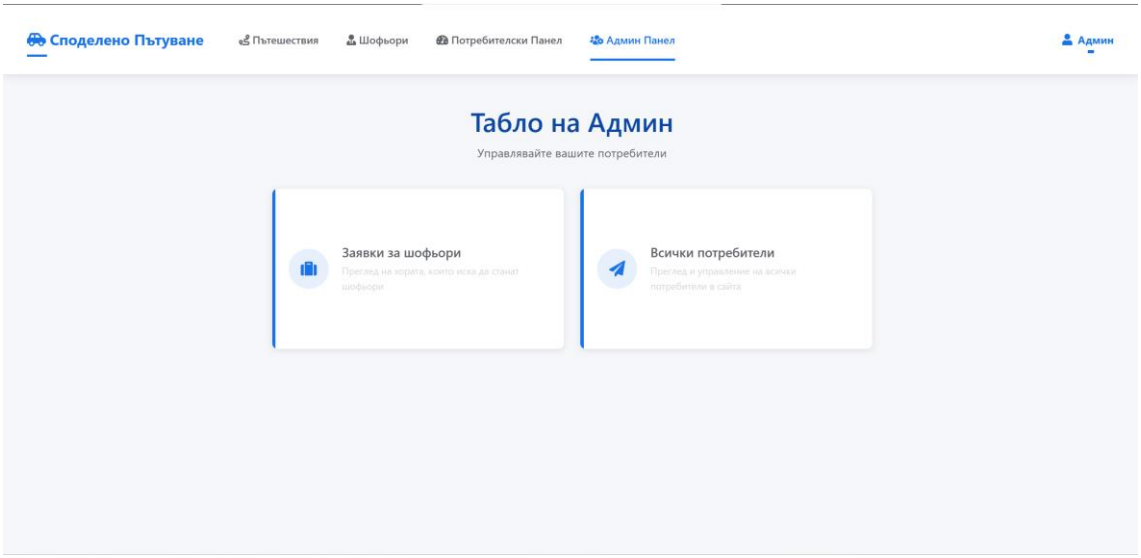
Фигура 39



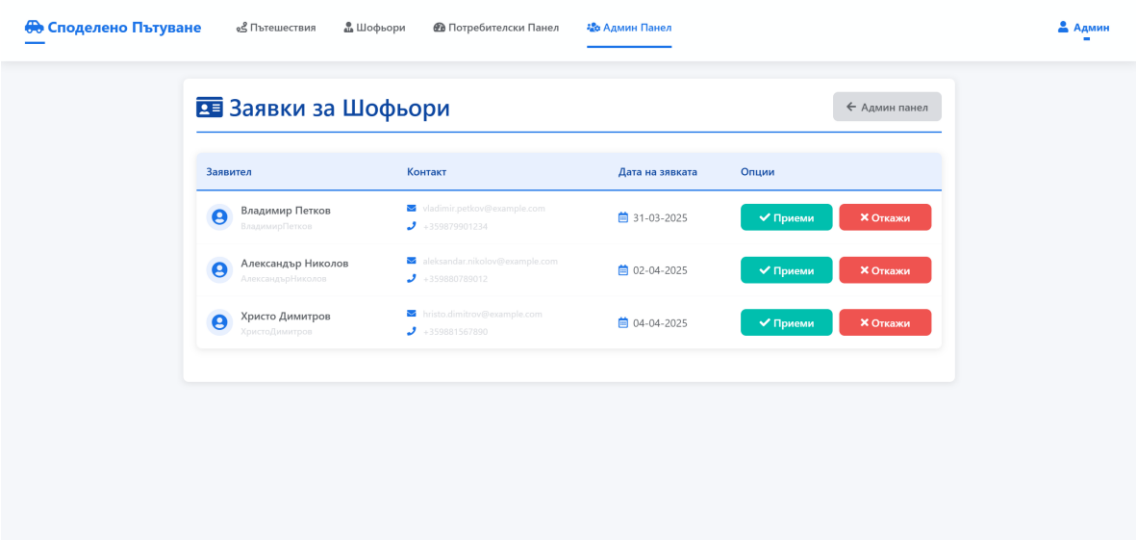
Фигура 40



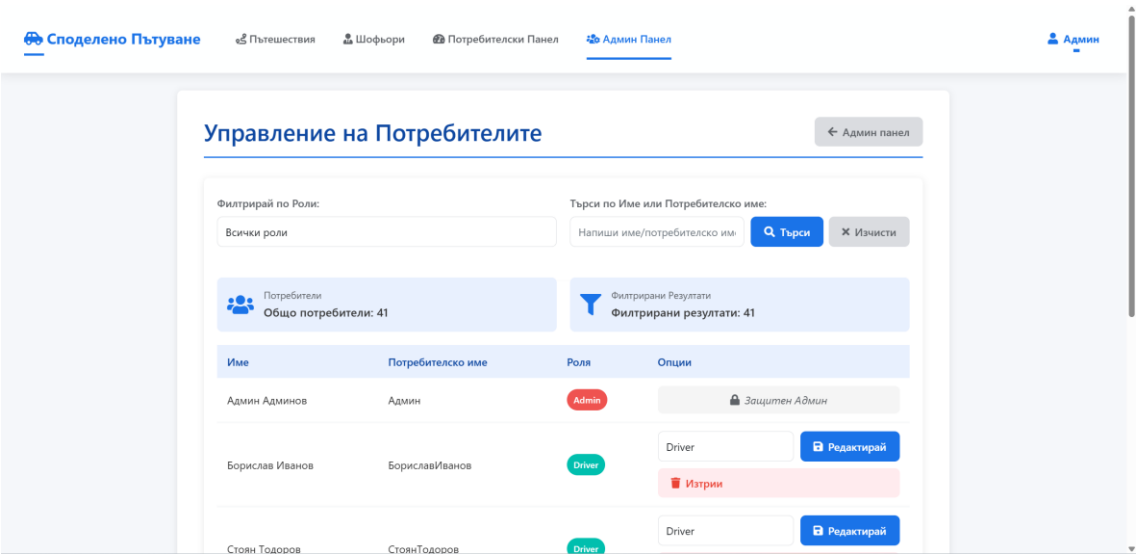
Фигура 41



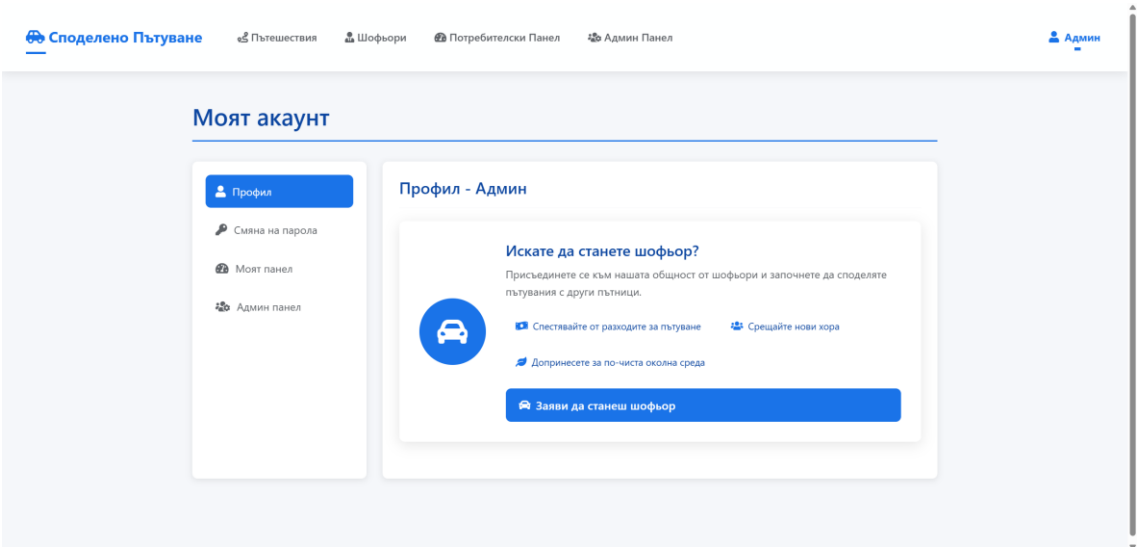
Фигура 42



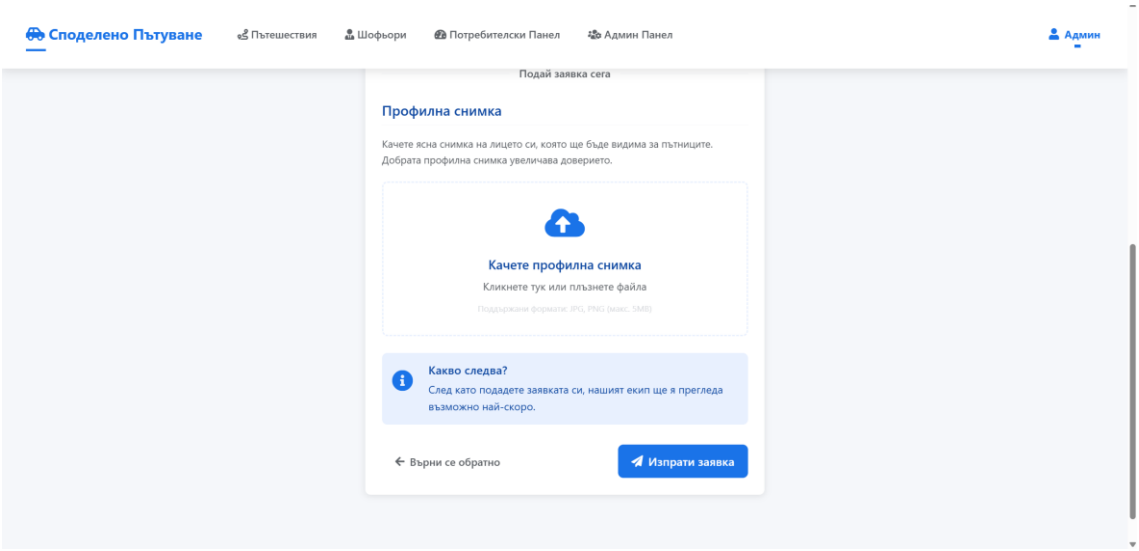
Фигура 43



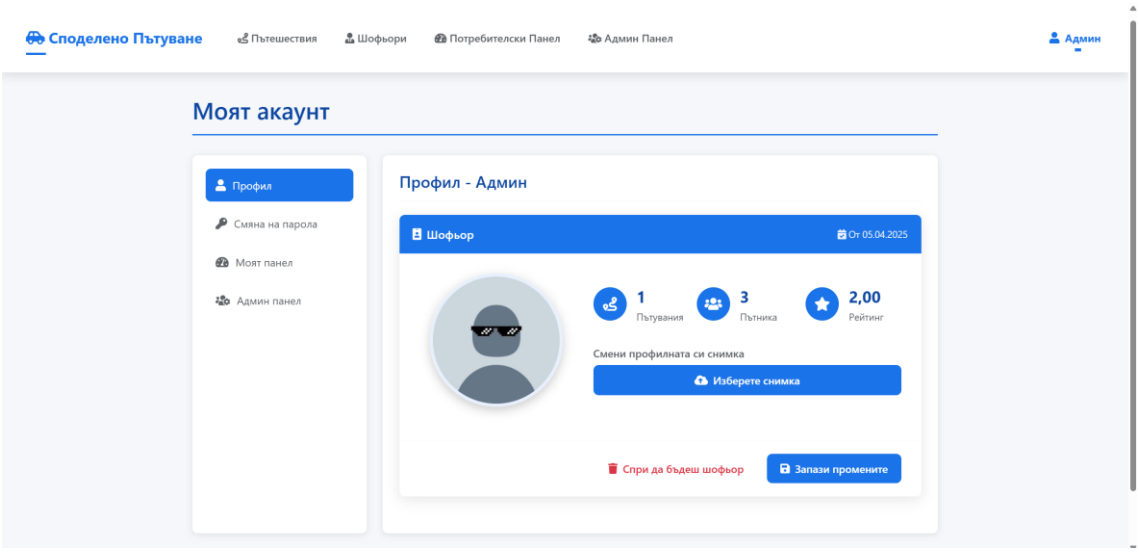
Фигура 44



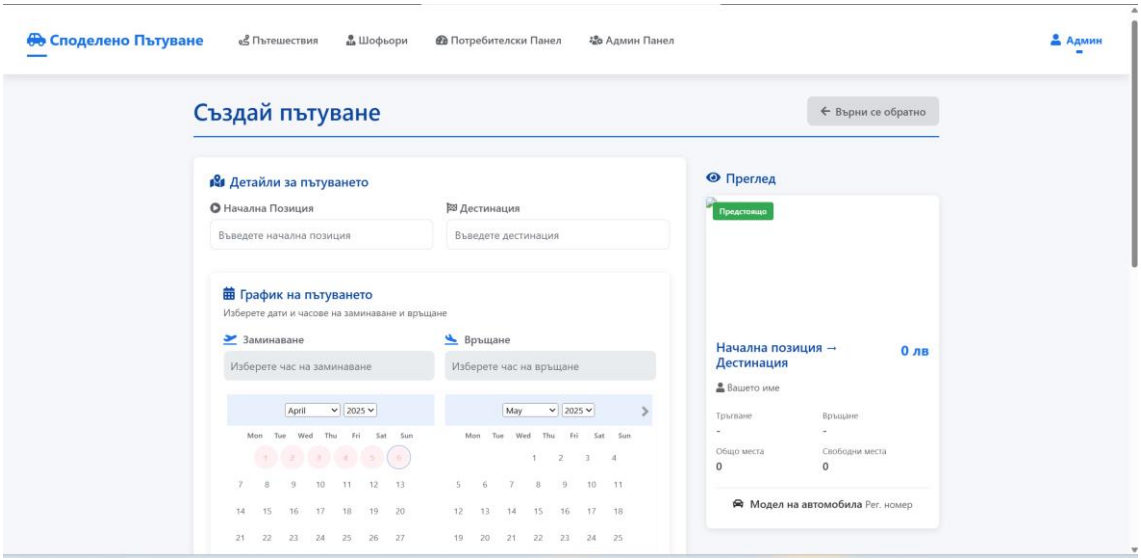
Фигура 45



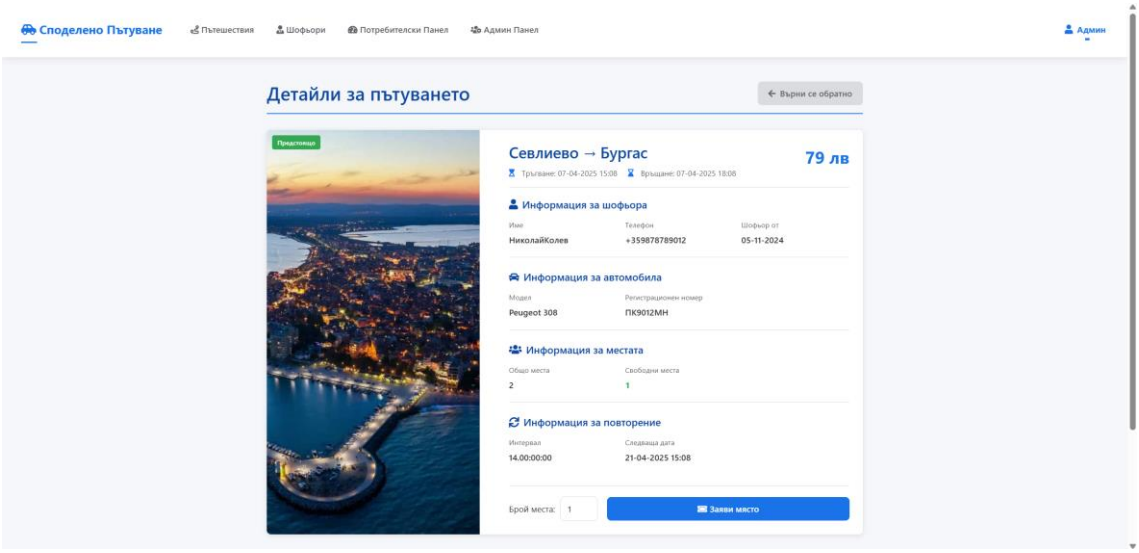
Фигура 46



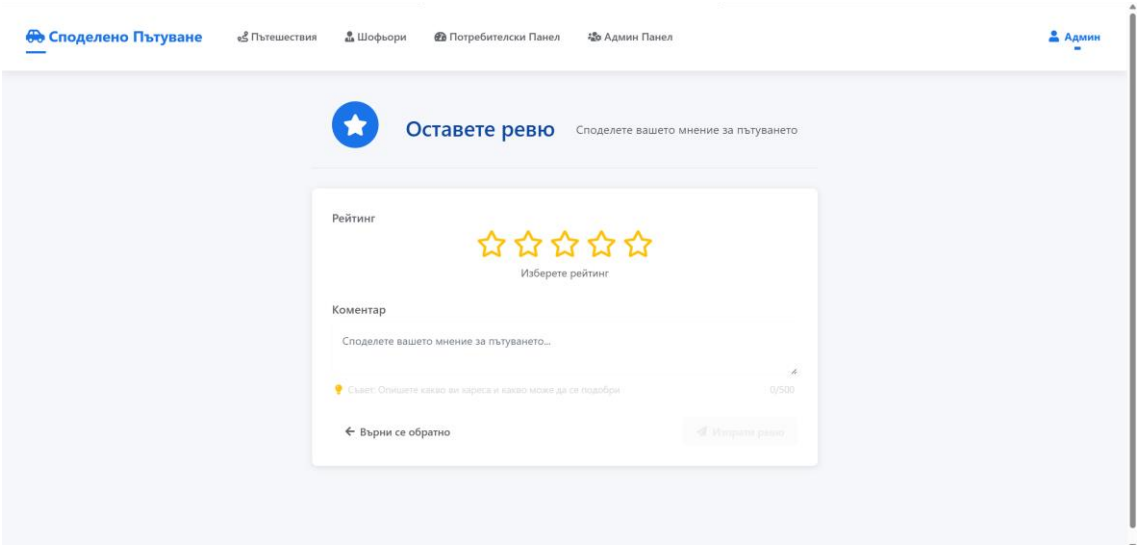
Фигура 47



Фигура 48



Фигура 49



Фигура 50