ДИПЛОМЕН ПРОЕКТ ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

Тема: „Информационна платформа за система за запазване на билети“

Автор:

Цветан Антонов Тинев, клас XII В

Ръководител:

Петър Георгиев

Бургас 2023СЪДЪРЖАНИЕ

[1 Увод 4](#_Toc132938169)

[2 Цели и обхват на софтуерното приложение 4](#_Toc132938170)

[2.1 Цели 4](#_Toc132938171)

[2.2 Автобусни компании, маршрути и запазване на билети 6](#_Toc132938172)

[2.2.1 Автобусни компании 7](#_Toc132938173)

[2.2.2 Маршрути 8](#_Toc132938174)

[2.2.3 Билети 8](#_Toc132938175)

[2.3 Роля “Потребител” 9](#_Toc132938176)

[2.4 Роля “Управител на компания” 9](#_Toc132938177)

[2.5 Роля “Администратор” 9](#_Toc132938178)

[2.6 Видове страници и тяхното предназначение 9](#_Toc132938179)

[3 Анализ на решението 11](#_Toc132938180)

[3.1 Потребителски изисквания и работен процес 11](#_Toc132938181)

[3.2 Примерен потребителски интерфейс 13](#_Toc132938182)

[3.3 Диаграми на анализа 17](#_Toc132938183)

[3.3.1 Съдържание на таблиците 18](#_Toc132938184)

[3.4 Модел на съдържанието / данните 23](#_Toc132938185)

[4 Дизайн 23](#_Toc132938186)

[4.1 Реализация на архитектурата на приложението 25](#_Toc132938187)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 26](#_Toc132938188)

[4.3 Организация и код на заявките към база от данни 28](#_Toc132938189)

[4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 28](#_Toc132938190)

[5 Ефективност и бързодействие на решението 29](#_Toc132938191)

[6 Тестване 30](#_Toc132938192)

[6.1 Методология на теставне 30](#_Toc132938193)

[6.2 Цели и обхват 31](#_Toc132938194)

[6.3 Описание на използваните типове тестове 31](#_Toc132938195)

[7 Заключение и възможно бъдещо развитие 39](#_Toc132938196)

[8 Използвани литературни източници и Уеб сайтове 42](#_Toc132938197)

[9 Приложения 43](#_Toc132938198)

Таблица 1 Видове страници 10

# Увод

Трябва да се обърне внимание на факта, че в момента запазването на билети по електронен път е налично само за някои от големите компании за автобусни превози, които имат свои уеб приложения или използват такива на посредници. По-малките компании, които често предлагат същото качество на услугата за по-ниска цена, нямат финансовия ресурс да поддържат такава услуга, а посредниците често работят само с най-големите от бранша. Това намалява разнообразието на много от потребителите и те често биват принудени да избират по скъпи опции тъй като не знаят за другите по-евтини алтернативи поради липсата на тяхното присъствие в интернет пространството.

Именно заради това проектът за информационна платформа за система за запазване на билети има за цел да създаде централизирана платформа, в която всякакви автобусни компании могат да предлагат своите услуги, което предлага на потребителите едно голямо разнообразие и подпомага правилният им избор при запазването на билети.

# Цели и обхват на софтуерното приложение

## Цели

Целта на този проект е да се разработи цялостна онлайн платформа за управление и резервиране на билети за транспорт, улесняваща процеса, както за транспортните компании, така и за пътниците. Тя предоставя лесна за използване централизирана система за управление на транспортни графици, подробности за маршрута, информация за автобуси и свободни места. Платформата има за цел да подобри ефективността и потребителското изживяване в целия процес на резервиране на билети.

Ключови цели на проекта са:

1. Опростяването на процеса на резервиране на билети: разработването на интуитивен и удобен за потребителя интерфейс, позволяващ на пътниците лесно да търсят, филтрират и сравняват опциите за транспорт, въз основа на критерии като час на заминаване, продължителност и цена. Това позволява на потребителите да вземат информирани решения, когато резервират билети за желаните от тях маршрути, като същевременно предлага допълнителни функции като избор на място, потвърждение на резервацията и възможност да управляват резервациите си онлайн.
2. Улесняване на по-доброто управление на транспортните компании: чрез осигуряване на цялостен административен панел, който позволява на транспортните компании да управляват ефективно всички аспекти на своя бизнес, от управление на автобусен парк и планиране на маршрути, до създаване и поддръжка на график. Това също така ще включва наблюдение и анализиране на ефективността на техните услуги, идентифициране на области за подобрение и прилагане на промени за подобряване на оперативната ефективност и удовлетвореността на клиентите.
3. Поддръжка на управление на потребителски акаунти и ролеви контрол на достъпа: Проектирането на сигурна система за удостоверяване и оторизация, позволява различни потребителски роли, включително пътници, мениджъри на компании и администратори. Всяка роля има свой собствен набор от разрешения и възможности в рамките на платформата, гарантирайки, че чувствителната информация и функционалност са защитени и достъпни само за оторизирани потребители.
4. Намаляване на вредните емисии: насърченото използване на обществен транспорт улеснява потребителите да намират и резервират екологично чисти транспортни опции. Това подпомага за намаляване на въздействието върху околната среда от използването на лични превозни средства, намалява задръстванията и допринася за общата цел за създаване на по-устойчиво бъдеще.

## Автобусни компании, маршрути и запазване на билети

Какво представляват автобусните компании?

* Това са компаниите, които предлагат своите услуги в системата. Управителите на компаниите ще имат профили, с които да ги създадат компании, да добавят автобуси, които изпълняват маршрутите. В допълнение те могат да създават конкретни маршрути.

Какво представляват маршрутите?

* Всяка компания може да добавя дестинации до различни географски точки в страната, а и в чужбина, за които потребителите ще запазват билети.

Какво представляват билетите?

* Според българския тълковен речник билетът е листче, картонче или код, с който се купува и служи за удостоверяване на право за използване на превозно средство. В случая на приложението, билетите се запазват в електронен вариант и няма да се заплащат. Вместо това компаниите взимат информация за това кои потребители са запазили билети за маршрути, предлагани от тях и те са тези, които управляват процесите на издаване на билети и заплащането им.

### Автобусни компании

Когато потребител се регистрира и акаунтът му получи роля “Управител на компания” той има правомощието да създаде компания с име и лого и след това той добавя автобуси, на които дава имена и записва броя на местата в него, с което системата знае колко билета могат да се издават за маршрута, който автобуса изпълнява.

В случай, че управителя на компания е допуснал грешка при именуването на компанията или е избрал грешно лого, той има възможността да поправи грешката си и да я редактира. Той също така може да я изтрие в случай, че компанията прекрати дейността си, но само когато няма маршрути, които все още да се предлагат в системата, за да се предотврати изтриването на компанията по погрешка, което ще доведе до много проблеми както за компанията, така и за потребителите, които са си запази билет за пътуване за маршрути, предлагани от компанията. Последно трябва да се отбележи, че управителя на компанията може да гледа детайли за нея, включително кога е създадена и последно променяна.

### Маршрути

Веднага след като компанията и автобусите са създадени, информация за маршрути, които предлага съответната компания, се въвежда от управителя. Избират се начална и крайна дестинация, дата на заминаване, дата на пристигане. Има и опционални полета за ден от седмицата, в който този маршрут може да се изпълнява, и броя на повторения, чрез които се определя колко време този маршрут се предлага.

Също така се въвежда и цена в диапазона от 1 стотинка до 9.999 лева. След като и последното задължително поле е въведено коректно и без да излизат съобщения за грешки, маршрутът е готов да се предлага за потребителите на системата.

### Билети

Всеки потребител след като си направи профил в системата може да достъпи страницата за запазвне на билети. Там има филтриране по специални критерии. На базата на това от и до коя дестинация иска да пътува, и кога иска да пътува и също с компанията, която иска да осъществи това, потребителите могат да изберат свой маршрут. При избран маршрут, потребителят е длъжен да избере и на кое място иска да седи по време на пътуването, а в случаите, когато иска двупосочен билет – той има правото да избере маршрут са връщане като за него се запазва същото място, което избира за първоначалната дестинация. След запазването на билета, потребителя може да отиде в друга страница, където да гледа всички билети, които е запазил. След като е решил

## Роля “Потребител”

Това е ролята с най-малко правомощия. С нея потребителите имат достъп само до страниците за запазване и преглеждане на своите билети.

## Роля “Управител на компания”

Потребителите с тази роля имат същите правомощия като предишната, но също и възможности за създаване на компании, автобуси и маршрути, които да предлагат.

## Роля “Администратор”

Администраторите са тези, които имат достъп до повечето страници в системата, включително и до панел, в който те могат да редактират информация за други потребители, да им дават повече правомощия под формата на роли и да променят и изтриват транспортните компании.

## Видове страници и тяхното предназначение

|  |  |  |  |
| --- | --- | --- | --- |
| Роля на потребителя  Страница | Потребител | Управител на компания | Администратор |
| Меню за запазване на билети | ☑ | ☑ | ☑ |
| Меню за преглеждане на запазени билети | ☑ | ☑ | ☑ |
| Меню за управление на транспортни компании |  | ☑ | ☑ |
| Меню за управление на автобусите |  | ☑ | ☑ |
| Меню за управление на маршрутите |  | ☑ |  |
| Страница със списък със администраторите |  |  | ☑ |
| Страница за управление на потребителите |  |  | ☑ |

Таблица 1 Видове страници

# Анализ на решението

## Потребителски изисквания и работен процес

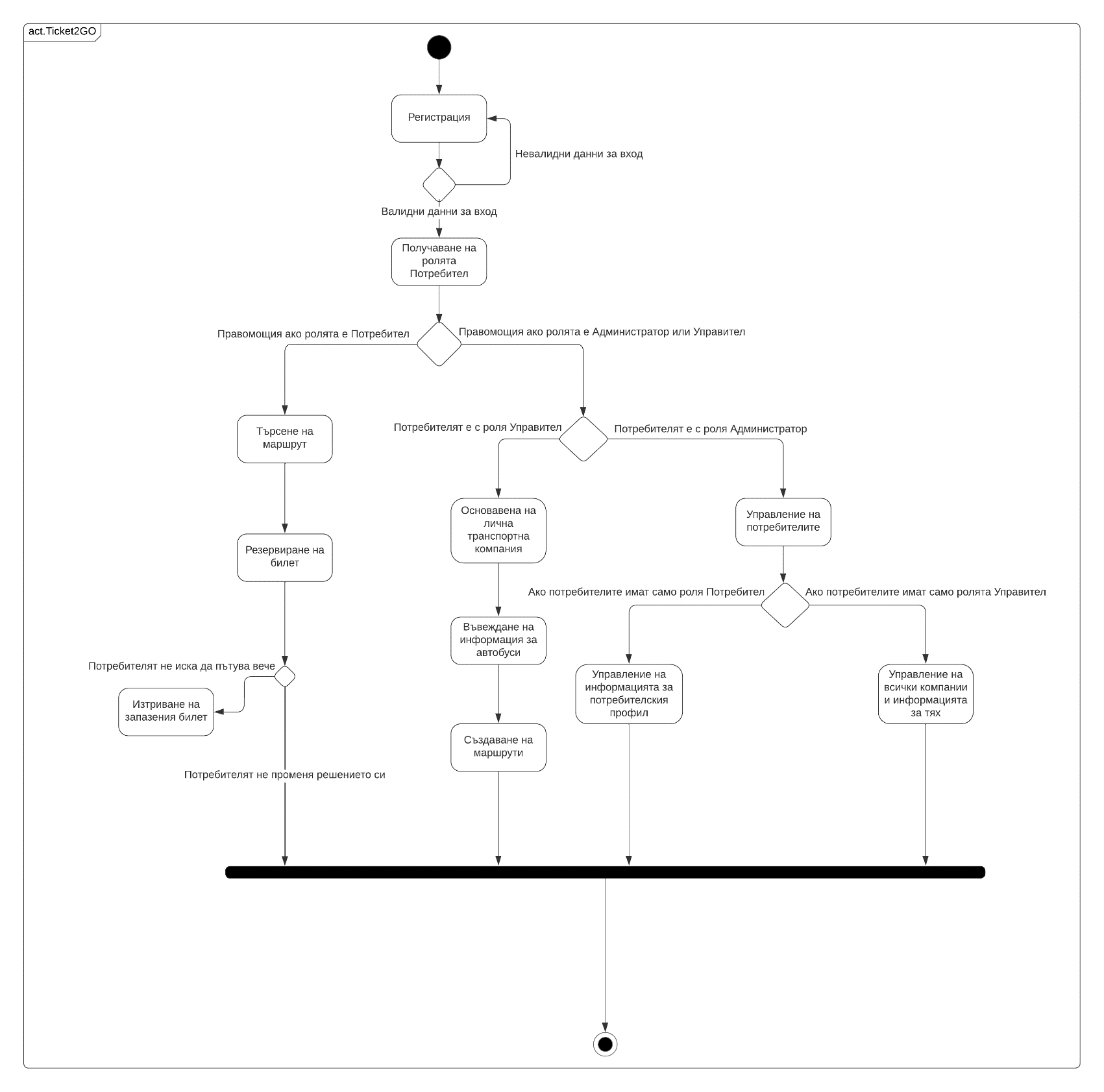
Потребителските изисквания са приложението да бъде с лесен за използване графичен интерфейс, който да помогне с по-лесното използване на софтуера.

Вход: Данните за вход на системата се получават от регистрацията на нови потребители, създаването на транспортни компании, автобуси и маршрути, с които потребителите ще избират превоза до желаната от тях дестинация. Тези данни се въвеждат от потребителите чрез формите за регистрация и публикувана на сайта

Обработка: Получените данни се съхраняват в бази данни, изградена с MS SQL сървър. Процесът на обработка включва филтриране на маршрутите по предварително зададени критерии и препоръчани резултати. Обработените данни се съхраняват за бъдещи препоръки.

Изходът на системата включва:

* Предоставяне на списък с маршрути за пътуване според критериите от потребителя
* Предоставяне на функционалност за комуникация между автобусните компании и пътниците.
* Предоставяне на административен панел за управление на потребители и други данни в системата.
* Получените резултати се използват от потребителите на сайта за намиране на подходящия маршрут, управляване на системата и анализиране на тенденциите.

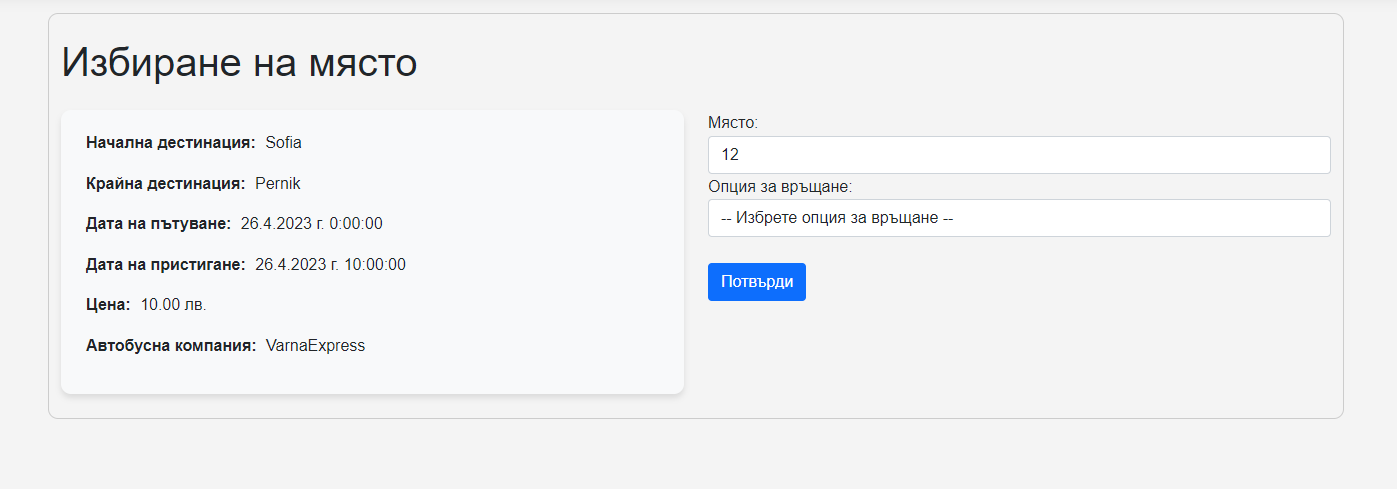
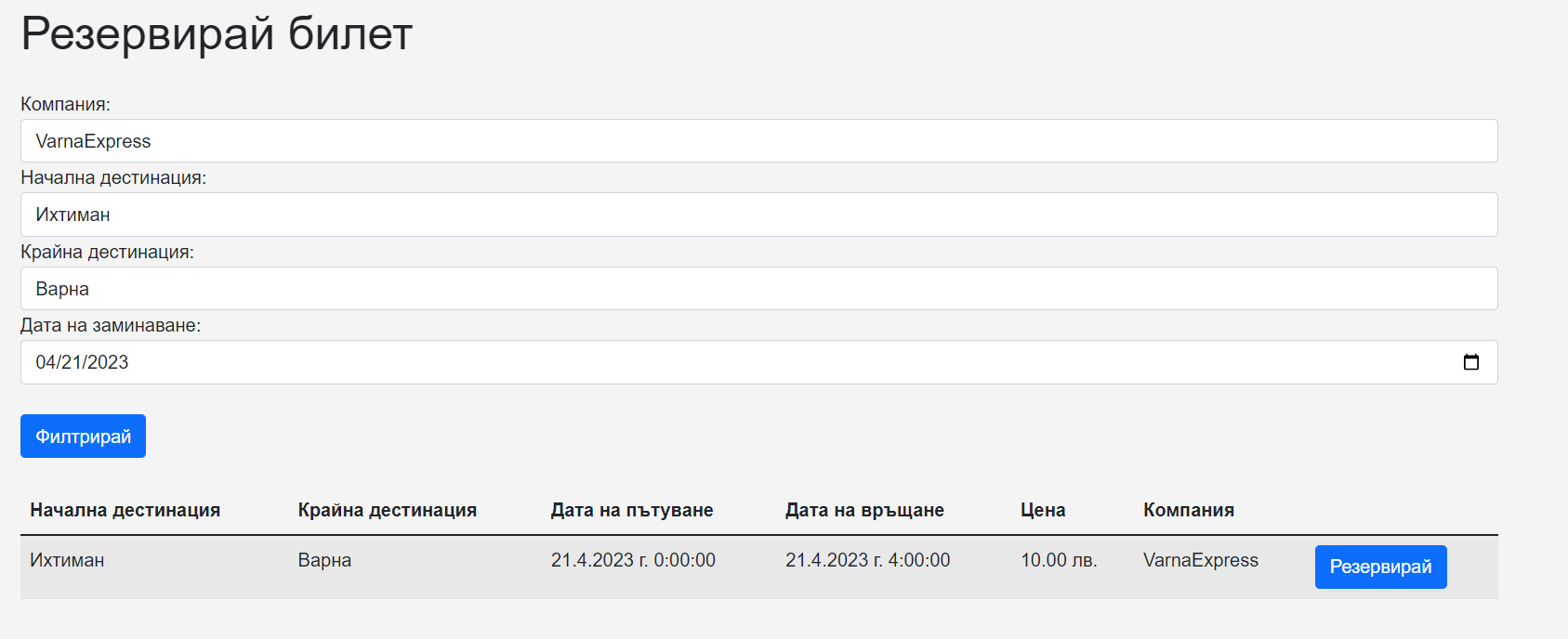
Във фигура 1 е представена диаграма на последователността (Activity Diagram), чрез която се представя процеса на работа. След вписването в програмата се проверява дали профилът, който е използван за вписване в системата, е с администраторски, управителски или потребителски права. Чрез диаграмата се представят функционалностите, които могат да се изпълняват от потребителите с определената роля.

Фигура 1- Диаграма на последователността

## Примерен потребителски интерфейс

Фигура 2 Страница за създаване на маршрути

Фигура 2 е примерен интерфейс за страницата за създаване на маршрути. След въвеждане на информация за полетата за начална и крайна дестинация, дати на потегляне и пристигане, продължителност на пътя, с кой автобус ще се пътува, ден от седмицата, в който този маршрут се повтаря и колко пъти и накрая цена. След като се натиска бутона “Създай”, ако всички полета са въведени коректно и стойностите са в допустимия лимит – маршрут се създава успешно и се показва на потребителите в страницата за запазване на билети. За цвят на полетата е избран бяло, светло сиво за фон на страницата. Текстът за имената и стойностите в полетата е с черен фон. Бутонът “Създай” е син, докато текстът е с бял фон. С бял фон е и текстът за бутона “Списък с маршрути”, който връща потребителите обратно в страницата с маршрутите, но цветът този път е тъмно сив.

Фигура 3 е примерен интерфейс на страницата за търсене на маршрути. При желание от потребителя да намери конкретна дестинация, след въвеждане на данни за компанията, с която иска да пътува, началната и крайната дестинация и датата на пътуване, ще се покажат всички налични избори. След това той може да натисне бутона “Резервирай”, за да продължи с избора си. Цветът на страницата отново е светло сив, а бутоните са с син цвят и бял текст. Останалата част от текста е с черен цвят.

Фигура 3Страница за търсене на маршрути

Фигура 4 Страница за запазване на билети

Фигура 4 е примерен интерфейс на страницата за запазване на билет. Вляво се показва информация за маршрута и потребителят трябва да си избере място, на което ще седи. При желание за избиране на второстепенна дестинация вдясно потребителят може да избере маршрут, с който да се върне обратно. Цветът на страницата е сив, докато полетата с информация и списъците са бели. Текстът навсякъде е с черен фон, докато този в бутона “Потвърди” е с бял, тъй като бутонът е син и това прави добра цветова гама.

Фигура 5 Карта на цветовете

Фигура 5 е карта на най-често срещаните цветове в интерфейса на приложението.

## Диаграми на анализа

Фигура 6 ER диаграма

Фигура 6 визуализира ER диаграмата, която описва дизайна на базата данни. Тя се състои от 14 таблици, 3 от които са свързващи таблици (mapping tables) и 7, които са системни. Всички таблици, които са именувани в стил AspNetСъществителноМножественоЧисло са системни таблици, които се добавят заради работата със зависимостта Microsoft.AspNetCore.Idеntity. В зависимост от целите на проекта, не всички таблици се използват задължително и често се случва повечето да останат празни. Същото важи и за полетата на таблица AspNetUsers.

Базата данни е в трета нормална форма (3NF), защото няма транзитивна зависимост и във всяка таблица полетата зависят от целия първичния ключ, а не само на част от него.

### Съдържание на таблиците

Таблицата с име “AspNetUsers” разполага с 17 полета

* Id: уникален идентификатор за всеки потребител, представен като низ от 450 символа.
* FirstName: ненулева колона от тип nvarchar, която съхранява първото име на потребителя. Има максимален размер от 255 символа.
* LastName: ненулева колона от тип nvarchar, която съхранява фамилията на потребителя. Има максимален размер от 255 символа.
* UserName: ненулева колона от тип nvarchar, която съхранява потребителското име на потребителя. Има максимален размер от 256 символа.
* NormalizedUserName: ненулева колона от тип nvarchar, която забранява на потребители да се регистрират с едно и също потребителско име като се различават само по главни букви. Максимален размер от 256 символа.
* Email: ненулева колона от тип nvarchar, която съхранява електронната поща на потребителите Максимален размер от 256 символа.
* NormalizedEmail: ненулева колона от тип nvarchar, която предотвратява регистрирането на потребители с една и съща електронна поща, различаващ се само по главни букви. Максимален размер от 256 символа.
* EmailConfirmed: колона от тип bit, определяща дали електронната поща е потвърдена.
* PasswordHash: колона от тип nvarchar(MAX), която съхранява в хеширан вид паролата на потребителя.
* SecurityStamp: колона от тип nvarchar(MAX), която се използва да генерира нова бисквита за защита от атаки.
* ConcurrencyStamp: колона от тип nvarchar(MAX), с която се предотвратява оптимистична конкурентност(optimistic concurrency).
* PhoneNumber: колона от тип nvarchar(MAX), която съхранява телефонният номер на потребителя във формата на низ.
* PhoneNumberConfirmed: ненулева колона от тип bit, с която се определя дали телефонният номер на потребителя е потвърден.
* TwoFactorEnabled: ненулева колона от тип bit, която определя дали потребителят има двустепенно удостоверяване.
* LockoutEnd: колона от тип datetimeoffset, с която се определя кога профилът ще бъде отключен.
* LockoutEnabled: колона от тип bit, която определя дали профилът е заключен.
* AccessFailedCount: ненулева колона от тип int, която запазва броя на неуспешните опити за влизане на потребителя.

Таблица с име на “AspNetUserLogins” разполага с 4 полета, 2 от които образуват съставен ключ, а именно LoginProvider и ProviderKey:

* LoginProvider: колона от тип nvarchar с максимален размер 450 символа, която представлява външният удостоверител.
* ProviderKey: колона уникален идентификатор от тип nvarchar за външния удостоверител. Максимален размер 450 символа.
* ProviderDisplayName: колона от тип nvarchar с максимален размер 450 низа съхранява името на външия удостоверител.
* UserId: колона външен ключ за таблицата AspNetUsers от тип nvarchar с максимален размер 450 низа.

Таблица с име “AspNetUserTokens” разполага с 4 полета, 3 от които (UserId, LoginProvider и Name) образуват съставен ключ:

* UserId: външен ключ към таблицата AspNetUsers.
* LoginProvider: външен ключ към таблицата AspNetUserLogins.
* Name: колона от тип nvarchar, която съхранява името на токена. Максимална стойност 450 символа.
* Value: колона от тип nvarchar(MAX), която съхранява стойността на токена.

Таблица с име “AspNetUserRoles” е междинна таблица между таблиците AspNetRoles и AspNetUsers. Тя разполага с 2 полета, които образуват композитен ключ:

* UserId: външен ключ към таблицата “AspNetUsers” от тип nvarchar с максимална стойност 450 символа
* RoleId: външен ключ към таблицата “AspNetRoles” от тип nvarchar с максимална стойност 450 символа.

Таблицата с име “AspNetRoles” разполага с 4 полета:

* Id: уникален идентификатор за всяка роля. Той е от тип nvarchar и е с максимална стойност от 450 символа.
* Name: колона от тип nvarchar, която съхранява името на ролята. Максимална стойност от 450 символа.
* NormalizedName: ненулева колона от тип nvarchar, която предотвратява да се създават роли с еднакви имена, различаващи се само по главни букви. Максимален размер от 256 символа.
* ConcurrencyStamp: колона от тип nvarchar(MAX), с която се предотвратява оптимистична конкурентност(optimistic concurrency).

Таблицата с име “AspNetRoleClaims” разполага с 4 полета:

* Id: уникален идентификатор за всяко твърдение. Той е от тип nvarchar и е с максимална стойност от 450 символа.
* RoleId: външен ключ към таблицата “AspNetRoles”. Той е от тип nvarchar и е с максимална стойност от 450 символа.
* ClaimType: колона от тип nvarchar(MAX), която съхранява типа на твърдението.
* ClaimValue: колана от тип nvarchar(MAX), която съхранява стойността на твърдението.

Таблицата с име “AspNetUserClaims” разполага с 4 полета:

* Id: уникален идентификатор за всяко твърдение. Той е от тип nvarchar и е с максимална стойност от 450 символа.
* UserId: външен ключ към таблицата “AspNetUsers” от тип varbinary с максимален размер от 450 символа.
* ClaimType: колона от тип nvarchar(MAX), която съхранява типа на твърдението.
* ClaimValue: колана от тип nvarchar(MAX), която съхранява стойността на твърдението.

Таблицата с име “Admins” разполага с 2 полета:

* AdminId: Уникален идентификатор за всяко твърдение. Той е от тип uniqueidentifier.
* ApplicationUserId: Външен ключ към таблицата “AspNetUsers” от тип nvarchar с максимален размер 450 символа.

Таблицата с име “TransportCompaniesAspNetUsers” е междинна таблица между таблиците “AspNetUsers” и “TransportCompanies”. Разполага с 2 полета, които образуват композитен ключ.

Таблицата с име “TransportCompanies” разполага с 2 полета:

* TransportCompanyId: уникален идентификатор от тип uniqueidentifier за всяка компания.
* Name: ненулева колона от тип nvarchar(MAX), която съхранява името на транспортната компания.
* Logo: ненулева колона от тип varbinary(MAX), която съхранява файла за логото на компанията под формата на масив от битове.
* DateCreated: ненулева колона от тип datetime2, която съхранява датата на създаване на компанията.
* DateEdited: ненулева колона от тип datetime2, която съхранява последният час на промените.

Таблицата с име “Buses” разполага с 4 полета:

* BusId: уникален идентификатор от тип uniqueidentifier за всеки автобус.
* TransportCompanyId: външен ключ от тип uniqueidentifier, който сочи към таблицата “TransportCompanies”.
* SeatsNumber: колона от тип tinyint, която съхранява броя на седалките в автобуса.
* Name: колона от тип nvarchar с максимален размер от 100 символа, която съхранява името на автобуса.

Таблицата с име “Destinations” разполага с 9 полета:

* DestinationId: уникален идентификатор от тип uniqueidentifier за всяка дестинация.
* BusId: външен ключ от тип uniqueidentifier, който сочи към таблицата “Buses”.
* StartingDestination: ненулева колона от тип nvarchar(MAX), която съхранява името на началната дестинация.
* FinalDestination: ненулева колона от тип nvarchar(MAX), която съхранява името на крайната дестинация.
* Duration: ненулева колона от тип time, която съхранява времетраенето на маршрута.
* Departure: ненулева колона от тип datetime2, която съхранява датата на заминаване.
* TimeOfArrival: ненулева колона от тип datetime2, която съхранява датата на пристигане.
* RepeatingDayOfWeek: колона от тип int, която съхранява дните от седмицата.
* Price: ненулева колона от тип decimal(6,2), която съхранява общата цена на билета.

Таблицата с име “TicketsDestinations” е междинна таблица между таблиците “Tickets” и “Destinations”. Разполага с 2 полета, които образуват съставен ключ.

Таблицата с име “Tickets” разполага с 4 полета:

* TicketId: уникален идентификатор от тип uniqueidentifier за всеки автобус
* ApplicationUserId: външен ключ от тип uniqueidentifier, който сочи към таблицата “AspNetUsers”
* TotalPrice: ненулева колона от тип decimal(6,2), която съхранява цената на билета.
* SeatNumber: ненулева колона от тип int, която съхранява номера на запазеното място.

Половината от тези диаграми са системни и се добавят заради използването на библиотеката Identity. Повечето от тях може да не се използват и това няма да повлияе негативно на работата на приложението. Част от таблиците и полетата служат за опционални функции като двуфакторно удостоверяване, потвърждаване на пароли, управление на твърдения (Claims), които нито ги има в заданието, нито са задължителни за имплементиране. Добре е да ги има, защото това са важни функционалности за едно уеб приложение. Модифицирането и премахването на част от тези системни таблици може да доведе до проблеми с проекта, поради което тези действия не са препоръчителни.

### Диаграма на случаите

Фигура 7- Диаграма на случаите

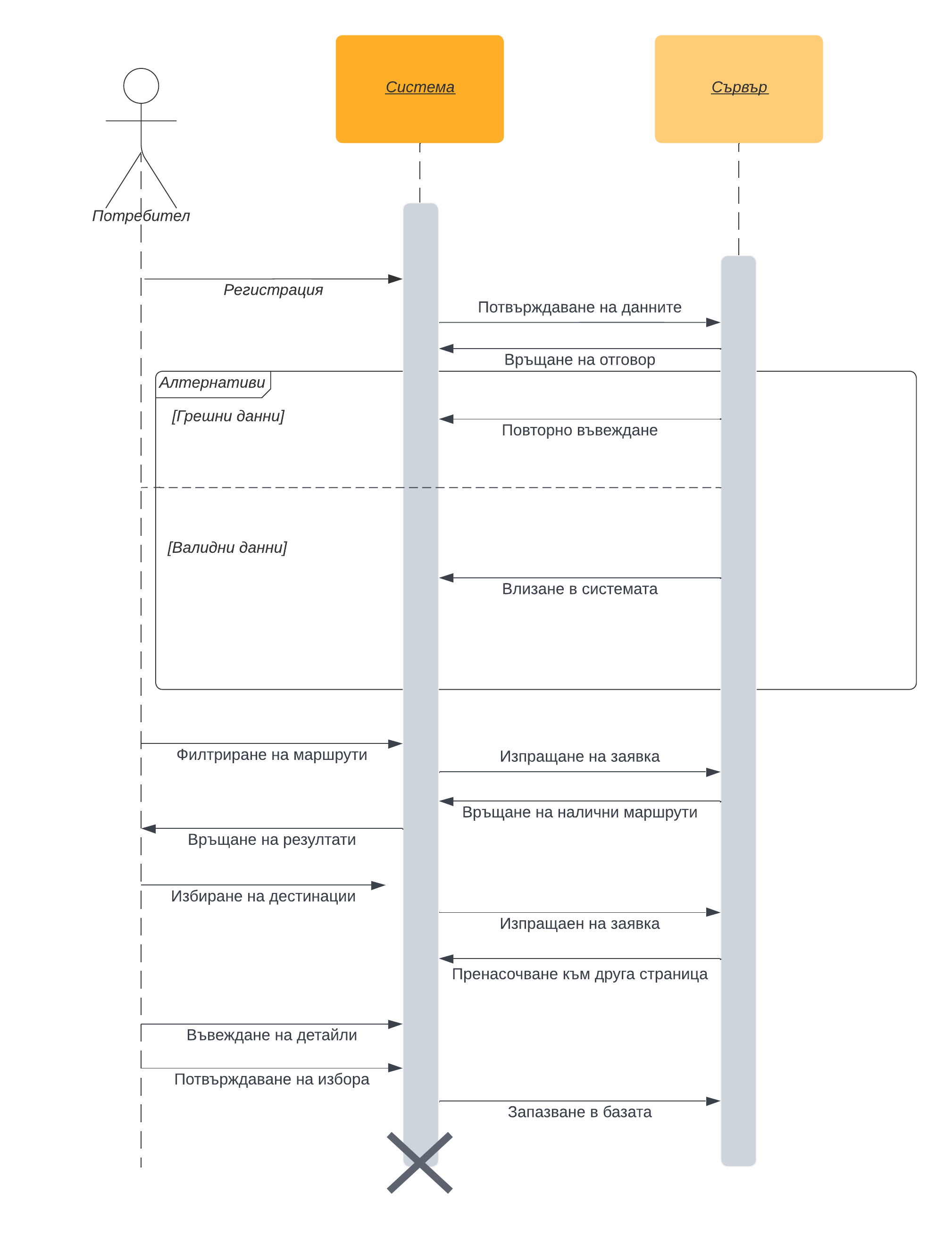
Фигура 7 е диаграма на случай на използване, която илюстрира система за резервация на билети. Системата позволява на потребителите да запазват билети, да преглеждат запазени билети, да филтрират билети по определени критерии, да избират маршрути, да избират места и да избират опции за връщане

В допълнение, системата има функция за административно и фирмено управление, където администраторът и управителят могат да управляват компании, да добавят автобуси, да управляват маршрути и потребители.

Диаграмата включва няколко участници, като потребители, администратори и управители на компании, и различни случаи на използване, като регистрация, управление на билети и управление на компания. Има също така няколко връзки между участниците и случаите на използване, като например връзките на включване между случая на използване на филтърните билети и случаите на използване на опцията за избор на маршрут/използване на място/използване на връщане.

Като цяло тази диаграма на случаите на използване предоставя общ преглед на функционалностите и характеристиките на системата за резервация на билети и как различните участници взаимодействат със системата, за да изпълнят своите задачи.

### Диаграма на последователността

Фигура 8 Диаграма на последователността

Фигурата 8 показва диаграмата на последователността за процеса на запазване на билети. При успешно влизане в системата и при успешно филтриране на маршрутите потребителят избира подходящия за него, въвежда допълнителна информация като запазено място и маршрут за връщане. Ако всички данни са въведени коректно, билетът се запазва в базата данни.

## Модел на съдържанието / данните

Данните и информацията, която се съдържа в приложението са с цел да информират лесно и разбираемо потребителя без значение неговата роля.

На ниво потребител в софтуера, може да се достъпи информация като запазени билети и налични маршрути. Потребителите могат и да следят в реално време добавянето и премахването на маршрути.

На ниво управител на компания могат да се създават компании, автобуси и маршрути.

От административна гледна точка типът информация, с който работят също е текстов. Те могат да наблюдават информацията за потребителите и управителите на транспортните компании, като определят тяхната лична информация, чрез която служителите могат да достъпят системата.

# Дизайн

ASP.NET Core MVC е подходящ избор за проект поради няколко причини:

Съвместимост между операционни системи:

* ASP.NET Core MVC може да работи на различни операционни системи като Windows, Linux и macOS, което улеснява внедряването и стартирането на приложението в различни среди.
* Мащабируемост и производителност: ASP.NET Core е проектиран да бъде лек и бърз, което позволява да създаването на приложения с висока производителност, които могат да се мащабират, за да се справят с голям брой потребители.
* Архитектурен модел MVC: Моделът Модел-Изглед-Контролер (Model-View-Controller) е добре установен модел на проектиране, който насърчава чистото разделяне на проблемите между данните, представянето и логиката на приложението. Това улеснява поддържането, разширяването и тестването на приложението.
* Вградено инжектиране на зависимости: ASP.NET Core включва вградена поддръжка за инжектиране на зависимости (Dependency Injection), което може да помогне за писането на по-модулен и тестван код, като позволява лесно да се инжектират услуги и компоненти в контролери и други класове.
* Богато разнообразие: ASP.NET Core разполага с богато разнообразие от библиотеки, пакети и инструменти, които могат да помогнат за изграждането на системата по-ефективно. Това включва вградена поддръжка за Entity Framework Core, който опростява достъпа и управлението на данни, както и поддръжка за популярни рамки и библиотеки.
* Сигурност: ASP.NET Core има силен фокус върху сигурността, като предоставя вградени функции, за да помогне за защитата от често срещани уязвимости в сигурността, като междусайтови скриптове, фалшифициране на заявки и атаки чрез SQL инжектиране.

В обобщение, ASP.NET Core MVC е отличен избор за проект от такъв мащаб, тъй като предоставя стабилна, мащабируема и голямо разнообразие с широк набор от функции и силна екосистема, което позволява изграждането на висококачествено уеб приложение с лекота.

При тази стратегия разработката е основана на модел за работа наречен MVC, подхода на работа по проекта започна със създаването първо с писането на кода, от който ще се генерира схемата на базата данни. Езикът за програмиране използван за направата на проекта е C#, и проектът представлява ASP.NET CORE уеб приложение. Изграждането на потребителският интерфейс (User Interface - UI) е постигнато чрез HTML, CSS, JS и Bootstrap 5.2 (рамка за изграждане на дизайна на изгледа(view)). Базата данни, която е използвана е SQL Server. Зависимостите, които проектът ползва, са:

* Microsoft.AspNet.Identity.EntityframeworkCore
* Microsoft.AspNetCore.Idеntity.UI
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools
* Microsoft.NET.Test.Sdk
* NUnit

## Реализация на архитектурата на приложението

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 Карта на сайта

Фигура 9 показва как изглежда картата на сайта. При успешно влизане в системата потребителят ще бъде допуснат до началната страница, от където в зависимост от правомощията, които има може да достъпи други страници и техните под страници.

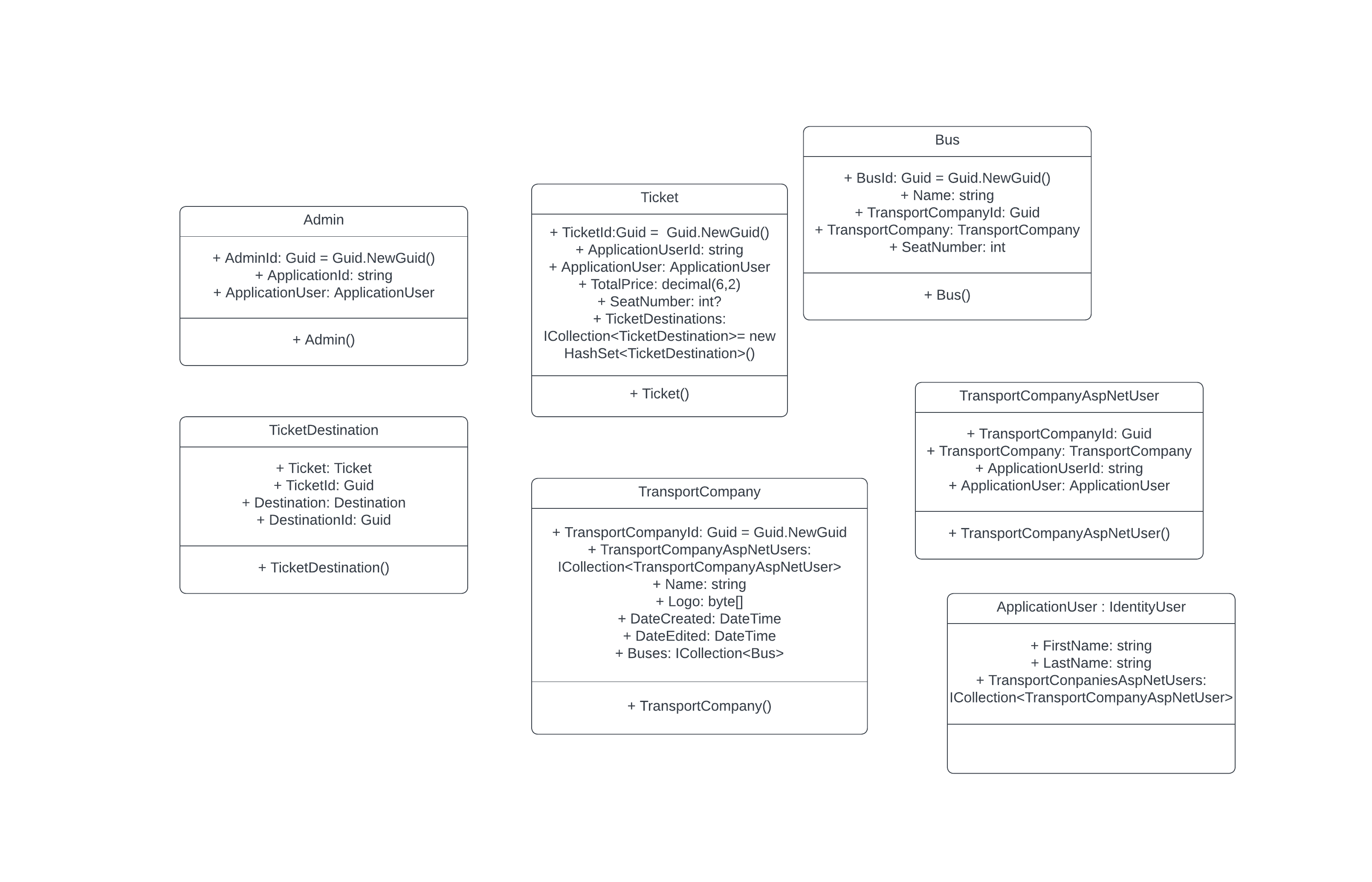
За създаването на системата се използва клиент-сървър архитектурата. Това е най-разпространения начин за разпространение на уеб приложения. В тази архитектура клиентската част общува със сървъра чрез HTTP протокола. Сървърът от своя страна обработва клиентските заявки, обработва данните и ги връща обратно на клиента.

Също така, проектът следва монолитна архитектура, тъй като слоевете за моделите, услугите (services), контролерите и т.н. са в едно уеб приложение. Монолитната архитектура прави тестването и разработването на малки проекти лесно, но в момента който те стават по-големи, трябва да се мисли за промяна на архитектурата, за да не се усложни много разработката на приложението. Използването на микроуслуги представлява евентуална алтернатива.

Добра практика е да има отделен проект за тестване, тъй като това прави тестовете изолирани.

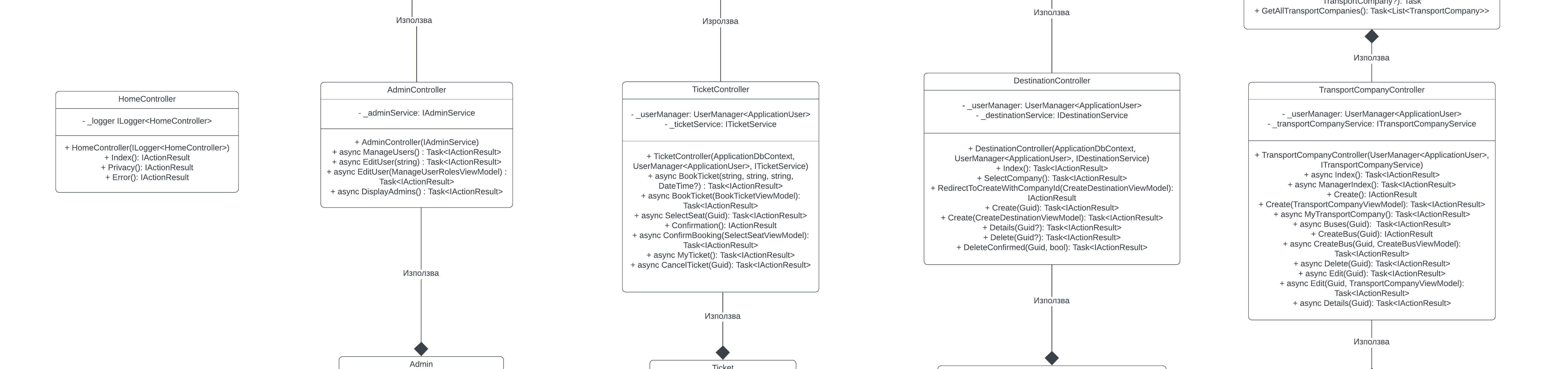
## Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой.

Системата за запазване на билети се състои от 3 слоя. Слой за услугите (Services), слой на контролерите (Controllers) и слой на моделите (Models). За всеки слой е поставена част от класовата диаграма, която показва нагледно атрибутите и методите на всеки обект. Пълната класова диаграма може да се погледне в точка 9 Приложения.

Слой на моделите (Models): 

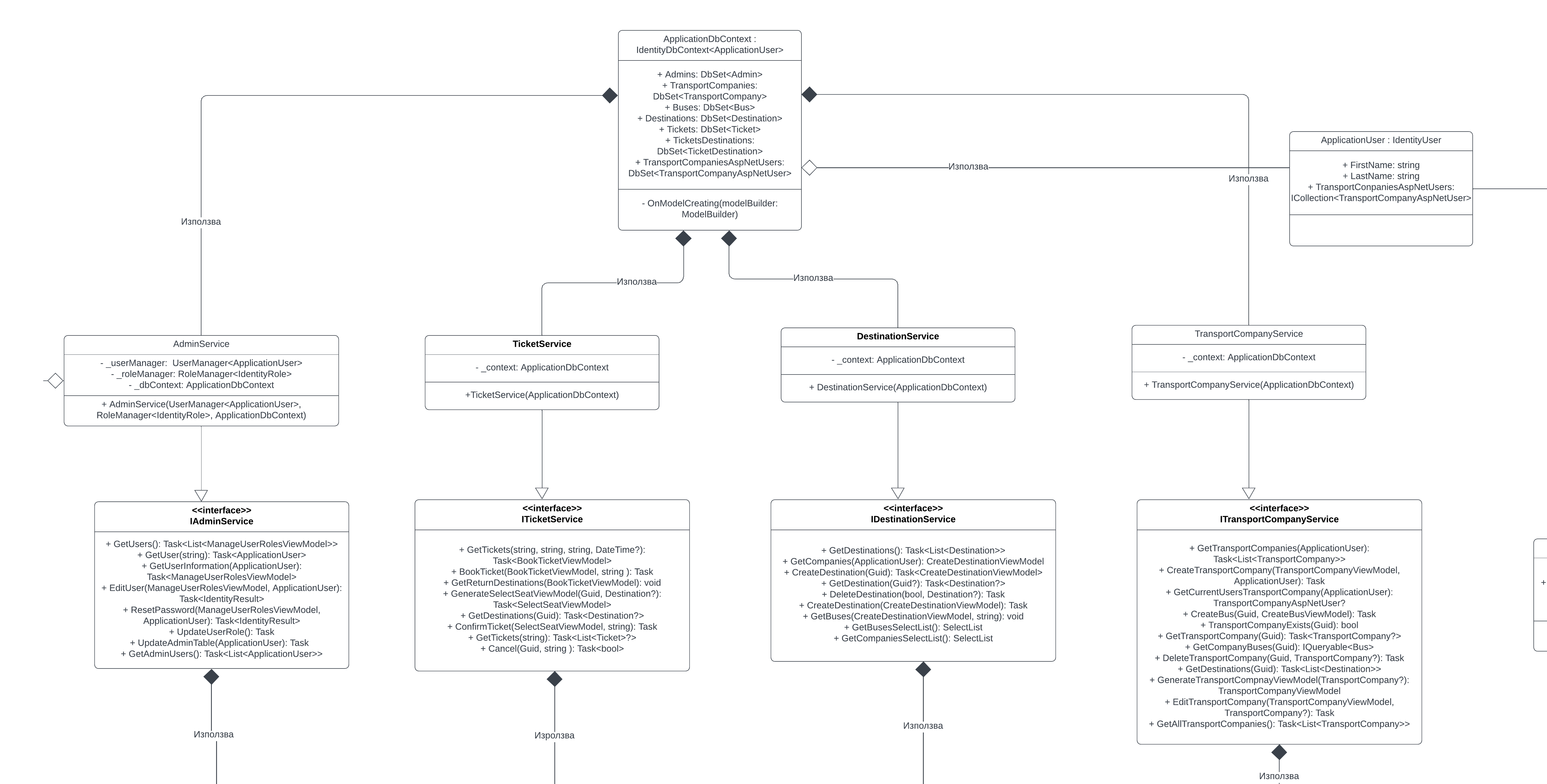
Фигура 10 Модели

* Предназначение: Този слой включва класовете, които представляват моделите на системата. Пример са моделите TransportCompany и Destination
* Библиотеки: Една от библиотеките, която се използва в този слой са System.ComponentModel.DataAnnotations, който осигурява анотации за моделите за валидации, именуване и подобни. Друга библиотека е System.ComponentModel.DataAnnotations.Schema, която предлага атрибути за конфигуриране на схемата за базата данни.
* Методи: Всички модели имат методът за четене на стойността му (getter) и метод за промяна на стойността (setter) за всеки атрибут.

Слой за контролерите (Controllers):

Фигура 11 Контролери

* Предназначение: Обработва входящите HTTP заявки и връща отговори за резервация на билети, управление на потребители и задачи, свързани с управлението на транспортните компании
* Библиотеки: Microsoft.AspNetCore.Mvc е библиотека, използвана в уеб приложенията на ASP.NET Core за внедряване на архитектурния модел Model-View-Controller (MVC). Microsoft.AspNetCore.Identity е библиотека, която предоставя рамка за внедряване на потребителско удостоверяване и оторизация в приложения на ASP.NET Core. Той включва класове и API за създаване и управление на потребителски акаунти, роли и разрешения, както и за хеширане на пароли, двуфакторно удостоверяване и други функции за сигурност.
* Методи: Методи за действие за крайни точки (endpoints), свързани с резервиране на билети, удостоверяване на потребители, управление на роли и задачи на администратора.

Слой за услугите (Services):

Фигура 12 Услуги

* Предназначение: Съдържа бизнес логика за резервиране на билети, управление на потребителите и административни задачи; взаимодейства със слоя за достъп до данни.
* Библиотеки: Microsoft.EntityFrameworkCore, която служи за комуникация с базата данни
* Методи: CRUD операции, валидиране и трансформация на данни, свързани с резервация на билети, удостоверяване на потребителя, управление на роли и задачи на администратора.

## Организация и код на заявките към база от данни

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using WebApp.Areas.Identity.Data;

using WebApp.Data;

using WebApp.Models;

using WebApp.Services.Interfaces;

using WebApp.ViewModels;

namespace WebApp.Services

{

public class DestinationService : IDestinationService

{

private readonly ApplicationDbContext \_context;

public DestinationService(ApplicationDbContext context)

{

\_context = context;

}

public async Task<List<Destination>> GetDestinations()

{

return await \_context.Destinations

.Include(d => d.Bus)

.ThenInclude(b => b.TransportCompany)

.ToListAsync();

}

public CreateDestinationViewModel GetCompanies(ApplicationUser user)

{

string managerId = user?.Id;

var companies = \_context.TransportCompaniesAspNetUsers

.Where(t => t.ApplicationUserId == managerId)

.Select(t => t.TransportCompany)

.ToList();

var companySelectListItems = companies.Select(c => new SelectListItem

{

Value = c.TransportCompanyId.ToString(),

Text = c.Name

}).ToList();

var viewModel = new CreateDestinationViewModel

{

Companies = companySelectListItems

};

return viewModel;

}

public async Task<CreateDestinationViewModel> CreateDestination(Guid companyId)

{

var buses = await \_context.Buses.Where(b => b.TransportCompanyId == companyId).ToListAsync();

var busSelectListItems = buses.Select(b => new SelectListItem

{

Value = b.BusId.ToString(),

Text = b.Name

}).ToList();

var viewModel = new CreateDestinationViewModel

{

SelectedCompanyId = companyId,

Buses = busSelectListItems

};

return viewModel;

}

public async Task<Destination?> GetDestination(Guid? id)

{

return await \_context.Destinations

.Include(d => d.Bus)

.FirstOrDefaultAsync(m => m.DestinationId == id);

}

public async Task DeleteDestination(bool deleteAllRepetitions, Destination? destination)

{

\_context.Destinations.Remove(destination);

if (deleteAllRepetitions)

{

var repeatedDestinations = \_context.Destinations

.Where(d => d.StartingDestination == destination.StartingDestination &&

d.FinalDestination == destination.FinalDestination &&

d.BusId == destination.BusId &&

d.Duration == destination.Duration &&

d.RepeatingDayOfWeek == destination.RepeatingDayOfWeek)

.ToList();

foreach (var repeatedDestination in repeatedDestinations)

{

int daysDifference = (repeatedDestination.Departure - destination.Departure).Days;

if (daysDifference % 7 == 0)

{

\_context.Destinations.Remove(repeatedDestination);

}

}

}

await \_context.SaveChangesAsync();

}

public async Task CreateDestination(CreateDestinationViewModel viewModel)

{

var destinations = new List<Destination>();

if (viewModel.RepeatingDayOfWeek.HasValue && viewModel.NumberOfRepetitions.HasValue)

{

var currentDate = viewModel.Departure;

int repetitions = 0;

while (repetitions < viewModel.NumberOfRepetitions.Value)

{

if (currentDate.DayOfWeek == viewModel.RepeatingDayOfWeek.Value)

{

destinations.Add(new Destination

{

StartingDestination = viewModel.StartingDestination,

FinalDestination = viewModel.FinalDestination,

Duration = viewModel.Duration,

Departure = currentDate,

TimeOfArrival = currentDate.Add(viewModel.Duration),

BusId = viewModel.SelectedBusId.Value,

RepeatingDayOfWeek = viewModel.RepeatingDayOfWeek,

Price = viewModel.TotalPrice

});

repetitions++;

}

currentDate = currentDate.AddDays(1);

}

}

else

{

destinations.Add(new Destination

{

StartingDestination = viewModel.StartingDestination,

FinalDestination = viewModel.FinalDestination,

Duration = viewModel.Duration,

Departure = viewModel.Departure,

TimeOfArrival = viewModel.TimeOfArrival,

BusId = viewModel.SelectedBusId.Value,

RepeatingDayOfWeek = viewModel.RepeatingDayOfWeek,

Price = viewModel.TotalPrice

});

}

\_context.AddRange(destinations);

await \_context.SaveChangesAsync();

}

public void GetBuses(CreateDestinationViewModel viewModel, string companyId)

{

viewModel.Buses = \_context.Buses

.Where(b => b.TransportCompanyId == new Guid(companyId))

.Select(b => new SelectListItem

{

Value = b.BusId.ToString(),

Text = b.Name

}).ToList();

}

public SelectList GetBusesSelectList()

{

return new SelectList(\_context.Buses, "BusId", "Name");

}

public SelectList GetCompaniesSelectList()

{

return new SelectList(\_context.TransportCompanies, "CompanyId", "Name");

}

}

}

Класът DestinationService дефинира методи за взаимодействие с таблицата Destinations в базата данни.

Класът е конструиран с инстанция на ApplicationDbContext, който се използва за взаимодействие с базата данни чрез Entity Framework Core.

Методите в класа са организирани по тяхната цел и тип връщане, като GetDestinations(), CreateDestination() и GetCompanies().

Методът GetDestinations() връща списък с обекти Destination, като прави заявка в таблицата Destinations и включва свързани обекти Bus и TransportCompany.

Методът GetCompanies() отправя запитване към таблицата за присъединяване на TransportCompaniesAspNetUsers, за да намери транспортните компании, свързани с даден потребител, и ги връща като списък с обекти SelectListItem.

Методът CreateDestination() отправя запитване към таблицата Buses, за да намери автобусите, свързани с дадена транспортна компания, и ги връща като списък с обекти SelectListItem.

Методът GetDestination() отправя запитване към таблицата Destinations, за да намери отделна дестинация по нейния ID и включва свързани обекти на Bus.

Методът DeleteDestination() изтрива местоназначение от базата данни и, по избор, всички негови повтарящи се екземпляри въз основа на определени критерии.

Методът CreateDestination() създава нови дестинационни обекти в базата данни въз основа на входния CreateDestinationViewModel, който съдържа информация за свойствата на дестинацията и повтарящия се график.

Методите GetBuses() и GetCompaniesSelectList() просто правят запитвания към съответните им таблици, за да върнат обекти SelectList за използване в падащи менюта или други компоненти на потребителския интерфейс.

Като цяло класът DestinationService е добре организиран и разделя притесненията по предназначение и тип връщане, като същевременно използва Entity Framework Core за обработка на заявки към база данни по структуриран и ефективен начин.

## Наличие на потребителски интерфейс (конзолен, графичен, уеб)

Графичният интерфейс е разработен с помощта на HTML,CSS, Bootstrap 5 и JavaScript. Той се състои от няколко страници, които служат за различни цели.

Административен панел

Страницата е предназначение за редактирането на информацията за потребителите като електронна поща, парола и имена. В страницата администраторът може да дава повече правомощия на потребители като им дава различни роли както и да ги отнема.

Страница за управление на транспортните компании

В тази страница управителите на транспортните компании могат да създават профил на транспортната компания, да добавят автобуси към нея, с които да се извършват маршрутите. Те също така могат да редактират информация за транспортната компания и да наблюдават промените.

Страница за маршрути

Тази страница се използва за преглеждане, изтриване и създаване на маршрути. Администраторът или управителят избира транспортна компания, въвежда данните за маршрут като дестинации, дата на пътуване и връщане, времетраене, цена и автобус, който да изпълнява маршрута.

Страница за запазване на билети

Тази страница визуализира всички налични маршрути и филтри за дата, дестинаци и компания, чрез които потребителите да търсят конкретни маршрути.

Интерфейсът на системата за запазване на билети е проектиран да бъде интуитивен и лесен за ползване и навигация от всички потребители. Административния панел позволява ефективното управление на потребители, докато страницата за транспортните компании и създаването на маршрутите позволяват бързото създаване на графици за маршрутите за месеци напред. Страницата за запазване на билети позлолява бързо търсене, а разнообразието от компании помага на потребителя да направи подходящия избор.

# Ефективност и бързодействие на решението

За да се осигури ефективността и бързодействието на системата за запазване на билети е добре да се разгледат вече съществуващи решения на подобни услуги както и различни бизнес тактики.

Един много добър пример за уеб приложение, с което хората могат да запазват билети до желаната за тях дестинация е приложението на българската автобусна компания Юнион Ивкони, чиято функционалност за запазване на билети улеснява и прави процеса много по-бърз.

Друг добър пример е малайзийската компания BusOnlineTicket, която предлага разнообразие от компании, които да предложат своите услуги. Отделно за всяка дестинация потребителите разполагат с различна и полезна информация, включително и дали билетите могат да се презаверяват и от коя автогара потегля автобуса. Многото филтри позволят потребителите да направят точния избор.

Една статия на китайската компания AllRide дава много разнообразни и добри предложения как една система за запазване на билети да привлича много потребители и да конкурентно способна. Някои от предложенията са свързани с опцията за удостоверяване чрез профили в социални мрежи, онлайн плащания, програма за лоялност, мобилно приложение и много други идеи, които водят до увеличение на потреблението.

В крайна сметка проучването на вече съществуващи решения и използването на различнки бизнес стратегии може да подобри много качеството на системата и да я направи конкурентноспособна и ефективна.

По отношение на ефективността на самите методи е трудно да се изчисли big O нотацията за всеки от тях тъй като зависи от обема на информацията, която се обработва. Поради това повечето методи имат big O нотация от O(n), а други, понеже обработват единични записи са с O(1) нотация. Пример е методът EditTransportCompany, който обработва единичен запис от тип TransportCompany Въпреки това става ясно, че повечето методи са ефективни и времето им за изпълнение е адекватно.

# Тестване

## Методология на теставне

За тестване на логиката в слоя на услугите (services) най-подходяща е методологията за тестване на бяла кутия.(white box testing).

## Цели и обхват

Целта на тестването на услугата е да се гарантира, че тя функционира по предназначение, предоставяйки надеждни и точни резултати на своите потребители. Това ще помогне за подобряване на удовлетвореността на потребителите и изграждане на доверие в услугата.

Обхватът на тестване на слоя за услугите включва всички нейни функционалности и функции, като създаване и извличане на данни, удостоверяване и оторизация на потребителя, обработка на грешки и интеграция с външни системи. Тестването ще се извърши както върху положителни, така и върху отрицателни тестови случаи, за да се гарантира, че услугата се държи правилно при различни сценарии.

## Описание на използваните типове тестове

За тестването на слоя на услугите се използва рамката с отворен код NUnit, която задоволява нуждите. Тестовете се правят в отделен проект с име WebApp.UnitTests, в който има отделен файл за тестване на методите на всяка една услуга.

Тестовете следват подхода на трите А-та (Arrange Act Assert) или подреждане, действане и твърдение. В конструктора се случва етапа на подреждането, където се подготвя контекста на базата в паметта и се зарежда с примерни данни от типа на съответното хранилище. В самите методи се случва етапа на действие, където се вика тествания метод, а накрая се завършва с етапа на твърдението, където се сравнява резултата от извиканата функция с очаквания. Във всички методи за тестване на базата данни се използва using ключовата дума, като в нея се слага променливата на контекста. Това се прави, за да се осигури изхвърлянето на класа на контекста, когато теста приключи.

Първи тест: Тестване за валидно въвеждане на данни за създаване на маршрут

public class DestinationServiceTests

{

private ApplicationDbContext \_context;

private DestinationService \_destinationService;

[SetUp]

public void Setup()

{

var options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new ApplicationDbContext(options);

\_destinationService = new DestinationService(\_context);

}

[Test]

public async Task CreateDestination\_ValidInput\_CreatesDestination()

{

var viewModel = new CreateDestinationViewModel

{

StartingDestination = "New York",

FinalDestination = "Los Angeles",

Duration = TimeSpan.FromHours(6),

Departure = DateTime.Now,

TimeOfArrival = DateTime.Now.AddHours(6),

SelectedBusId = Guid.NewGuid(),

RepeatingDayOfWeek = null,

TotalPrice = 100

};

await \_destinationService.CreateDestination(viewModel);

Assert.AreEqual(1, \_context.Destinations.CountAsync().Result);

var destination = await \_context.Destinations.FirstAsync();

Assert.AreEqual(viewModel.StartingDestination, destination.StartingDestination);

Assert.AreEqual(viewModel.FinalDestination, destination.FinalDestination);

Assert.AreEqual(viewModel.Duration, destination.Duration);

Assert.AreEqual(viewModel.Departure, destination.Departure);

Assert.AreEqual(viewModel.TimeOfArrival, destination.TimeOfArrival);

Assert.AreEqual(viewModel.SelectedBusId, destination.BusId);

Assert.AreEqual(viewModel.RepeatingDayOfWeek, destination.RepeatingDayOfWeek);

Assert.AreEqual(viewModel.TotalPrice, destination.Price);

}

[TearDown]

public void TearDown()

{

\_context.Database.EnsureDeleted();

}

}

Този тест е за метода CreateDestination на класа DestinationService. Методът приема обект CreateDestinationViewModel като вход и създава нова дестинация въз основа на информацията, предоставена в модела на изгледа. Целта на този тест е да провери дали методът CreateDestination правилно създава нова дестинация в базата данни.

В метода Setup се създава база данни в паметта с помощта на ApplicationDbContext и екземпляр на класа DestinationService се създава с помощта на този контекст.

Тестовият метод CreateDestination\_ValidInput\_CreatesDestination създава нов обект CreateDestinationViewModel с валидни входни стойности и извиква метода CreateDestination на екземпляра \_destinationService, създаден в метода Setup. След това тестът проверява дали базата данни съдържа само един целеви запис и че полетата на местоназначението съответстват на стойностите, предоставени в модела на изглед.

Методът TearDown се извиква след завършване на теста и гарантира, че базата данни в паметта е изтрита.

Втори тест: Въвеждане на валиден инпут за създаване на транспортна компания.

public class TransportCompanyServiceTests

{

private ApplicationDbContext \_context;

private ITransportCompanyService \_transportCompanyService;

private UserManager<ApplicationUser> \_userManager;

private ServiceProvider \_serviceProvider;

[SetUp]

public void Setup()

{

IServiceCollection services = new ServiceCollection();

services.AddLogging();

services.AddDbContext<ApplicationDbContext>(options => options.UseInMemoryDatabase("TestDb"));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

services.AddAuthentication();

\_serviceProvider = services.BuildServiceProvider();

var options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new ApplicationDbContext(options);

\_transportCompanyService = new TransportCompanyService(\_context);

\_userManager = \_serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();

}

[Test]

public async Task CreateTransportCompany\_ValidInput\_CreatesTransportCompany()

{

var user = new ApplicationUser

{

UserName = "testuser",

Email = "testuser@example.com",

FirstName = "John",

LastName = "Doe"

};

await \_context.Users.AddAsync(user);

await \_context.SaveChangesAsync();

var stream = new MemoryStream(Encoding.UTF8.GetBytes("This is a test file."));

var viewModel = new TransportCompanyViewModel

{

Name = "Test Transport Company",

Logo = new FormFile(stream, 0, stream.Length, "testLogo.jpg", "image/jpeg")

};

await \_transportCompanyService.CreateTransportCompany(viewModel, user);

var transportCompany = await \_context.TransportCompanies.FirstOrDefaultAsync(tc => tc.Name == viewModel.Name);

Assert.IsNotNull(transportCompany);

}

[TearDown]

public void TearDown()

{

\_context.Database.EnsureDeleted();

}

}

Този тест е за метода CreateTransportCompany на класа TransportCompanyService. Тестът проверява дали методът правилно създава нова транспортна компания с валидни входни данни.

Тестът първо създава нов потребител и го добавя към базата данни в паметта, след което създава обект TransportCompanyViewModel с име и тестов файл с лого. След това методът CreateTransportCompany на класа TransportCompanyService се извиква с този модел на изглед и създадения потребител като параметри. И накрая, тестът потвърждава, че транспортната компания е създадена в базата данни чрез проверка дали съществува запис с даденото име.

Методът SetUp създава контекст на база данни в паметта и доставчик на услуги, които да се използват от теста. Методът TearDown гарантира, че базата данни ще бъде изтрита след теста.

Трети тест: Проверзка за правилно филтриране на дестинации.

public class TicketServiceTests

{

private ApplicationDbContext \_context;

private TicketService \_ticketService;

[SetUp]

public void Setup()

{

var options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new ApplicationDbContext(options);

\_ticketService = new TicketService(\_context);

}

[Test]

public async Task GetTickets\_ValidInput\_ReturnsFilteredTickets()

{

var transportCompany = new TransportCompany { Name = "Test Company", Logo = Encoding.UTF8.GetBytes("test-logo.png") };

\_context.TransportCompanies.Add(transportCompany);

var bus = new Bus { Name = "Test Bus", SeatsNumber = 50, TransportCompany = transportCompany };

\_context.Buses.Add(bus);

var destination = new Destination

{

StartingDestination = "A",

FinalDestination = "B",

Departure = DateTime.Now.AddDays(1),

TimeOfArrival = DateTime.Now.AddDays(1).AddHours(4),

Price = 30.0m,

Bus = bus

};

\_context.Destinations.Add(destination);

await \_context.SaveChangesAsync();

var result = await \_ticketService.GetTickets(transportCompany.TransportCompanyId.ToString(), "A", "B", null);

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Destinations.Count);

var firstResult = result.Destinations.First();

Assert.AreEqual(destination.DestinationId, firstResult.DestinationId);

Assert.AreEqual(destination.StartingDestination, firstResult.StartingDestination);

Assert.AreEqual(destination.FinalDestination, firstResult.FinalDestination);

}

[TearDown]

public void TearDown()

{

\_context.Database.EnsureDeleted();

}

}

Това e тест за класа TicketService в пространството на имената WebApp.Services. Тестът проверява поведението на метода GetTickets на TicketService при валиден вход.

Методът SetUp създава контекст на база данни в паметта с помощта на ApplicationDbContext и инстанцира обект на TicketService с помощта на контекста.

Методът GetTickets\_ValidInput\_ReturnsFilteredTickets създава обект TransportCompany, обект Bus и обект Destination с посочените стойности и ги записва в контекста на базата данни в паметта с помощта на обекта \_context. След това извиква метода GetTickets на обекта \_ticketService с TransportCompanyId на transportCompany, начална дестинация „A“, крайна дестинация „B“ и null като ден от седмицата. Методът връща обект от тип TicketViewModel, който е присвоен на резултатната променлива.

И накрая, тестът използва няколко метода Assert, за да провери дали резултатният обект не е нулев, че броят на обектите DestinationViewModel в свойството Destinations на резултат е равен на 1 и че свойствата на първия обект DestinationViewModel в свойството Destinations на резултата съответстват на свойствата на целевия обект, който е създаден по-рано в теста.

Методът TearDown се извиква след теста и изтрива базата данни в паметта, за да се гарантира, че следващите тестове не са засегнати от този.

# Заключение и възможно бъдещо развитие

Проектът помага за избора на потребителите като предлага разнообразие от много компании. Потребителският интерфейс е улеснен и помага на управителите да създават график за месец напред. Въпреки това вратата за иновации винаги е отворена и някои потенциални планове за бъдещо развитие биха били:

* Разработване на мобилни приложения: В допълнение към уеб-базираната платформа, инвестирането в създаване на собствени мобилни приложения за Android и iOS устройства. Това ще предостави на потребителите по-удобно и безпроблемно изживяване, позволявайки им да резервират билети в движение, да получават известия и да имат достъп до хронологията на резервациите си директно от своите смартфони.
* Поддръжка за чат и помощен център в приложението: Внедряването на интегрирана система за поддръжка за чат, където потребителите могат да задават въпроси и да получават помощ в реално време от агенти за поддръжка на клиенти. Освен това друга идея е създаването на помощен център с често задавани въпроси, уроци и ръководства за отстраняване на проблеми, за да се даде възможност на потребителите да намират отговори бързо и лесно.
* Разширено планиране и оптимизиране на маршрут: Подобряването на приложението, като се добави функция за планиране на маршрут, която помага на потребителите да намерят най-ефективните маршрути и връзки между множество дестинации. Интегрирането на картографски услуги, които да предоставят данни за трафика в реално време, очаквано време за пътуване и предложения за алтернативни маршрути.
* Наличност и резервация на места в реално време: Внедряването на система, която показва информация за наличността на места в реално време за автобуси, позволявайки на потребителите да избират и резервират предпочитаните от тях места без никакви проблеми. Тази функция може да подобри потребителското изживяване чрез предоставяне на актуална информация и минимизиране на шансовете за свръхрезервация или проблеми с недостъпността.
* Персонализирани препоръки и потребителско изживяване: Разработване на механизъм за препоръки, който анализира потребителските предпочитания, историята на резервациите и популярните маршрути, за да предложи най-подходящите опции за транспорт. Тази функция може да помогне на потребителите да открият нови маршрути, да спестят време за вземане на решения и да подобри цялостното удовлетворение на потребителите.
* Изчерпателна програма за лоялност: Създаването на стабилна програма за лоялност, която награждава потребителите с точки, отстъпки или изключителни оферти за чести резервации, препоръки или предоставяне на обратна връзка. Това може да насърчи повторен бизнес, да подобри задържането на клиенти и да подобри лоялността към марката.
* Интегриране с допълнителни услуги: Разширена функционалност на приложението чрез интегриране с популярните услуги за организиране на превози, хотелски резервации или услуги за туристически атракции. Като се предлага обслужване на едно гише за всички нужди при пътуване, може да се осигури допълнително удобство и да се създаде възможности за увеличаване на продажбите.
* Задълбочен анализ и отчитане за транспортни компании: Внедряване на усъвършенствани функции за анализ и отчитане, които помагат на транспортните компании да оптимизират своите операции, управление на автопарк и стратегии за ценообразуване. Предоставете информация за популярността на маршрута, тенденциите в пътниците и генерирането на приходи, за подпомагане на вземането на решения, базирани на данни.
* Функции за достъпност и приобщаване: Уверението, че приложението е достъпно за потребители с увреждания, чрез включването на функции като поддръжка на екранен четец, теми с висок контраст, регулируеми размери на шрифта и навигация с клавиатура демонстрират ангажимент за приобщаване и ще достигне до по-широка потребителска база.
* Многоезична поддръжка и локализация: Добавянето на поддръжка за множество езици, за да се покаже грижата за по-широка аудитория и да се подбере потребителското изживяване за хора, които не говорят български. Локализирането на съдържанието, интерфейса и поддръжката на клиентите на приложението го направят по-подходящо и удобно за потребители от различни региони на страната и света.

Тези бъдещи идеи имат за цел да подобрят обхвата и функционалността на системата за запазване на билети с идеята тя да предлага повече стойностна потребителите на системата и да се откроява на пазара сред останалите подобни услуги.

# Използвани литературни източници и Уеб сайтове

Bootstrap. (н.д.). *Докуменатция на Bootstrap*. Извлечено от bootstrap.com: https://getbootstrap.com/

Microsoft. (н.д.). *Документация на ASP.NET*. Извлечено от learn.microsoft.com: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0

Refsnes Data AS. (н.д.). *W3Schools*. Извлечено от W3Schools.com: https://www.w3schools.com/

StackOverflow. (н.д.). *StackOverflow*. Извлечено от stackoverflow.com: https://stackoverflow.com/

# Приложения

Фигура 13 Диаграма на класовете