**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

Факултет: **Природни науки и образование**

Катедра: **Информатика и информационни технологии**

**ДИПЛОМЕН ПРОЕКТ**

**Дипломант: Научен ръководител:**

Тихомир Василев гл. ас. д-р М. Андреева

**Фак. No** **126605**

**РУСЕ**

**2018 г.**

##### **Русенски университет “Ангел Кънчев”**

##### Факултет: Природни науки и образование

##### Катедра: Информатика и информационни технологии

## 

## ДОПУСКАМ ДО ЗАЩИТА

Ръководител на специали-

зиращата катедра:……..………

(доц. д-р Румен Русев)

#### ОБЯСНИТЕЛНА ЗАПИСКА

#### НА ДИПЛОМЕН ПРОЕКТ

**Тема:** Информационна система за обслужване на фирма за транспорт

### Дипломант: ………………… ОДОБРЯВАМ

(Тихомир Василев) Ръководител:.................................

(гл. ас. д-р. М. Андреева)

Русе, 2017 г.

**Русенски университет “Ангел Кънчев”**

Катедра **“Информатика и информационни технологии”**

## Регистрирал УТВЪРЖДАВАМ

### Факултет ПНО Р-л на специализи-

Инспектор: …………….. ращата катедра: …………

(М. Чолакова) (доц. д-р Р. Русев)

....../......, 20\_\_ г.

#### Задание за дипломен проект

На студент: **Тихомир Иванов Василев** Фак.№ 126605

Специалност: **Компютърни науки**

ОКС: **Бакалавър**

1.Тема: **Информационна система за обслужване на фирма за транспорт**

2. Изходни данни: XAMPP или друга среда за програмиране в Интернет, данни за транспортната фирма

3. Съдържание на обяснителната записка:

3.1. Увод

3.2. Преглед на съществуващи решения, цели и задачи

3.3. Описание на проекта на системата

3.4. Описание на програмните модули

3.5. Ръководство за работа със системата

3.6. Изводи и препоръки

Срок за предаване на дипломния проект: 26.01.2018 г.

Зададено на: 01.02.2017 г.

Дипломант:………………….. Ръководител: ......................

(Тихомир Василев) (гл. ас. д-р М. Андреева)

**Съдържание**

[1.Увод 7](#_Toc504678238)

[2.Обзор на съществуващите решения. Изводи. Цел и задачи 9](#_Toc504678239)

[2.1. Обзор на съществуващи решения. 9](#_Toc504678240)

[2.1.1. Trans.Eu 9](#_Toc504678241)

[2.1.2. my.tacho.bg 10](#_Toc504678242)

[2.1.3. Транспорт Плюс 11](#_Toc504678243)

[2.2. Анализ на разгледаните системи. 11](#_Toc504678244)

[2.3. Изводи. Цели и задачи. 12](#_Toc504678245)

[2.3.1 Изводи 12](#_Toc504678246)

[2.3.2 Цели и задачи 12](#_Toc504678247)

[3. Архитектурен и логически модел на системата 13](#_Toc504678248)

[3.1. Архитектурен модел на системата 13](#_Toc504678249)

[3.1.1. Използвани технологии 13](#_Toc504678250)

[3.1.2 Архитектурата на системата 17](#_Toc504678251)

[3.2. Логически модел на системата 18](#_Toc504678252)

[3.2.1. Диаграми на случаите на употреба 18](#_Toc504678253)

[3.2.2. Диаграми на дейностите – Activity Diagrams 22](#_Toc504678254)

[3.2.3. Диаграми на последователностите - Sequence diagram 27](#_Toc504678255)

[4. Описаниe на базата данни 29](#_Toc504678256)

[4.1. Релационен модел на базата данни 29](#_Toc504678257)

[4.1.1. Основни таблици и връзките между тях 29](#_Toc504678258)

[4.1.2. Създаване на връзки между таблиците в Laravel. 30](#_Toc504678259)

[4.2. Описание на таблиците 31](#_Toc504678260)

[4.2.1. Таблица users 31](#_Toc504678261)

[4.2.2. Таблица roles 32](#_Toc504678262)

[4.2.3. Таблица vehicles 32](#_Toc504678263)

[4.2.4. Таблица vehicle\_status 33](#_Toc504678264)

[4.2.5. Таблица vehicle\_types 33](#_Toc504678265)

[4.2.6. Таблица logvehicle 34](#_Toc504678266)

[4.2.7. Таблица vehicle\_repair 34](#_Toc504678267)

[4.2.8. Таблица vehicle\_reservation 35](#_Toc504678268)

[4.2.9. Таблица services 35](#_Toc504678269)

[4.2.10. Таблица orders 35](#_Toc504678270)

[4.2.11. Таблица order\_status 36](#_Toc504678271)

[5.1. Основни модули (Контролери). 37](#_Toc504678272)

[5.1.1. Модул за вписване в системата (LoginController) 37](#_Toc504678273)

[5.1.2. Модул за регистриране в системата (RegisterController) 39](#_Toc504678274)

[5.1.3. Модул за началната страница (HomeController) 40](#_Toc504678275)

[5.1.4. Мениджърски модул (ManagerController) 40](#_Toc504678276)

[5.1.5. Административен модул (AdminController) 41](#_Toc504678277)

[5.1.6. Модул за управление на шофьори (DriverController) 43](#_Toc504678278)

[5.1.7. Модул за управление на клиенти (CustomerController) 43](#_Toc504678279)

[5.2. Помощни модули за валидиране на входни данни 43](#_Toc504678280)

[5.3. Модули за връзка с базата данни (Модели) 44](#_Toc504678281)

[5.3.1. Модел за таблицата users (User) 45](#_Toc504678282)

[5.3.2. Модел за таблицата roles (Role) 45](#_Toc504678283)

[5.3.3. Модел за таблицата vehicles (Vehicles) 46](#_Toc504678284)

[5.3.4. Модел за таблицата orders (Order) 46](#_Toc504678285)

[5.3.5. Модел за таблицата services (Service) 47](#_Toc504678286)

[5.3.6. Модел на таблицата vehicle\_status (Status) 47](#_Toc504678287)

[5.3.7. Модел на таблицата logvehicle (LogVehicles) 47](#_Toc504678288)

[5.3.8. Модел на таблицата order\_status (OrderStatus) 48](#_Toc504678289)

[5.3.9. Модел на таблицата vehicle\_types(Vehicle\_types) 48](#_Toc504678290)

[5.3.10. Модел на таблицата vehicle\_repair (VehicleRepair) 48](#_Toc504678291)

[5.3.11. Модел на таблицата vehicle\_reservation (VehicleReservation) 48](#_Toc504678292)

[5.4. Използвани Middleware класове 49](#_Toc504678293)

[5.4.1. Middleware за проверка на достъп до административния панел (CheckRole) 49](#_Toc504678294)

[5.5. Основни изгледи 49](#_Toc504678295)

[5.5.1. Изгледи за неоторизирани потребители 50](#_Toc504678296)

[5.5.2. Изгледи за оторизирани потребители 50](#_Toc504678297)

[5.5.2.1. Основни изгледи за административния панел 51](#_Toc504678298)

[5.6. Тестване на приложението 52](#_Toc504678299)

[6.1. Инсталиране на системата 55](#_Toc504678300)

[6.1.1. Изисквания при инсталиране на системата 55](#_Toc504678301)

[6.1.2. Инсталиране на системата 55](#_Toc504678302)

[6.2. Достъп до административния панел 58](#_Toc504678303)

[6.3. Ръководство за потребител „Клиент” 59](#_Toc504678304)

[6.3.1. Регистрация в системата 59](#_Toc504678305)

[6.3.2. Вход в системата 60](#_Toc504678306)

[6.3.3. Основна страница след вписване в системата като клиент 61](#_Toc504678307)

[6.4. Ръководство за потребител „Мениджър” 65](#_Toc504678308)

[6.4.1. Основна страница след вписване в системата като мениджър 65](#_Toc504678309)

[6.4.2. Добавяне на ново превозно средство в системата. 65](#_Toc504678310)

[6.4.2. Управление на автопарка 68](#_Toc504678311)

[6.5. Ръководство за потребител „Шофьор” 77](#_Toc504678312)

[6.5.1. Основна страница след вписване в системата като клиент 77](#_Toc504678313)

[7. Основни резултати, изводи и препоръки 79](#_Toc504678314)

[8. Използвана литература 81](#_Toc504678315)

# 1.Увод

Исторически погледнато човечеството непрекъснато се стремяло към усъвършенстване и развитие. Този стремеж е довел до откриването на огъня, електричеството, компютрите и още стотици хиляди неща, без които животът ни днес би бил много по труден.

Бизнесът, като всеки продукт създаден от човек, също се стреми да се развива и усъвършенства. Когато говорим за бизнес, може да се обърнем към една стара поговорка: “Не можеш да управляваш онова, което не можеш да измериш”.

Независимо дали предметът на действие на една фирма е свързан със спедиторска или транспортна дейност, „рент-а-кар“ услуги, отдаване под наем на луксозни превозни средства или каквато и да е друга дейност, разполагаща с автомобилен парк, изниква необходимостта от използването информационна система, която да улесни, автоматизира и оптимизира управлението и експлоатацията на един такъв автопарк.

Внедряването на такава информационна система би довело от една страна до по-високо ниво на управление и контрол на бизнеса, от друга - възможност за постигане на максимална координация между отделните работни звена.

Когато става въпрос за успешното управление на автопарка, няма значение дали една организация разполага със 100 превозни средства, или със 100 000. Правилното управление трябва да включва водене на отчетност и съхраняване на ключови данни, отчетност за фирмените превозни средства и други мобилни активи и насрочване на интервали за осигуряване на оптимална поддръжка.

Години наред мениджърите на такива фирми са разчитали на ръчни пътни листове и хартиени формуляри и са подавали сервизни записи, за да направят възможно проследяването и управлението на операциите на автопарковете си. Днес съществуват технологии за усъвършенстване на тези процеси.

До колко внедряването на подобна информационна система е ефективно, може да се открие в доклад на Aberdeen Group от ноември 2010 г [1]. Той е посветен на решенията за закупуване на организации, които неотдавна са въвели подобни системи. В проучването са включени отговорите на 123 анкетирани представители на такива организации. Приблизително 70% от анкетираните са посочили, че главна причина за внедряването на такава система е желанието за подобряване на услугите, които предлагат на клиентите си. 55 на сто пък са отговорили, че сериозна причина за тяхното решение за имплементация е необходимостта от намаляване на разходите, а още толкова са споменали и нуждата за увеличаване на продуктивността и ефективността на услугите им. Малко над 40 процента пък са посочили, че е трябвало да увеличат приходите си от услуги.

Целта на настоящият дипломен проект е създаването на онлайн базирана информационна система за управление на фирма за автотранспорт, където:

* Потребителите (клиенти) ще могат да подават заявки за извършване на транспортни услуги.
* Мениджърите ще могат да организират целия технологичен процес включващ:
* управление на автопарка ;
* обработване на поръчките, получени от потребителите;
* избор на подходящо превозно средство за извършването на услугата;
* назначаване на шофьори.
* Шофьорите ще могат да виждат възложените им поръчки, датата и мястото на тръгване, ще дават обратна връзка към системата, когато започват изпълнението и приключват изпълнението на дадена поръчка.

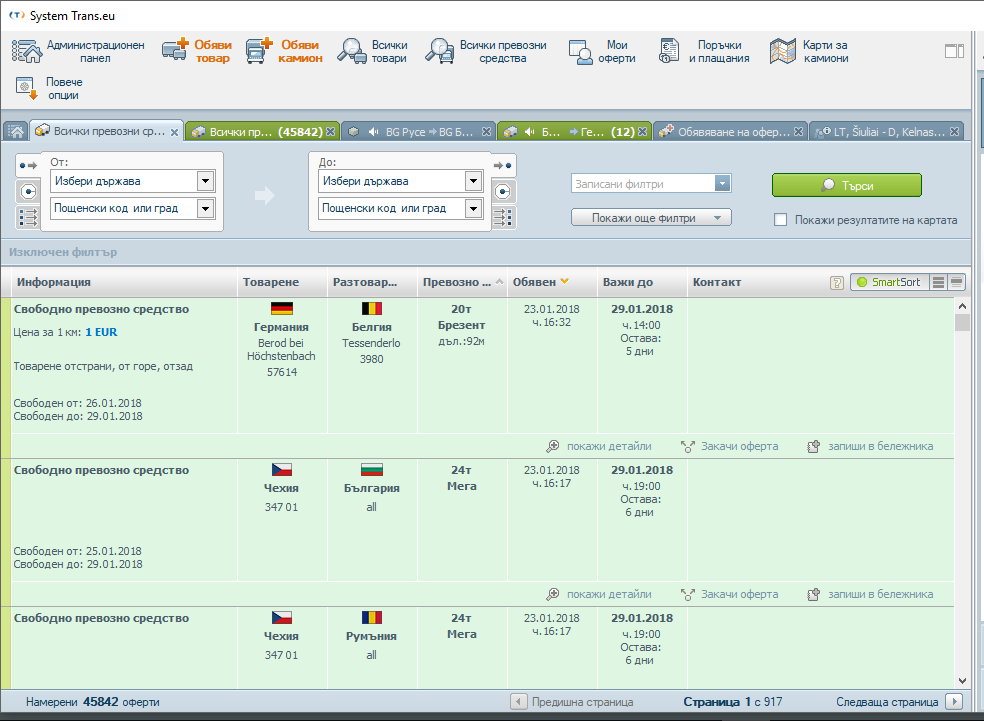
За постигане на тази цел, бе разработена интернет система, за три основни категории потребители – Клиент, Мениджър и Шофьор. Използвани са съвременни технологии за уеб приложения, а потребителският интерфейс е изчистен и интуитивен.

# 2.Обзор на съществуващите решения. Изводи. Цел и задачи

## 2.1. Обзор на съществуващи решения

### 2.1.1. Trans.Eu

Trans.eu е десктоп приложение, което има два основни модула, за фирми превозвачи и фирми клиенти [2]. Модулът за превозвачи дава възможност на различни логистични фирми да предлагат своите услуги. В този случай фирмата превозвач предлага превозни средства, с които разполага и на какви дестинации може да извършва транспортни услуги. Модулът за клиента дава възможност за подготовка и проследяване на изпратените пратки.



*Фигура 2.1 Trans.Eu*

**Предимства:**

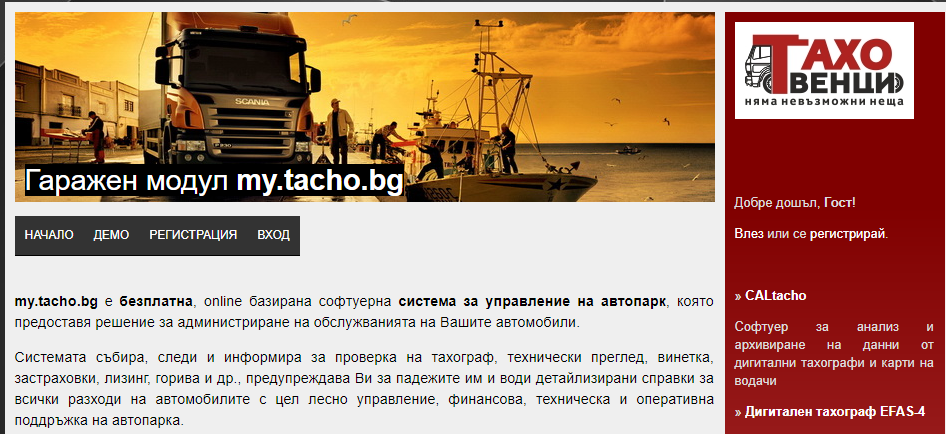
* Има възможност много фирми превозвачи да предлагат своите услуги;
* Клиентът може да избира между по-голямо разнообразие от оферти за транспорт.

**Недостатъци:**

* Тъй като това е десктоп приложение, трябва задължително да се инсталира на компютър;
* Регистрация могат да правят само фирми, т.е. няма липсва възможност физическо лице да ползва съответните услуги;
* Предоставя възможност само за превоз на големи товари.

### 2.1.2. my.tacho.bg

Това е безплатна, уеб базирана софтуерна система за управление на автопарк, която предоставя решение за администриране на обслужванията и поддръжка на автопарк [3].



*Фигура 2.2 my.tacho.bg*

**Предимства:**

* Безплатна;
* Позволява да се регистрират както фирми, така и физически лица.

**Недостатъци:**

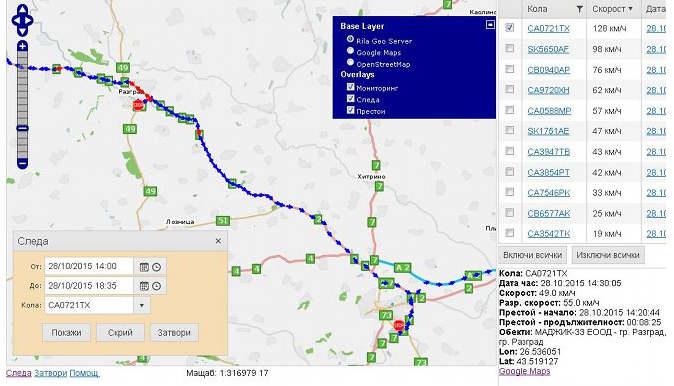
* Системата поддържа повече техническото състояние на автопарка. Обслужването на клиентите е много слабо;
* Използвани са много ярки цветове;
* Използван е шаблон за дизайн, който е неудобен и информацията е прекалено сбита.

### 2.1.3. Транспорт Плюс

Това е уеб базирана система, предназначена за фирми, които управляват собствен автомобилен парк, пътно-транспортна техника или машини. Софтуерът осигурява ефективно управление на дейностите в направленията куриерски услуги, разнос, управление на доставките, обслужване с мобилни екипи, механизация и автотранспорт. [4]

**Предимства:**

* Системата е с добри функционални възможности.
* Изграден е отделен модул за техническо обслужване на автомобилния парк;
* В системата е вграден модул за GPS мониторинг



*Фигура 2.3 Транс Плюс*

**Недостатъци**:

* Предлага регистрация само за фирми;
* Предлага обслужване само за големи товари.

## 2.2. Анализ на разгледаните системи.

Сравнение на функционалните възможности на разгледаните системи е показано в таблица 2.2. *Таблица 2.2 Сравнителна таблица*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Trans.Eu** | **my.tacho.bg** | **Транспорт Плюс** |
| Уеб базирана | НЕ | ДА | ДА |
| Позволява регистрация на физически лица | НЕ | ДА | НЕ |
| Превоз на малки и средни товари | НЕ | ДА | НЕ |
| Удобрен потребителски интерфейс | ДА | НЕ | ДА |

## 2.3. Изводи. Цели и задачи.

### 2.3.1 Изводи

От направения преглед на съществуващите решения, се виждат следните недостатъци:

* Не всички системи са уеб базирани;
* Куриерските услуги са с ограничен или фиксиран лимит;
* Почти всички системи са предвидени за автопаркове само за големи товари;
* Не всички системи имат модул за клиенти;
* Почти всички системи налагат ограничения при регистрация на потребители (само фирми).

### 2.3.2 Цели и задачи

Целта на настоящия дипломен проект е създаване на уеб базирана информационна система, за обслужване на фирма за автотранспорт.

За постигането на целта са формулирани следните задачи:

* Създаване на приложение, което да е платформено независимо;
* Възможност за регистрация като клиент на фирми и на физически лица;
* Потребителят с административни права да може да управлява останалите потребители;
* Създаване на модул за управление на автомобилния парк;
* Създаване на модул, с помощта на който клиентите да могат да се регистрират, да правят он-лайн заявки за извършване на услуги и да следят състоянието им;
* Създаване на модул за водачите на превозни средства в автопарка;
* Изграждане на удобен потребителски интерфейс.

# 3. Архитектурен и логически модел на системата

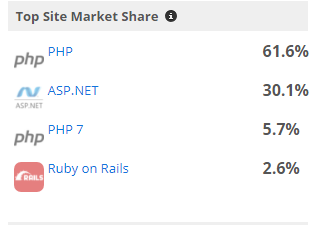
## 3.1. Архитектурен модел на системата

### 3.1.1. Използвани технологии

За реализацията на този проект са използвани следните технологии: PHP, MVC шаблон за разработване на софтуер-Laravel 5.4, MySQL, HTML5, CSS3, JavaScript, Jquery 1.8.3, GIT и Composer. Причините, за използване на всяка една от изброените технологии, са описани в следващите точки.

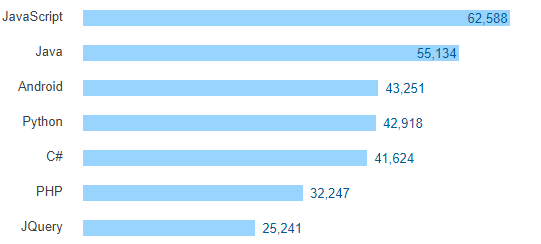
#### 3.1.1.1. Избор на програмен език

Съществуват много езици за програмиране, които се използват за създаване на уеб базирани приложения. Такива са C#, Java, PHP, Python, Perl, Ruby и други. Според данните на [https://trends.builtwith.com/framework/programming-language](https://trends.builtwith.com/framework/programming-language%20) от 01.01.2018г., показани на Фигура 3.1, става ясно, че най-използваният език за програмиране на уеб приложения в момента е PHP.



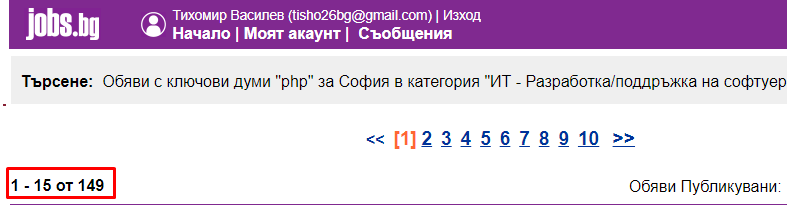
*Фигура 3.1 Най-използвани езици за програмиране на уеб приложения към 01.01.2018г*

На Фигура 3.2 е показана друга статистика от [stackoverflow.com](https://insights.stackoverflow.com/survey/2016#technology-top-tech-on-stack-overflow), която показва най-използваните технологии в сайта.



*Фигура 3.2 Най-използвани технологии според* [stackoverflow.com](https://insights.stackoverflow.com/survey/2016#technology-top-tech-on-stack-overflow) *за 2017г.*

Ситуацията в България може да бъде анализирана с резултат от търсенето в най-големия сайт за обяви за работа в България [jobs.bg](http://www.jobs.bg).На Фигура 3.3 е показана справка с ключова дума PHP.



*Фигура 3.3 Обяви за работа от* [*jobs.bg*](http://www.jobs.bg)

От тези данни можем да заключим, че PHP все още е един от най-използваните и най-търсените езици за програмиране. Точно този език за програмиране бе избран за разработването на този проект.

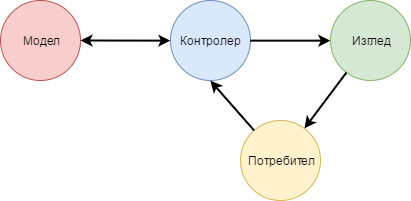
#### 3.1.1.2. MVC шаблон за програмиране

Разработената система използва MVC (Model – View – Controller) архитектурата за разработване на уеб базирани приложения. Това е шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение [5].

Самият шаблон се състои от 3 основни компонента:

* Модел (Model) – ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които сме моделирали и над които искаме да работим - да въвеждаме, променяме, визуализираме и т.н. Моделите служат за извличане на данни от базата данни
* Изглед (Vew) – тази част от изходния код на приложението е отговорна за показването на данните записани в модела. В изгледите няма бизнес логика, въпреки че може и да се постави малка логика за визуализиране на данните.
* Контролер (Controller) – тази част от сорс кода (клас или библиотека), взима данните от модела или извиква методи на модела, след което предварително обработва данните, и чак след това ги предава към изгледа. В контролера се поставя бизнес логиката на разработваната система.

На Фигура 3.4 е показана начина на взаимодействие между отделните компоненти на MVC шаблона.



*Фигура 3.4 Взаимодействие между отделните компоненти на MVC шаблона*

#### 3.1.1.3. Laravel 5.4 Framework – Рамка за създаване на уеб приложения

Laravel е софтуерна рамка за създаване на приложения, използващи MVC шаблона. Софтуерната рамка за програмиране (software framework) е структура на програмна система (софтуер), което облекчава разработката и обединението на различни компоненти на голям програмен проект. За разлика от библиотеките, които обединяват набор от подпрограми с близка функционалност, рамката за програмиране съдържа в себе си голямо количество различни по предназначение библиотеки [6].

#### 3.1.1.4. Система за управление на бази данни

MySQL е SQL система за управление на релационни бази данни (СУБД), подходяща за малки или средни приложения. Тази СУБД е безплатна и е с отворен код. Обикновено MySQL се използва при създаване уеб базирани приложения с програмните езици PHP и Java, които изискват място за съхранение на данните. MySQL има едно основно предимство пред останалите СУБД, а именно че е той безплатен. Обикновено се използва при споделен хостинг пакети и може лесно да бъде инсталирана на всички видове операционни системи.

#### 3.1.1.5. HTML5

HTML5 (HyperText Markup Language) е основният език за описание и дизайн на уеб страници, използван в проекта. Описанието на отделен документ става чрез специални елементи наречени HTML елементи или маркери, които се състоят от етикети или тагове и ъглови скоби, например <html>, <body>, <div>. 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, препратки към други страници и други [[7]](#HTML5).

#### 3.1.1.6. Cascading Style Sheets 3 (CSS3)

CSS е език за описание на стилове. Използва се основно за описване на представянето на документи, написани на език за маркиране като HTML. Той е създаден с цел да бъдат разделени структурата на уеб страниците отделно от тяхното визуално представяне. CSS 3 е най-новата версия на CSS. Тя поддържа нови стилове които позволяват заобляне на ъгли, придаване на сенки на елементи, градиенти, анимации и много други [8].

#### 3.1.1.7. JavaScript

JavaScript е език за програмиране, разпространяван с повечето Уеб браузъри. Поддържа обектно-ориентиран и функционален стил на програмиране. Най-често се прилага към HTML-а на интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове, както и за много други приложения [9].

#### 3.1.1.8. Jquery

jQuery е разпространена алтернатива на JavaScript, публикувана в началото на 2006 от Джон Резиг. В основата си, jQuery опростява достъпа до всеки елемент на дадена уеб-страница, като по този начин позволява лесно изграждане на динамична функционалност в страниците [10].

#### 3.1.1.9. Мениджър на пакети Composer

Composer e мениджър на пакети, често използван при разработката на PHP приложения. Той предоставя стандартен формат за управление на зависимости (dependencies). Composer работи с интерфейс с команден ред (CLI) и инсталира зависимости (например библиотеки) към дадено приложение. Той също позволява на потребителите да инсталират PHP приложения, които са достъпни в „Packagist“. „Packagist“ е основното хранилище на Composer, което съдържа наличните пакети. Composer предлага и възможности за автоматично зареждане (autoload) на библиотеки, като специфицира информация за автоматично зареждане с цел по-лесна употреба на код от трети страни [11].

Composer се използва като съществена част от няколко известни PHP проекти с отворен код, включително и в Laravel.

#### 3.1.1.10. Система за контрол на версиите Git

Система за контрол на версиите е механизмът, чрез който се управлява работата по даден софтуерен проект. За да се улесни разработката на софтуер са създадени специални системи, които намаляват неудобствата при съвместна работа на много хора върху един проект. При разработката на този проект бе използвана системата за контрол на версиите Git [12].

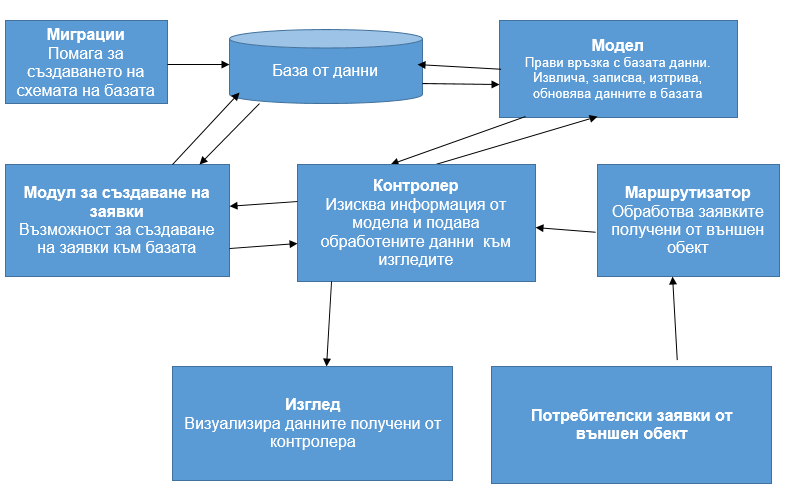
#### 3.1.1.11. Google Maps API

Google Карти е популярна картографска система, която предоставя географски карти на целия свят. Освен това в Google Крати са налични и подробни упътвания за придвижване от една точка до друга с кола, велосипед или пеша. Google Карти постоянно се допълва с клиентски отзиви за различни хотели, ресторанти, туристически обекти, събития, магазини и разнообразни бизнес предложения. Те могат да бъдат вградени в уебсайтове и са безплатни за ползване [13].

### 3.1.2 Архитектура на системата

На Фигура 3.5 е показана архитектурата на използваната софтуерна рамка Laravel. В нея може да се види ясно, как е реализиран MVC шаблона за разработка на уеб приложения. Отделните части от архитектура са:

* Маршрутизатор (Routing) – Всички заявки (POST, GET, PUT, DELETE...) към системата минават първо през маршрутизатора на Laravel. Той служи за координиране на заявките в системата.
* Контролер (Controller) – основната част от MVC шаблона и работи както с моделите, така и с изгледите в системата. По същество, за да се покаже съответната информация на потребителя, контролерът я изисква от модела и изпраща обработените данни към изгледа.
* Модел (Model) – друга основна част от MVC шаблона, която работи директно с базата данни като изисква, добавя, изтрива и обработва данните в нея.
* Изглед (View) – ролята на изгледите е да показват информацията, подадена им от контролера, на потребителя.
* Модул за създаване на заявки (Query Builder) – В Laravel е добавен модул за създаване на заявки към базата данни. Чрез него, контролерът изпълнявава заявки в базата данни директно, без да минава през модела. Този модул се използва за създаване на справки и отчети.
* Миграции (Migration) – За улеснение на работата на разработчика, в Laravel е добавен модул за създаване и редактиране на схемата на базата данни. Чрез него се създават таблици и се правят връзки между тях, чрез PHP код.



*Фигура 3.5 Архитектура на Laravel*

## 3.2. Логически модел на системата

Логическият модел на системата е представен чрез унифицирания език за програмиране UML, който дава възможност за визуализиране, специфициране, конструиране и документиране на елементите на една софтуерна система.

UML е развит и стандартизиран от Object Management Group (OMG) под ръководството на Ръмбаух, Бууч и Джакобсън. OMG е индустриален консорциум на водещи ИТ компании, вкл. Hewlett-Packard, IBM, Sun Microsystems, Apple Computer, American Airlines и Data General. Първата версия е обявена през 1997, а актуалната версия на езика е UML 2.5 [14].

### 3.2.1. Диаграми на случаите на употреба

Първата диаграма, която се създава в процеса на проектиране на софтуерен продукт е диаграмата на потребителските случаи. Чрез нея се описват на най-високо ниво целите на потребителя, които системата трябва да покрива. Тези цели могат да бъдат и общи изисквания към функционалността на системата. На практика това е техника за определяне функционалността на систмата, като се описва взаимодействието между потребителите и системата.

Диаграмата на случаите на употреба (use case) на разработената система е дадена на Фигура 3.6.

**

*Фигура 3.6 Случай на употреба от първо ниво*

Основните потребители на системата са четири – администратор, мениджър, шофьор и клиент.

**Администраторът** е потребител, който създава и управлява потребителите - мениджър, шофьор и клиент. Също така може да добавя, премахва и редактира предлаганите транстпорни услуги.

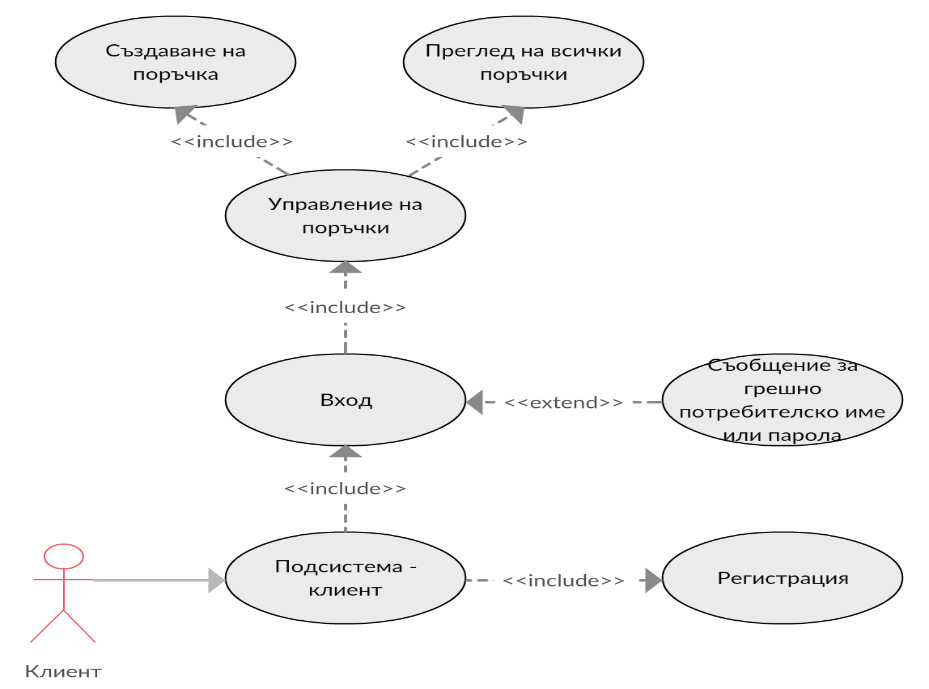
**Мениджърът** е потребител, който има права, свързани с управлението на автопарка и поръчките на клиентите.

Потребителите **Шофьор** и **Клиент** имат достъп само до ресурсите, свързани с техните роли.

В следващите диаграми са описани всички случаи на употреба, свързани с отделните потребители.

До подсистема „Клиент” (фиг. 3.7) имат достъп всички потребители на системата, регистрирани като потребител Клиент. Основните функции са следните:

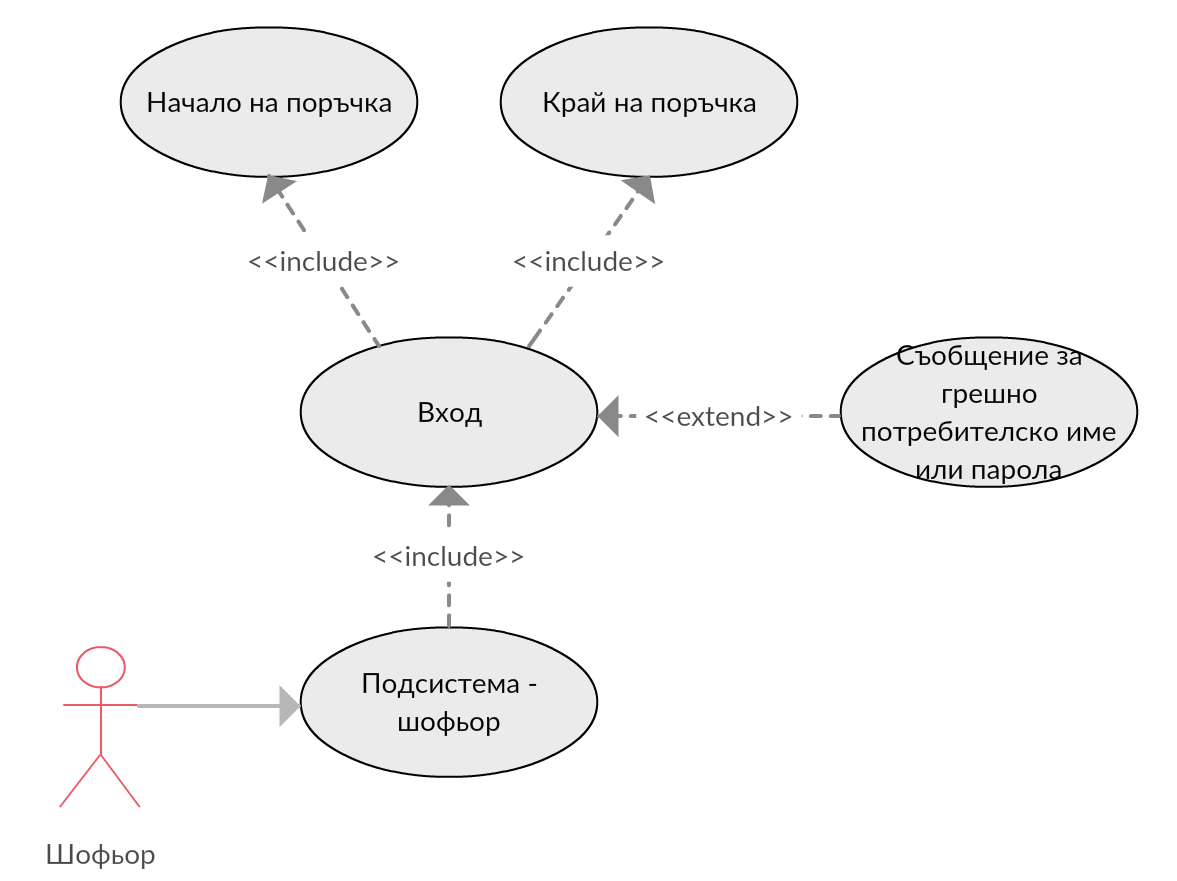
* Подаване на заявка за поръчка.
* Справка за направените поръчки.

****

*Фигура 3.7 Случай на употреба – Клиент*

На фигура 3.8 е показан случай на употреба на подсистема „Шофьор”. До нея имат достъп всички потребители в системата, регистрирани като потребител Шофьор. Основните функции са следните:

* Начало на поръчката – получаване на нова поръчка.
* Край на поръчката – приключване на поръчката.

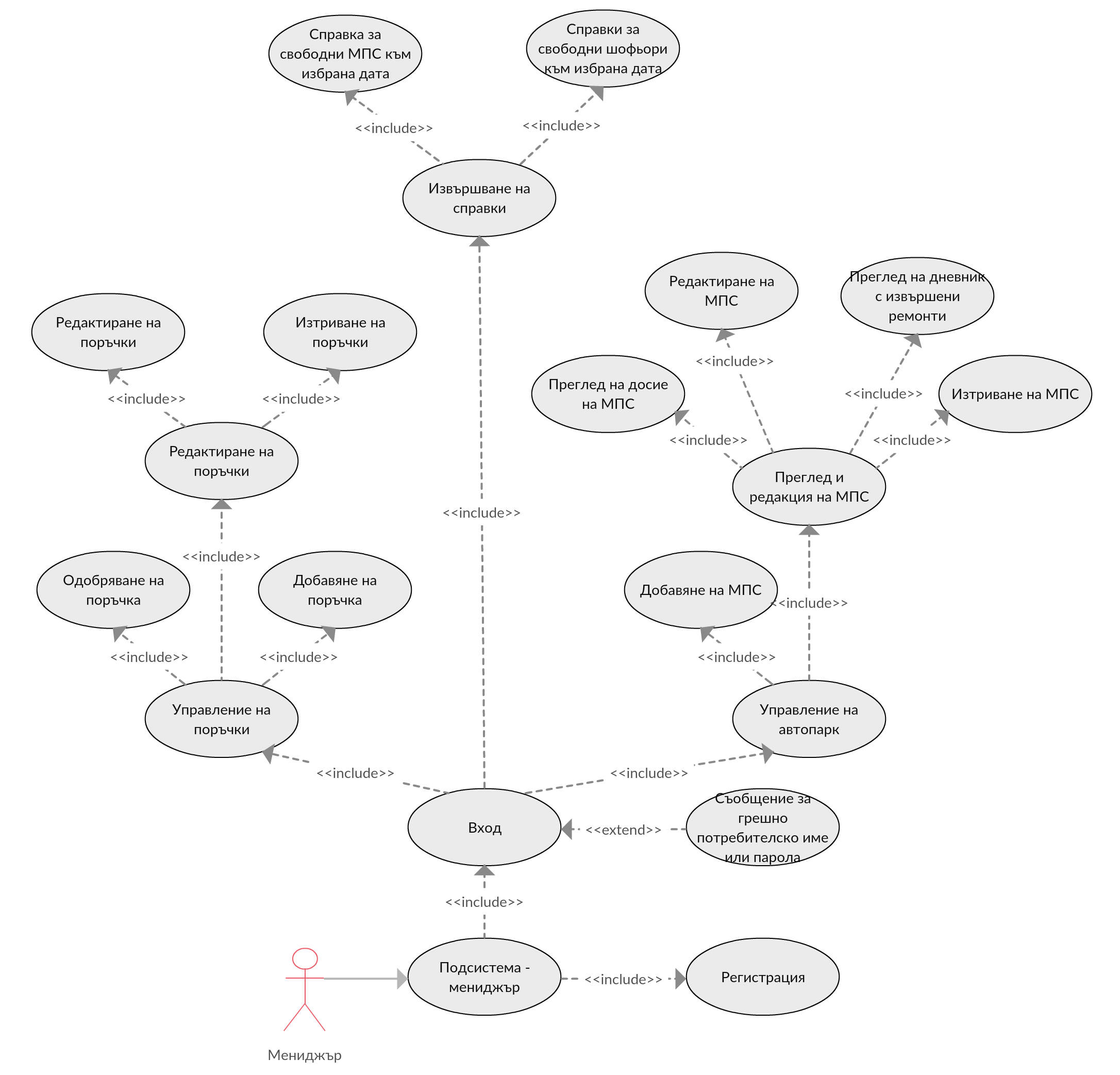


*Фигура 3.8 Случай на употреба – Шофьор*

На фигура 3.9 е показан случай на употреба на подсистема „Мениджър. До нея имат достъп всички потребители в система, регистрирани като потребител Мениджър.

В тази подсистема потребителите имат достъп до следните функции:

* Управление на автопарк
* Добавяне на ново превозно средство
* Редактиране на съществуващо превозно средство
* Преглед на състоянието на автопарка
* Управление на поръчки
* Създаване на нова поръчка
* Редактиране на поръчка
* Преглед на всички поръчки
* Приемане на заявка за транспортна услуга (поръчка) от клиент
* Справки
* Справка за свободни превозни средства за определена дата
* Справка за свободни шофьори за определена дата

****

*Фигура 3.9 Случай на употреба – Мениджър*

Потребител Администратор може да изпълнява ролята на всеки тип потребител – Клиент, Мениджър и Шофьор.

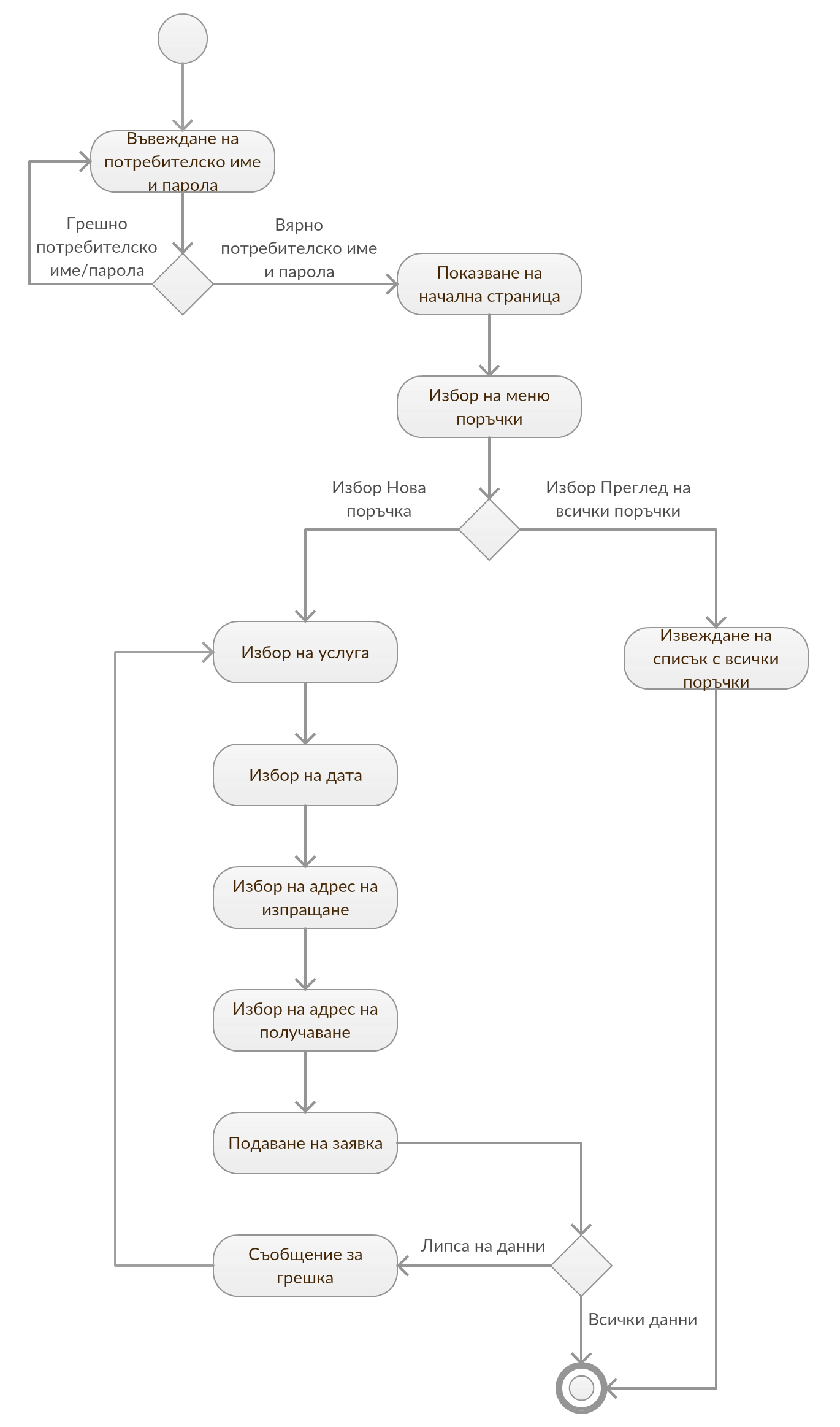
### 3.2.2. Диаграми на дейностите – Activity Diagrams

Диаграмата на дейности е техника за описване на процедурна логика, бизнес процес и работен поток. Чрез нея се показва логиката на поведение, като състоянията са самите дейности. Всяка дейност е изпълнение на определена задача, а това може да бъде физическо действие или изпълнение на код. Чрез тези диаграми се осигурява начин за моделиране на потока на бизнес процесите. Използват се за моделиране на специфичната информация при кодиране на операциите, изпълнявани от класове.

На фигура 3.10 е показан потока на действия в процеса на подаване на заявка за извършване за транспортна услуга от клиент. Първо се проверява дали потребителят има регистрация в системата. Ако това е така, се проверява дали потребителят има права за достъп до този ресурс. В случай, че не е така системата извежда съобщение за грешка. След като се влезе в системата, клиентът може да подаде заявка за извършване на транспортна услуга. За да бъде успешна заявката, трябва да се спази следната последователност от действия :

* Избор на вид услуга
* Избор на дата за извършване на услугата
* Избор на начален и краен адрес.

Всички параметри трябва да бъдат въведени. В противен случай системата ще генерира съобщение за грешка и няма да допусне подаването на заявката. При коректни данни, поръчката ще бъде изпратена и ще се генерира съобщение за успешно подаване на заявка.

****

*Фигура 3.10 Диаграма на дейностите – Подаване на заявка за извършване на транспортна услуга от клиент*

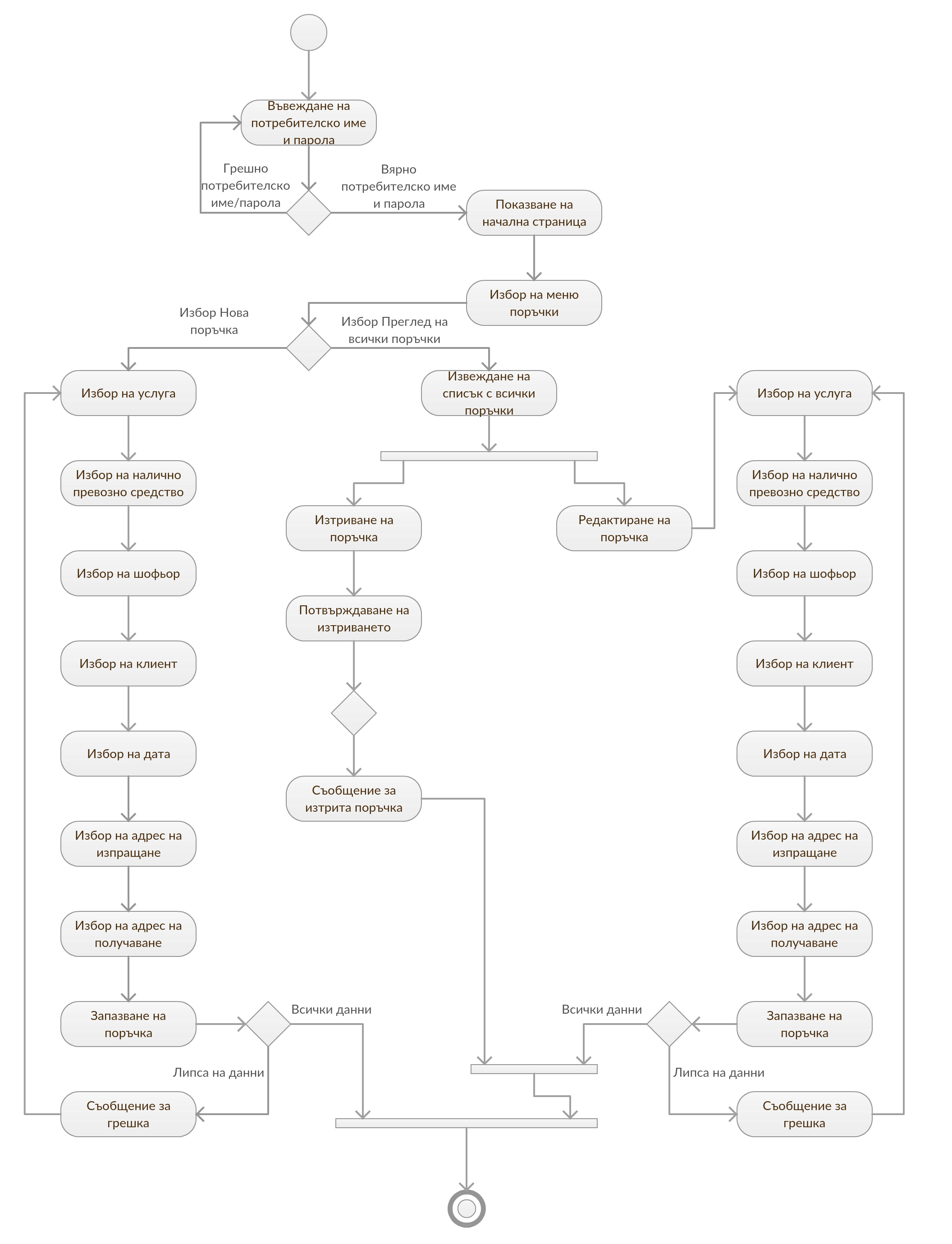
На фигура 3.11 е показан потока на действия в процеса на създаване на нова поръчка и редактиране на съществуваща такава от мениджъра на системата. И тук първо се проверява оторизацията на потребителя. За да бъде успешно създадена поръчката, мениджърът трябва да следва същите действия :

* Избор на вида на услугата
* Избор на превозно средство, с което ще извърши поръчката. В меню се показват само тези, които могат да извършват избрания вид услуга.
* Избор на шофьор, който ще извърши превоза. В меню се показват само тези, които могат да управляват съответния вид превозно средство.
* Избор на дата за извършване на услугата
* Избор на клиент, направил поръчката
* Избор на начален и краен адрес

Всички параметри трябва да бъдат въведени, иначе системата ще генерира съобщение за грешка и няма да допусне подаването на заявката. При коректни данни, поръчката ще бъде изпратена и ще се генерира съобщение за успешно подаване на заявка.

На фигура 3.12 е показан потока на действия в процеса на добавяне на ново превозно средство от мениджъра на системата. За да бъде успешно добавено, мениджърът изпълнява следната последователност от действия :

* Въвежда марката на превозното средство
* Въвежда регистрационния му номер
* Избира вида на двигателя
* Избира вид на превозното средство
* Задава статус
* Задава категорията, необходима на водача за управление на вида на превозното средство
* Задава разхода на гориво
* Задава изминатите километри, от както превозното средство е в сервиза. Понеже то е ново, тук се записват 0 километра
* Задава обема на полезния товар
* Задава датата на изтичане на гражданската му отговорност
* Задава датата на изтичане на техническия преглед

****

*Фигура 3.11 Диаграма на дейностите – Създаване, редактиране и изтриване на поръчки от мениджъра на системата*

****

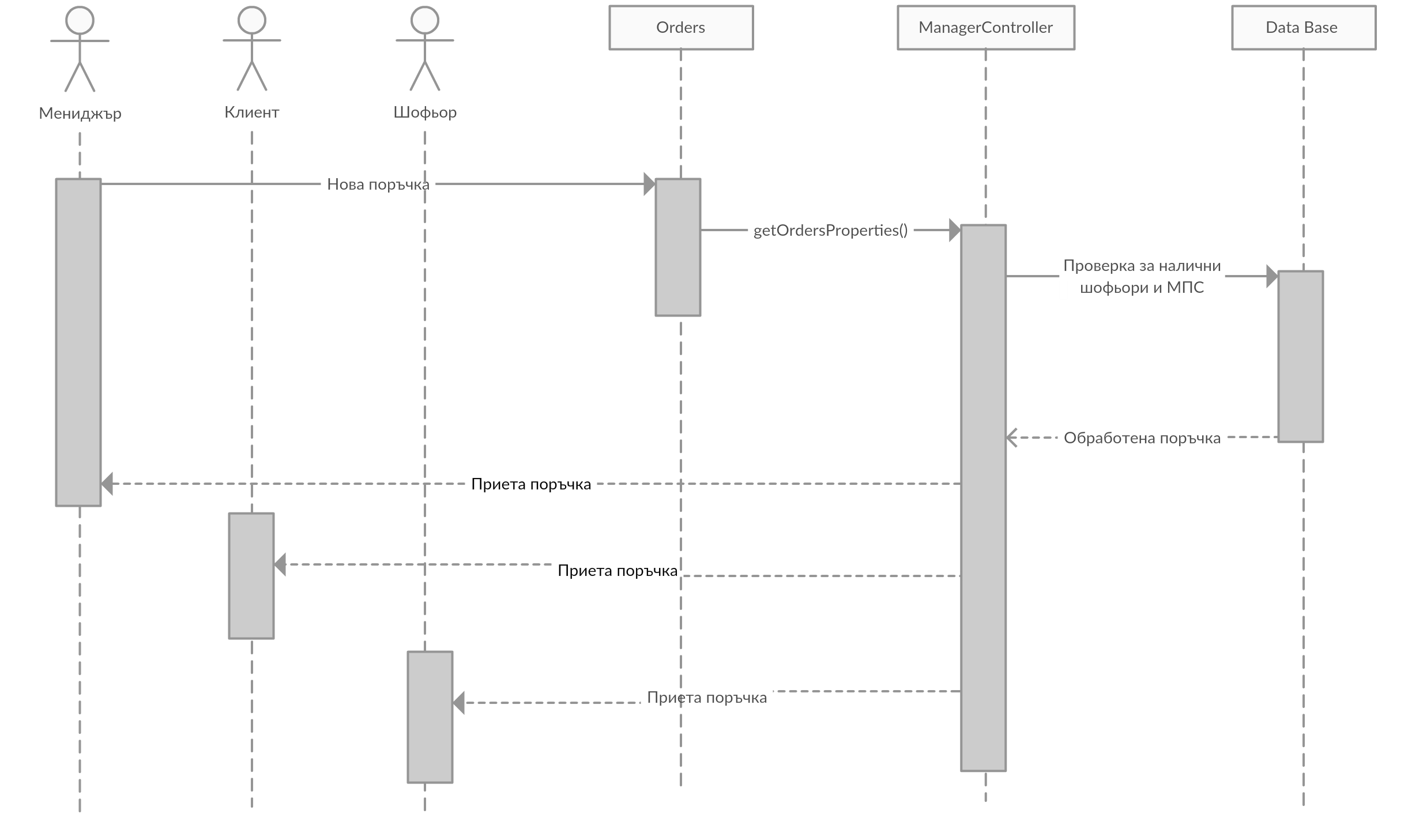
*Фигура 3.12 Диаграма на дейностите – Добавяне на ново превозно средство от мениджъра на системата*

Всички параметри трябва да бъдат въведени, иначе системата ще генерира съобщение за грешка и няма да допусне въвеждане на данните. При коректни данни, новото превозно средство ще бъде въведено в базата данни и ще се генерира съобщение за успех.

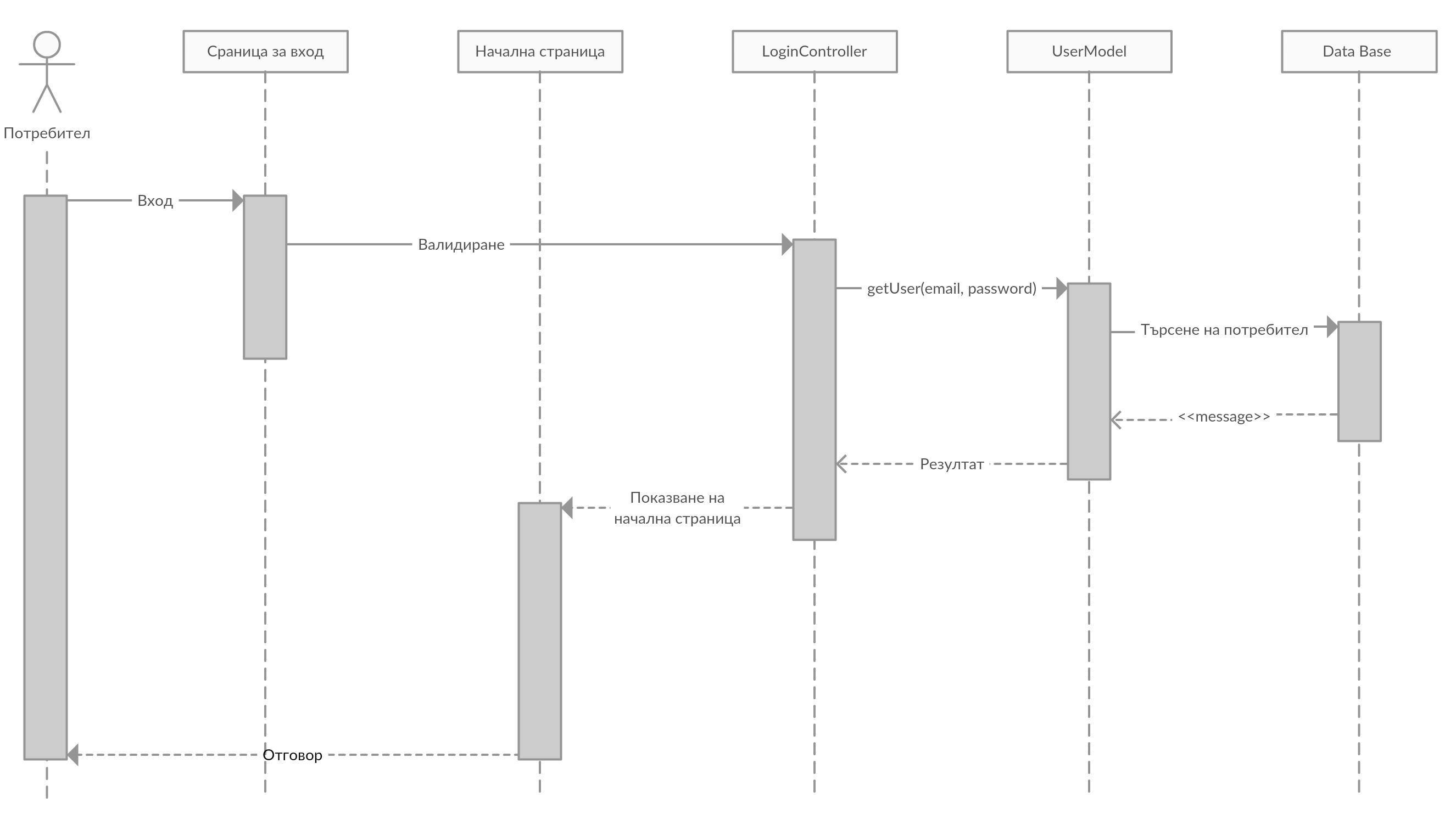
### 3.2.3. Диаграми на последователностите - Sequence diagram

Тези диаграми се използват, за да се проследи поведението на няколко обекта в рамките на един единствен случай на употреба. С помощта на този тип диаграми, съвместната работа между обектите се онагледява по-добре, като се описва реда, в който протичат процесите, като последователност от съобщения.

На фигура 3.13 и фигура 3.14 са показани съответно диаграмите на последователностите на създаване на поръчка от мениджъра системата и на вписване в системата.

****

*Фигура 3.13 Диаграма на последователностите на създаване на нова поръчка*

****

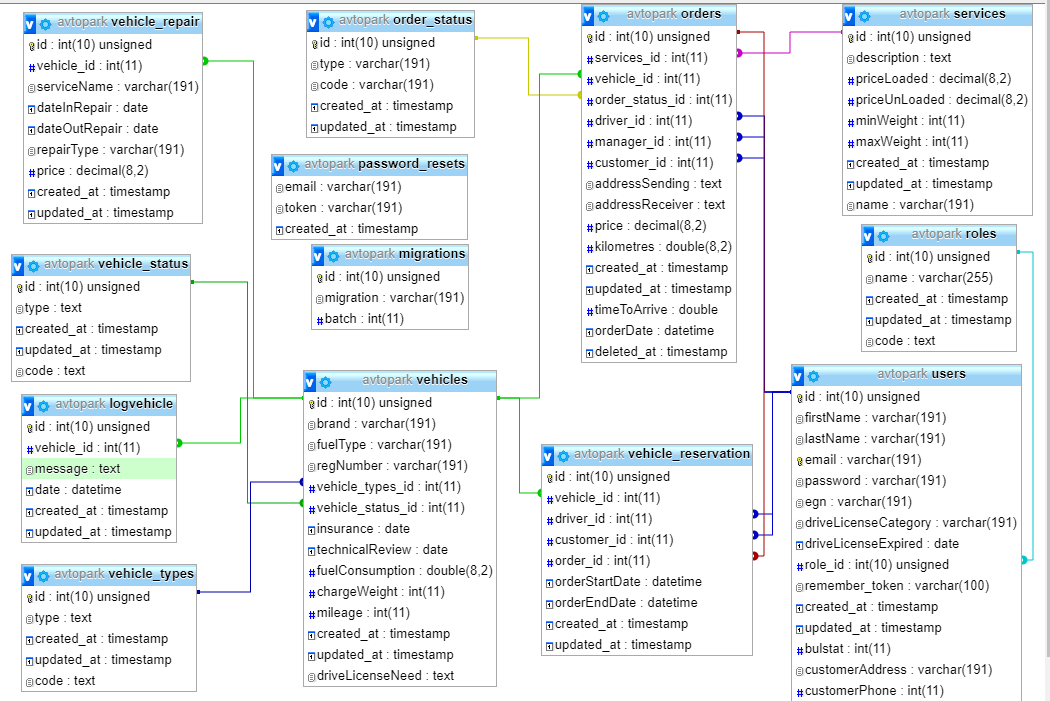
*Фигура 3.14 Диаграма на последователностите на вписване в системата*

# 4. Описаниe на базата данни

За реализацията на системата бе използвана релационна база от данни. Използвана е СУБД (RDBMS) MySQL с колация utf8\_unicode\_ci. Беше избрана точно тази колация, за да могат да се поддържат имена на роли и потребители на кирилица. Освен със символи на кирилица, колацията работи и с други нелатински символи.

## 4.1. Релационен модел на базата данни

На фигура 4.1 е показан релационния модел на проектираната база данни, с таблиците, техните атрибути и връзки.



*Фигура 4.1 Релационен модел на базата данни*

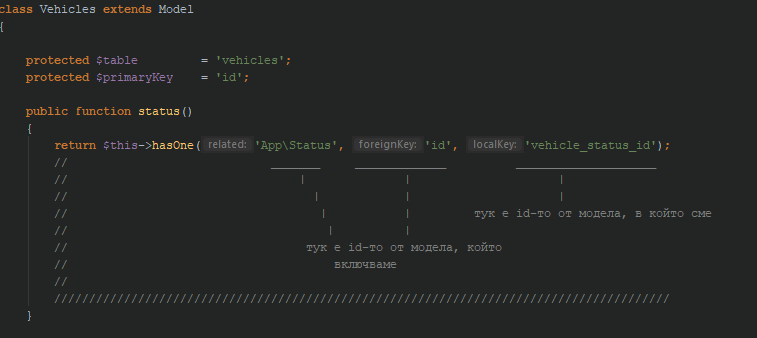
### 4.1.1. Основни таблици

* **migrations** – създадена от Laravel за следене на версиите на изпълнените миграции върху базата
* **users** – съдържа информация за потребителите
* **roles** – съдържа информация за ролите на потребителите
* **orders** – съдържа информация за поръчките
* **order\_status** – съдържа информация за статуса на поръчките
* **services** – съдържа информация за услугите
* **vehicles** – съдържа информация за превозните средства
* **logvehicle** – тук се записват всички настъпили събития в системата, изискващи използване на превозно средство
* **vehicle\_repair** – тук се записват всички случаи, когато едно превозно средство постъпи за ремонт
* **vehicle\_reservation** – съдържа информация за резервираните превозни средства
* **vehicle\_status** – съдържа информация за статуса на превозноте средства
* **vehicle\_types -** съдържа информация за вида на транспортните средства

### 4.1.2. Създаване на връзки между таблиците в Laravel.

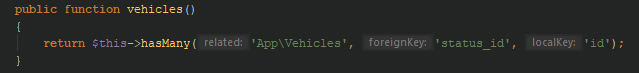
Създаването на връзка между две таблица се прави по следният начин :

* в модела, който обслужва дадена таблица, създаваме функция, в която се описва типа на връзката, таблицата с която искаме да направим връзка, локалния и външния ключ за таблицата. На фигура 4.2 е показана връзката между таблиците vehicles и vecle\_status. Намираме се в модела Vehicles, който обслужва таблицата vehicles. Функцията с която правим връзката с таблица vehicle\_status, обслужван от модела Status, е **public function status()**. Локалният ключ е **vehicles\_status\_id**, а външния ключ – **id**.



*Фигура 4.2 Изграждане на връзка от тип 1:М*

* В модела Status създаваме функция **public function vehicles().** Връзката с таблица vehicles e от тип много към едно (М:1) . Локалният ключ е **id**, а външния - **status\_id.**



*Фигура 4.3 Изграждане на връзка от тип М:1*

## 4.2. Описание на таблиците и връзките между тях

### 4.2.1. Таблица users

В таблицата users се съхраняват всички данни на потребителя, както и връзка към ролята, която притежава. Също така в тази таблица се записват данните за достъп на потребителя до системата: email и password.

Полетата в таблица users са:

* **id** – идентификатор;
* **firstName** – име на потребителя;
* **lastName** – фамилия на потребителя;
* **created\_at** – дата на създаване
* **email** – електронна поща, използва се при регистрацияя на потребителя;
* **password** – парола;
* **egn** – единен граждански номер;
* **driveLicenseCategory** – свиделство за правоуправление. Това поле е задължително само за потребителите с роля „Шофьор“
* **driveLicenseExpired** – дата на изтичане на шофьорската книжка;
* **role\_id** – използва се за връзка с таблица roles;
* **bulstat** – булстат. Задължително е само за потребители регистрирали се в системата като фирми;
* **customerAddress** – адрес на клиент;
* **customerPhone** – телефонен номер на клиент;
* **created\_at** – дата на създаване;
* **updated\_at** – дата на последна редакция.

Тази таблица има следните връзки:

* **public function role()** **-** връзка с таблицата roles. Връзката е от тип едно към едно (1:1). Предназначена за достъпване на информация за ролята на потребителя.
* **public function orders()** - връзка с таблица orders. Връзката е от тип връзката е от тип едно към много (1:М), защото един потребител може да направи много поръчки.

### 4.2.2. Таблица roles

Тази таблица се използва за запазване на основните роли, поддържани от системата и кратка информация за всяка от тях. Тя има връзка от тип 1:1 с таблицата users. Полетата в нея са:

* **id** – идентификатор;
* **code** – кратко обозначение на ролята. Използва се в кода при задаване на права;
* **name** – име на ролята;
* **created\_at** – дата на създаване;
* **updated\_at** – дата на последна редакция.

### 4.2.3. Таблица vehicles

В таблица vehicles се съхраняват всички данни за превозните средства в системата, както и връзки статуса и типа им. Нейните полетата са:

* **Id** – идентификатор;
* **brand** – марка на превозното средство;
* **fuelType** – тип на двигателя на превозното средство;
* **regNumber** – регистрационен номер на превозното средство;
* **vehicle\_types\_id** – връзка с таблица vehicle\_types;
* **vehicle\_status\_id** – връзка с таблица vehicle\_status;
* **Insurance** – дата на изтичането на гражданската отговорност;
* **technicalReview** – дата на изтичането на техническия преглед;
* **fuelConsumption** – разход на гориво;
* **chargeWeight** – полезен товар;
* **Mileage** – изминати километри от постъпването в автопарка;
* **created\_at** – дата на създаване;
* **updated\_at** – дата на последна редакция
* **driveLicenseNeed** – необходима шофьорска книжка;

Връзки в таблицата vehicles :

* **public function status()** - Връзка с таблица vehicle\_status, предназначена за достъпване на информация за статуса на превозното средство. Връзката е от тип едно към много (1:М).
* **public function vehicle\_types()** - Връзка с таблица vehicle\_type, предназначена за достъпване на информация за вида на превозното средство. Връзката е от тип едно към много (1:М).
* **public function vehicleRepair()** - Връзка с таблица vehicle\_repair, предназначена за достъпване на информация за ремонтите на превозните средства. Връзката е от тип едно към много (1:М).
* **public function logVehicles()** - Връзка с таблица logvecle, предназначена за достъпване на информация за събитията свързани с превозните средства. Връзката е от тип едно към много (1:М).
* **public function orders()** - Връзка с таблица orders. Връзката е от тип едно към много (1:М).

### 4.2.4. Таблица vehicle\_status

В таблицата vehicle\_status се съхраняват данните за статуса на превозните средства в системата. Полетата са:

* **id** – идентификатор
* **code** – кратко обозначение на статуса
* **type** – име на статуса
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

Връзки в таблицата vehicle\_status :

* **public function vehicles()** – Връзка, от тип много към едно (М:1), с таблица vehicles

### 4.2.5. Таблица vehicle\_types

В таблицата vehicle\_types се съхраняват данните за типа на превозните средства в системата. Полета:

* **id** – идентификатор
* **code** – кратко обозначение на типа на превозното средство
* **type** – тип на превозното средство
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

Връзки в таблицата vehicle\_types :

* **public function vehicles()** – Връзка, от тип много към едно (М:1), с таблица vehicles

### 4.2.6. Таблица logvehicle

В таблицата logvehicle се записват данните за превозно средство, след настъпило събитие довело до неговото използване, също така и самото събитие.

Полета в таблицата logvehicle :

* **id** - идентификатор
* **vehicle\_id** – външен ключ към таблицата vehicles
* **message** – съобщение за настъпилото събитие
* **date** – дата на настъпилото събитие
* **created\_at** – дата
* **updated\_at** – дата на последна редакция

Връзки в таблицата logvehicle :

* **public function vehicles()** – връзка от тип много към едно (М:1) с таблица vehicles

### 4.2.7. Таблица vehicle\_repair

В таблицата vehicle\_repair се записват данните за превозно средство когато се налага то да бъде ремонтирано. В тази таблица се записват датите на постъпване и на излизане от сервиза, в кой сервиз е извършен ремонта, какъв е бил ремонта или услугата и цената на самият ремонт.

Полета в таблицата vehicle\_repair :

* **id** - идентификатор
* **vehicle\_id** – външен ключ към таблицата vehicles
* **serviceName** – име на сервиза, в който постъпва превозното средство
* **dateInRepair** – дата влизане в сервиза
* **dateOutRepair** – дата на излизане от сервиза
* **repairType** – описание на ремонта или услугата
* **price**  - цена
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

Връзки в таблицата vehicle\_repair :

* **public function vehicles()** – връзка от тип много към едно (М:1) с таблица vehicles

### 4.2.8. Таблица vehicle\_reservation

В таблицата vehicle\_reservation се записват данните постъпили в системата след заявка за извършване на транспортна услуга.

Полета в таблицата vehicle\_reservation:

* **id** - идентификатор
* **vehicle\_id** – външен ключ към таблицата vehicles
* **driver\_id** – външен ключ към таблицата users
* **customer\_id** – външен ключ към таблицата users
* **order\_id** - външен ключ към таблицата orders
* **orderStartDate** – начало на поръчката
* **orderEndDate** – край на поръчката
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

Връзки в таблицата vehicle\_reservation :

* **public function order()** – връзка от тип много към едно (М:1) с таблица vehicles

### 4.2.9. Таблица services

В таблицата services се записват данните за предлаганите услуги.

Полета в таблицата services:

* **id** - идентификатор
* **description** – описание на услугата
* **price** – цена на услугата
* **minWeight** – минимално тегло на поръчка
* **maxWeight** – минимално тегло на поръчка
* **name** – наименование на услугата
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

### 4.2.10. Таблица orders

В таблицата orders се записват данните за всички поръчки.

Полета в таблицата orders:

* **id** - идентификатор
* **services\_id** – външен ключ към таблица services
* **vehicle\_id** – външен ключ към таблица vehicles
* **order\_status\_id** – външен ключ към таблица order\_status
* **driver\_id** – външен ключ към таблица users
* **manager\_id** – външен ключ към таблица users
* **customer\_id** – външен ключ към таблица users
* **addressSending** – адрес на изпращане
* **addressReceiver** – адрес на получаване
* **price** – цена
* **kilometres** – разстояние
* **timeToArrive** – време за пристигане
* **orderDate** – дата на поръчката
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

Връзки в таблица orders :

* **public function manager()** – Връзка едно към много (1:М) с таблица users
* **public function customer()** – Връзка едно към много (1:М) с таблица users
* **public function driver()** – Връзка едно към много (1:М) с таблица users
* **public function services()** – Връзка едно към много (1:М) с таблица services
* **public function vehicles()** – Връзка едно към много (1:М) с таблица vehicles
* **public function orderStatus()** – Връзка едно към много (1:М) с таблица order\_status
* **public function vehicleReservation()** – Връзка едно към много (1:М) с таблица vehicle\_resrvation

### 4.2.11. Таблица order\_status

В таблицата order\_status се съхраняват данните за статуса на превозните средства в системата. Полетата са:

* **id** – идентификатор
* **code** – кратко обозначение на статуса
* **type** – име на статуса
* **created\_at** – дата на създаване
* **updated\_at** – дата на последна редакция

**5. Програмна реализация**

Проектът се състои от осем модула, наречени контролери. Също така се използват модули предоставени от Laravel рамката. Голямото количество и разнообразие на входни данни, доведе до нуждата от създаване на помощни модули за валидация на данните постъпили от потребителите. В тази глава са описани всички модули. Системата използва и няколко допълнителни модула предоставени от Laravel. Такива са Middleware класове.

Middleware класовете в Laravel, се използват за изпълняване на дадена логика преди да се изпълнят или след изпълнението на логиката в контролерите. Например чрез Middleware класовете може да се проверява дали потребителя е вписан в системата преди да се изпълни контролера или да се проверява дали потребителя има достъп до дадена страница преди изпълнението на контролера.

Всички модули в системата се намират в /App/Http/Controllers.

Всички помощни модули за валидация се намират в /App/Http/Requsets.

Всики Middleware класове се намират в /App/Http/Middleware.

## 5.1. Основни модули (Контролери).

### 5.1.1. Модул за вписване в системата (LoginController)

Използва се за удостоверяване на самоличността на потребителите чрез имейл и парола. Контролерът използва AuthenticatesUsers Trait предоставен от Laravel, където се намира всичко необходимо за удостоверяването на самоличността на потребителя. Понеже AuthenticatesUsers Trait-a не може да се използва без да е включен в друг клас, бе създаден модулът LoginController. Също така в LoginController-а е добавена допълнителна логика за работата с конкретната система.

Намира се в /app/Http/Controllers/Auth.

* Методи предоставени от AuthenticatesUsers Trait:
  + **public function showLoginForm()** - използва се за извличане на изгледа за вписване в системата.
  + **public function login(Request $request)** – извиква се при опит за вписване в системата. Приема параметър $request от тип Request, който е предоставен от Laravel. Там се съдържат цялата информация от заявката направена от потребителя. В този метод се валидират въведените данни от потребителя, проверява се дали е достигнал лимита с грешни опити за вписване, прави се опит за вписване и ако се получи грешка се връща отговор с грешката, иначе потребителя се вписва успешно в системата.
  + **protected function validateLogin(Request $request)** - в този метод се извършва валидирането на потребителски данни изпратени чрез формата от изгледа за вписване. Приема параметър $request от тип Request. Връща грешка при грешни потребителски данни.
  + **protected function attemptLogin(Request $request)** - чрез този метод се извършва опит за вписване на потребителя в системата. Приема параметър $request от тип Request. Връща грешка при неуспешно вписване.
  + **protected function credentials(Request $request)** – използва се за извличане на нужните полета, които трябва да се попълнят за успешно вписване в системата.
  + **protected function sendLoginResponse(Request $request)** – използва се за създаване на отговор на заявката след успешно вписване в системата. Изпраща потребителя в началната страница след успешно вписване.
  + **protected function sendFailedLoginResponse(Request $request)** – използва се за създаване на отговор за грешка при вписване. Връща грешка за неуспешно вписване.
  + **public function username()** - използване за извличане на полето използвано за потребителско име в системата. В случая връща email.
  + **public function logout(Request $request)** – използва се при опит за излизане от системата. Изчиства сесията.
  + **protected function guard()** - използва се за извличане на начина на запазване на информацията за вписания потребител, дали се пази в сесията, в бисквитките или в базата от данни.
* Методи и променливи в LoginController модула
  + **protected $redirectTo** – използва се за указване на страницата за показване след успешно вписване в системата. Използва се в sendLoginResponse в AuthenticatesUsers Trait-а.
  + **public function \_\_construct()** - конструктор на класа, предназначен за указване на методите, които могат да се достъпват от неавтентикирани потребители.
  + **public function login(Request $request)** – използва се при опит за вписване в системата. Този метод е същия, който се предоставя от AuthenticatesUsers Trait-а, само че е доработен за целите на системата. Тук е добавена допълнителна проверка за това, дали потребителя е блокиран от администратора.
  + **protected function sendLockedAccountResponse(Request $request)** – използва се за създаване на отговор за грешка, при опит за вписване на блокиран потребител. Приема параметър $request от тип Request и връща отговор с грешка.

### 5.1.2. Модул за регистриране в системата (RegisterController)

Използва се при регистриране на нов потребител. Използва RegistersUsers Trait предоставен от Laravel.

* Методи в RegistersUsers Trait
  + **public function showRegistrationForm()** - използва се за извличане на изгледа за регистрация
  + **public function register(Request $request)** – използва се за валидиране на потребителските данни, регистриране на потребителя и вписване на потребителя в системата. Приема параметър $request от тип Request и при успешна регистрация изпраща потребителя в началната страница на системата
  + **protected function guard()** - използва се за извличане на начина на запазване на информацията за вписания потребител, дали се пази в сесията, в бисквитките или в базата от данни.
* Методи и променливи в RegisterController
  + **protected $redirectTo** – променлива указваща, коя е началната страница на системата. Използва се за изпращане на потребителя към началната страница след успешна регистрация
  + **public function \_\_construct()** - конструктор на класа, който се използва за указване на, това че всички методи са достъпни публично.
  + **protected function validator(array $data)** – използва се за валидиране на потребителските данни. При откриване на грешка, връща масив с откритите грешки и съобщенията към тях.
  + **protected function create(array $data)** - използва се за създаване на нов запис в базата от данни с изпратените данни от потребителя. Тук се задава пакета по подразбиране и ролята по подразбиране на потребителя. Също така се създава главна директория на профила, където ще се записват файловете и директориите му.

### 5.1.3. Модул за началната страница (HomeController)

Използва се за показване на началната страница на потребителите на системата.

Методи в HomeController модула:

* **public function \_\_construct()** - конструктор на клас, който служи за извикване на middleware класа Authenticate, предоставен от Laravel за проверка, дали потребителя е вписан в системата.
* **public function index(Request $request)** – проверява дали потребителят има валидна сесия, ако е така го препраща, към ресурса който отговаря за неговата роля като потребител.

### 5.1.4. Мениджърски модул (ManagerController)

До този модул имат достъп само мениджърите на системата. Използва се за управление на автопарка и обработване на поръчки.

Методи в ManagerController :

* **public function index()** - служи за показване на началната страница на мениджъра с информация за :
* автопарка - възможности за добавяне, редактиране и изтриване на превозно средство от автопарка.
* поръчки – добавяне и редактиране на поръчки
* справки за свободни превозни средства и шофьори
* **public function createVehicle()** – използва се за показване на страницата за добавяне на ново превозно средство в автопарка.
* **public function storeVehicle(VehicleFormRequest $request)** – използва се за съхраняване на данните на новодобавеното превозно средство. Приема параметър $request от тип VehicleFormRequest, който съдържа данните за новото превозно средство.
* **public function showVehicles()** – използва се за показване на всички налични превозни средства в автопарка.
* **public function editVehicle($id)** – използва се за редактиране на съществуващо превозно средство. Приема параметър $id, указващ кое е редактираното превозно средство.
* **public function deleteVehicle($id, Request $request)** – използва се за изтриване на съществуващо превозно средство. Приема два параметъра $id, указващ кое е изтриваното превозно средство и $request от тип Request.
* **public function saveVehicle($id, Request $request)** – използва се за съхраняване на данните на превозно средство, след редактирането му. Приема два параметъра $id, указващ кое е редактираното превозно средство и $request от тип Request.
* **public function vehicleProfile($id)** – използва се за показване на досието на превозно средство. Приема параметър $id.
* **public function vehicleRepairProfile($id)** – използва се за показване на досието с ремонти на превозно средство. Приема параметър $id.
* **public function showOrders()** – използва се за показване на всички поръчки, записани в базата данни.
* **public function createOrder()** – използва се за извикване на формата за създаване на нова поръчка.
* **public function storeOrder(OrdersFormRequest $request)** – използва се за съхраняване да данните за добавената поръчка. Приема параметър OrdersFormRequest $request,съдържащ данните за новата поръчка.
* **public function editOrder($id)** – използва се за :
* показване на формата за редактиране на съществуваща поръчка.
* показване на формата за добавяне на нова поръчка. Когато мениджърът избере някоя поръчка от менюто с известия, показано на фигура 4.43, на практика се извиква този метод.

Приема параметър $id, указващ коя е редактираната поръчка.

* **public function saveOrder($id, Request $request)** – използва се за :
* съхраняване на данните за поръчка, след редактирането й.
* запазване на нова поръчка.

Приема два параметъра $id, указващ коя поръчка се редактира или запазва и Request $request.

* **public function deleteOrder($id, Request $request)** – използва се за изтриване на поръчка. Приема два параметъра $id, указващ коя поръчка се изтрива и Request $request.
* **public function newOrderFromCustomer()** – използва се за визуализиране на новите поръчки направени от клиентите.

### 5.1.5. Административен модул (AdminController)

Административният модул, до който има достъп само администратора на системата, се използва за управление на потребителите. Само администраторът на системата може да добавя, редактира или изтрива потребител от системата.

Методи в AdminController :

* **public function index()** – използва се за показване на началната страница на администратора.
* **public function users()** – използва се за показване на списък с всички потребители в системата.
* **public function createUser()** – използва се за показване на формата за добавяне на нов потребител.
* **public function storeUser(ShareFormRequest $request)** – използва се за съхраняване на данните за добавените потребители. Използва параметър $request от тип ShareFormRequest., който съдържа даннитеза новите потребители.
* **public function deleteUser($id, Request $request)** – използва се за изтриване на съществуващ потребител. Приема два параметъра : $id, указващ изтривания потребител и Request $request.
* **public function editUser($id)** – използва се за показване на формата за редактиране на съществуващ потребител. Приема параметър $id, указващ редактираният потребител.
* **public function saveUser($id, ShareFormRequest $request)** – служи за записване на информацията след редактиране на потребител. Приема като параметри $id, указващ редактираният потребител и ShareFormRequest $request, съдържащ информацията за потребителите.
* **public function showService()** – служи за показване на всички услуги, предлагани от системата.
* **public function createService()** – служи за показване на формата, за добавяне на нова услуга.
* **public function storeService(StoreService $request)** – служи за съхранение на данните за добавените услуги. Приема като параметър StoreService $request.
* **public function editService($id)** – използва се за показване на формата за редактиране на услуга. Приема параметър $id, указващ редактираната услуга.
* **public function saveService($id, StoreService $request)** – използва се за съхранение на данните за услуги след редактирането им. Приема два параметъра : $id, указващ редактираната услуга и StoreService $request.
* **public function deleteService($id, Request $request)** – служи за истриване на съществуваща услуга.

### 5.1.6. Модул за управление на шофьори (DriverController)

До модула за управление на шофьори имат достъп само потребителите, регистрирани като шофьори в системата.

Методи в DriverController :

* **public function index()** – зарежда началната страница на шофьора и показва всички поръчки адресирани до него.
* **public function startOrder(Request $request)** – шофьорът потвърждава, че започнал изпълнението на поръчка.
* **public function endOrder(Request $request)** – шофьорът потвърждава, че приключил с изпълнението на поръчка.

### 5.1.7. Модул за управление на клиенти (CustomerController)

До клиентския модул имат достъп само потребителите, регистрирани като клиенти в системата.

Методи в CustomerController :

* **public function index()**  - зарежда началната страница на клиента.
* **public function createOrder()** – показва формата за подаване на заявка за трянспортна услуга.
* **public function storeOrder(CustomerCreateOrder $request)** – подаване на заявка за транспортна услуга.
* **public function showCustomerOrders()** – показва всички поръчки направени от клиента.

## 5.2. Помощни модули за валидиране на входни данни

Всички входни данни постъпващи от потребител към системата, трябва да бъдат проверявани за коректност, за да се изключи възможността на потребителя волно или не да повреди данните в системата. Това се прави с помощта на Request класове.

Това са най-обикновенни php - класове, които обаче наследяват един клас в Laravel, **FormRequest**. Този клас от своя страна наследява базовия клас **Request,** и имплементира интерфейса **ValidatesWhenResolved**.

За да създадем свой собствен клас за валидиране на данни, с помощта на конзолата, в папката в която е нашият проект трябва да въведем командата : **php artisan make:request VehicleFormRequest**. Всички Request файлове се записват в папката /App/Http/Requests .



Във файла, който създадохме има три метода :

* **public function authorize()** – за да укажем на Laravel, че този файл трябва да се използва при валидиране на данни, тази функция трябва да връща резултат ***true.***
* **public function rules()** – в този метод се описват, всички полета в една форма, от която очакваме да получим данни и условията, на които те трябва да отговарят.
* **public function messages()** – тук се описват всички съобщения за грешки, които ще се появят, при не коректно попълване на някое от полетата във формата.

Помощните модули за валидиране да входни данни са :

* **RegistrationFormRequest** – валидира данните постъпили от формата за регистрация на потребители.
* **OrdersFormRequest** – валидира данните постъпили от формата за създаване на нова поръчка, от мениджъра на системата.
* **CustomerCreateOrder** – валидира данните постъпили от формата за подаване на заявка за транспортна услуга от клиент.
* **ShareFormRequest** – валидира данните постъпили от формата за добавяне на нови потребители от администратора на системата.
* **VehicleFormRequest** – валидира данните постъпили от формата за добавяне на ново превозно средство.
* **StoreService** – валидира данните постъпили от формата за добавяне на нов вид услуга, от администратора на системата.

## 5.3. Модули за връзка с базата данни (Модели)

Модулите за връзка с базата данни представляват класове и са моделите от MVC шаблона за разработка на софтуерни приложения. Те служат за описване на таблиците от базата данни като обекти и помагат при работа със съответната таблица. Те се намират в /app/ директорията.

### 5.3.1. Модел за таблицата users (User)

При работа с бази данни, Laravel очаква от софтуерният разработчик да спазва определена конвенция. Името на таблицата в базата данни трябва да е в множествено число и да започва с малка буква, а името на модела да бъде в единствено число и да започва с главна буква. Разбира се тази конвенция не винаги може да се спазва. Много фирми имат изградена своя собствена конвенция за именоване на таблици и класове. За да се избегне този проблем, в модела трябва да добавим две променливи :

* **protected $primaryKey** – използва се за указване на първичния ключ на таблицата. Може да се пропусне, ако колоната се казва „id“.
* **protected $table** – указва коя таблица от базата данни използва този модел.

Другите методи и променливи в класа User са :

* **protected $fillable** – масив от низове, който указва кои полета от таблицата могат да се попълват.
* **protected $hidden** – масив от низове, който указва кои полета от таблицата трябва да се скрият при показване на информацията за съответния ред.
* **public function role()** - използва се за описване на връзката между users и roles таблиците. В случая тя е 1:1.
* **public function orders()** – използва се за създаване на връзка едно към много (1:М) с таблицата **orders**.
* **public function hasRole($roles)** – използва се за проверяване на това, дали потребителя има връзка с подадената роля като параметър.
* **private function getUserRole()** - използва се за извличане на ролята на потребителя.
* **private function checkIfUserHasRole($need\_role)** – метод, който се използва от hasRole($roles), метода при проверка, дали потребителя притежава дадена роля.
* **public function getDriverNewOrders()** – метод с който се показват новите поръчки за шофьорите.
* **public function getCustomerNewOrders()**  - метод, с който се показват новите поръчки направени от клиент.
* **public function store(Request $request)** – метод за добавяне на нов потребител.

### 5.3.2. Модел за таблицата roles (Role)

В този проект ще спазваме наложената от Laravel конвенция, за именуване на таблиците и моделите, но като още един вид застраховка, във всеки модел ще добавяме полетата **protected $table** и **protected $primaryKey**.

Моделът Role се използва за описване и обработка на таблица roles. В него има един метод : **public function users**, с помощта на който се прави връзка много към едно (М:1) с таблица users.

### 5.3.3. Модел за таблицата vehicles (Vehicles)

Моделът Vehicle се използва за описаниe и обработка на таблицата vehicles. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function status()** – използва се за създаване на връзка едно към много (1:М) с таблица vehicle\_status.
* **public function vehicle\_types()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicle\_types.
* **public function vehicleRepair()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicle\_repair.
* **public function logVehicles()** - използва се за създаване на връзка едно към много (1:М) с таблица logvehicle.
* **public function orders()** - използва се за създаване на връзка едно към много (1:М) с таблица orders.
* **public function create(Request $request)** – използва се за добавяне на ново превозно средство.

### 5.3.4. Модел за таблицата orders (Order)

Моделът Order се използва за описаниe и обработка на таблицата orders. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function manager()** - използва се за създаване на връзка едно към много (1:М) с таблица users.
* **public function customer()**  - използва се за създаване на връзка едно към много (1:М) с таблица users.
* **public function driver()**  - използва се за създаване на връзка едно към много (1:М) с таблица users.
* **public function services()** - използва се за създаване на връзка едно към много (1:М) с таблица services.
* **public function vehicles()**  - използва се за създаване на връзка едно към много (1:М) с таблица vehicles.
* **public function orderStatus()** - използва се за създаване на връзка едно към много (1:М) с таблица order\_status.
* **public function vehicleReservation()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicle\_reservation.
* **public function create(Request $request)** – използва се за създаване на нова поръчка от мениджъра на системата.
* **public function createForCustomer(Request $request)** – използва се за подаване на заявка от клиент.

### 5.3.5. Модел за таблицата services (Service)

Моделът Service се използва за описаниe и обработка на таблицата services. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function orders()** - използва се за създаване на връзка едно към много (1:М) с таблица orders.
* **public function create(Request $request) –** използва се за добавяне на нова услуга

### 5.3.6. Модел на таблицата vehicle\_status (Status)

Моделът Status се използва за описаниe и обработка на таблицата vehicle\_status. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function vehicles()**  - използва се за създаване на връзка едно към много (1:М) с таблица vehicles.

### 5.3.7. Модел на таблицата logvehicle (LogVehicles)

Моделът LogVehicles се използва за описаниe и обработка на таблицата logvehicle. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function vehicles()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicles.
* **public function create($vehicleId, $message)** – при настъпване на някакво събитие свързано с конкретно превозно средство, се създава обект от тип LogVehicle. Приема два параметъра :
* $vehicleId, указващ кое е превозното средство.
* $message – съобщение за самото събитие.

### 5.3.8. Модел на таблицата order\_status (OrderStatus)

Моделът OrderStatus се използва за описаниe и обработка на таблицата order\_status. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function orders()** - използва се за създаване на връзка едно към много (1:М) с таблица orders.

### 5.3.9. Модел на таблицата vehicle\_types(Vehicle\_types)

Моделът Vehicle\_types се използва за описаниe и обработка на таблицата vehicle\_types. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function vehicles()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicles.

### 5.3.10. Модел на таблицата vehicle\_repair (VehicleRepair)

Моделът VehicleRepair се използва за описаниe и обработка на таблицата vehicle\_repair. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function vehicles()** - използва се за създаване на връзка едно към много (1:М) с таблица vehicles.
* **public function create(Request $request)**

### 5.3.11. Модел на таблицата vehicle\_reservation (VehicleReservation)

Моделът VehicleReservation се използва за описаниe и обработка на таблицата vehicle\_reservation. Методите и променливите в него са :

* **protected $table**
* **protected $primaryKey**
* **public function order()** - използва се за създаване на връзка едно към много (1:М) с таблица orders.

## 5.4. Използвани Middleware класове

Middleware класовете се използват за изпълняване на дадена логика преди или след изпълнението на контролера. С тяхна помощ е реализирана проверката за достъп до административния панел.

Те се намират в /app/Http/Middleware

### 5.4.1. Middleware за проверка на достъп до административния панел (CheckRole)

Извиква се преди изпълнението на административните контролери, с цел проверка дали потребителя има достъп до тях.

Методи в CheckRole Middleware:

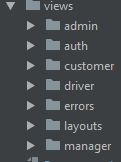
* **public function handle($request, Closure $next)** – използва се за проверяване на достъпа до даден метод от заявката. Параметърът $request съдържа информация за направената заявка, а с помощта на $next се изпълнява пуснатата заявка от потребителя (изпълнява се метода от съответния контролер, изискан от заявката).
* **private function getRequiredRoleForRoute($route)** – използва се за извличане на нужните права за съответната заявка.

## 5.5. Основни изгледи

Файловете, които се използват за създаването на потребителския интерфейс, наречени изгледи, се намират в **resources/views** директорията на проекта. За създаването на изгледите бе използван вграденият в Laravel, генератор на HTML код – **Blade template engine**.С помощта на този генератор много лесно може да се внедрява PHP код вътре в HTML. За да се използва генератора всички изгледи трябва да са с **.blade.php** разширение.

В папката **layouts** се намира основният шаблон app.blade.php, който се използва от изгледите за автентикирани и неавтентикирани потребители. В него се намират основните части от системата, като мета тагове на сайта, навигацията на сайта, също така и външните ресурси като CSS и JavaScript, които се използват от всички страници в системата. При генериране на изходния HTML, който ще се подаде като отговор на браузъра, всички изгледи необходими за визуализирането на страницата се включват в този основен шаблон.

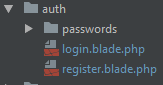
Основните директории на изгледите са показани на фигура 5.1.



*Фигура 5.1 Основни изгледи*

### 5.5.1. Изгледи за неоторизирани потребители

Изгледите за потребители, които не са се вписали в системата се намират в директорията **auth** (фиг 5.2).



*Фигура 5.2 Изгледи за неоторизирани потребители*

Изгледите за неоторизирани потребители са:

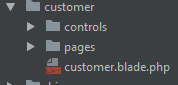
* **register.blade.php** – използва се за показване на страницата за регистрация на нови потребители.
* **login.blade.php** – използва се за показване на страницата за вписване в системата.

### 5.5.2. Изгледи за оторизирани потребители

Изгледите за оторизирани потребители са разделени в отделни папки според ролята на потребителя. Това са папките **admin**, **manager**, **driver** и **customer.**

Отделните блокове от изгледите, които се повтарят във всяка страница, наречени контроли, са отделени в директорията **controls.** Тези контроли се включват в основните изгледи. По този начин, ако трябва да се промени част от дадена контрола, която се използва от няколко изгледа, то се прави на едно място. Такива контроли са падащите менюта за добавяне на превозни средства, създаване на поръчки, създаване на потребители, основната навигация на системата и други.

Изгледите, които се използват за показване на страниците за потребителските данни и редактирането им също са отделени в отделна директория, наречена **pages.** По този начин разработчикът лесно може да се ориентира в структурата на кода (фиг 5.3).



*Фигура 5.3 Структура на изгледите за оторизирани потребители*

### 5.5.2.1. Основни изгледи за административния панел

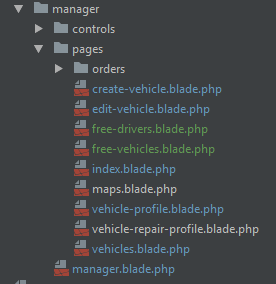
При създаването на административния панел, бе създаден нов основен шаблон **admin.blade.php.** Тук се включват всички ресурси нужни за работата на административния панел.

Основният изглед след вход в административния панел е **index.blade.php.** В него се съдържа само текст, показващ на потребителя, че се намира в административния панел.

#### 5.5.2.2. Основни изгледи за мениджърския панел

При създаването на мениджърския панел, бе създаден нов основен шаблон **manager.blade.php.** Тук се включват всички ресурси нужни за работата на мениджърския панел. Също така бе добавена допълнителна папка **orders**, в която се намират само тези изгледи свързани с поръчките. Структурата на изгледите за мениджърския панел може да се види на фигура 5.4.

Основният изглед след вход в мениджърския панел е **index.blade.php.** В него се съдържат хипер връзки към основните ресурси на системата.



*Фигура 5.4 Структура на изгледите за мениджърския панел*

Изгледите използвани в мениджърския панел са :

* **index.blade.php** -използва се за показване на основната страница с потребителските данни.
* **create-vehicle.blade.php** – използва се за показване на формата за добавяне на ново превозно средство в автопарка.
* **edit-vehicle.blade.php** – използва се за редактиране на превозно средство.
* **vehicle-profile.blade.php** – използва се за показване на досието на избрано превозно средство.
* **vehicle-repair.blade.php**  - използва се за показване на досието с ремонти на избрано превозно средство.
* **vehicles.blade.php** – показва състоянието на автопарка, към настоящата дата.
* **free-vehicles.blade.php** – справка за показване на всички превозни средства, към избрана дата.
* **free-drivers.blade.php** – справка за показване на свободните шофьори, към избрана дата.

В папката **orders**  се намират всички изгледи свързани с поръчките :

* **create-order.blade.php** – показва формата за създаване на нова поръчка .
* **edit-order.blade.php** – показва формата за редактиране на съществуваща поръчка.
* **new-orders.blade.php** – показва таблица с всички нови поръчки, чакащи одобрение.
* **create-customer-order.blade.php** – форма за подаване на заявка за транспортна услуга от клиент.

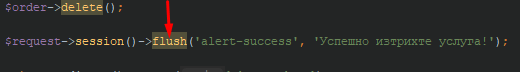
## 5.6. Тестване на приложението

При разработването на системата бяха извършени ръчни тестове при различни сценарии. По време на провеждането на тестовете бяха открити и отстранени значителен брой проблеми, като някои от тях са:

* При редактиране на поръчка, не се показваха старите стойности. Причината за това беше, че не бяха подадени данните за поръчката от контролера към изгледа за редакция.



* При изтриване на поръчка, потребителят биваше отписван от системата и пренасочван към страницата за вход в системата. Това беше един от най-трудно откритите проблеми, защото се оказа, че всъщност не е грешка или бъг, а правописна грешка.



Командата **session()->flush**, всъщност прави точно това – прекратява сесията на потребителя. Правилната команда за потвърждаване на успешно действие трябва да бъде **session()->flаsh**.

* В таблицата за визуализиране на информацията за потребителите хипервръзките за редактиране и изтриване на потребители, работиха само на първата страница. Проблемът се оказа в джаваскрипт функцията **DataTable**, която разделя таблицата на страници, според броя на потребители, които искаме да виждаме на една страница. За да се реши проблема, функцията трябваше да се преработи по следният начин :

<script>

$(document).ready(function () {

$('#datatables').on('click', '.remove', function () {

if (confirm('Сигурни ли сте?')) {

var id = $(this).data('user');

window.location = "/admin/delete-user/" + id;

}

});

$('#datatables').on('click', '.edit', function () {

var id = $(this).data('user');

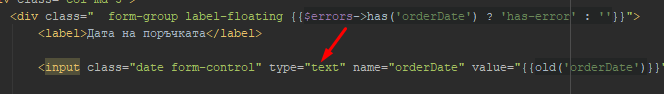
window.location = "/admin/edit-user/" + id;

});

});

</script>

* При създаване на нова поръчка, не се записваха датите **orderStartDate** и **orderEndDate** в таблицата **vehicle\_reservation**. За да може клиента по-лесно да въведе дата и някакъв ориентировъчен час за своята заявка, беше използван **datetimepicker** – класа, от библиотеката **jQuery**. Проблема се оказа в това, че този клас връща обект от тип **datetime**, а полетата **orderStartDate** и **orderEndDate** първоначално бяха от тип **date**. За да решим проблема, в изгледа за създаване на поръчка **create-order.blade.php**, типа на полетата, в които се въвевеждат датите, беше променен от **date** на **text.**



След това беше направена миграция и беше променен типа на полетата **orderStartDate** и **orderEndDate** от **date** на **datetime**.

* След промяна на статуса на превозно средство от „В ремонт“ на „Свободен“, ако в таблицата **vehicle\_repair** имаше други записи те се презаписваха с данните за последния ремонт. Причините за това се оказаха няколко :
* Във функцията **saveVehicle($id, VehicleFormRequest $request)**, в която запазваме промените, записвах променливата **$brokenVehicle** = VehicleRepair::find($vehicle->vehicleRepair->id); Това се оказа погрешно, защото трябваше да взема това превозно средство, за което **dateOutRepair=0**, тоест все още е в сервиза.
* В модела **VehicleRepair**, не бях добавил масива **$fillable** с полетата, които могат да се записват. Преди да се запишат промените проверявах данните които пристигат от $request **($request->all() as $k => $v)** , а всъщност не трябваше да проверявам тях, а тези от масива **$fillable**. За това промених този ред от кода така:

**foreach ($brokenVehicle->getFillable() as $k => $v)**

* След добавяне на полето order\_status\_id в таблицата orders, се появи следния проблем : в изгледа за създаване на поръчка, в полето за избор на шофьор, не се връщаха обратно тези шофьори, които са приключили с възложената им поръчка. Проблемът се оказа в това, че за да можем първоначално да извикаме всички шофьори, отговарящи на критерия за необходима шофьорска книжка, за управление на МПС, ние търсим тези от тях, които имат orders.id = 0. Тоест, ако искаме да покажем тези шофьори, които към даден момент са на разположение, то тяхното id, не бива да фигурира в поръчките. Изглежда логично, но ако изпратим един и същ шофьор на повече от една поръчка, неговото id вече го има в таблицата orders. Реших този проблем по следния начин - вместо да слагам флагове или полета за текущия статус на шофьорите, просто когато шофьорът изпрати заявка, че е приключил с поръчката занулявам неговото id : $order->driver\_id = NULL. Това по никакъв начин няма да доведе до загуба на данни, тъй като в таблицата **logvehicles** се пазят всички данни за превозните средства, включително от кога и до кога са били на път, и кой ги е управлявал.

**6. Ръководство за работа със системата**

За да бъде напълно ефективно използвана, всяка система трябва да има подробни инструкции за работа с нея и ясно дефинирани системни изисквания.

## 6.1. Инсталиране на системата

### 6.1.1. Изисквания при инсталиране на системата

За да се инсталира успешно системата, уеб сървърът трябва да отговаря на следните изисквания:

* PHP >= 5.6.4
* OpenSSL PHP Extension
* PDO PHP Extension
* Mbstring PHP Extension
* Tokenizer PHP Extension
* XML PHP Extension
* Apache сървър
* СУБД MySQL
* Composer

### 6.1.2. Инсталиране на системата

#### 6.1.2.1. Инсталиране в Linux среда

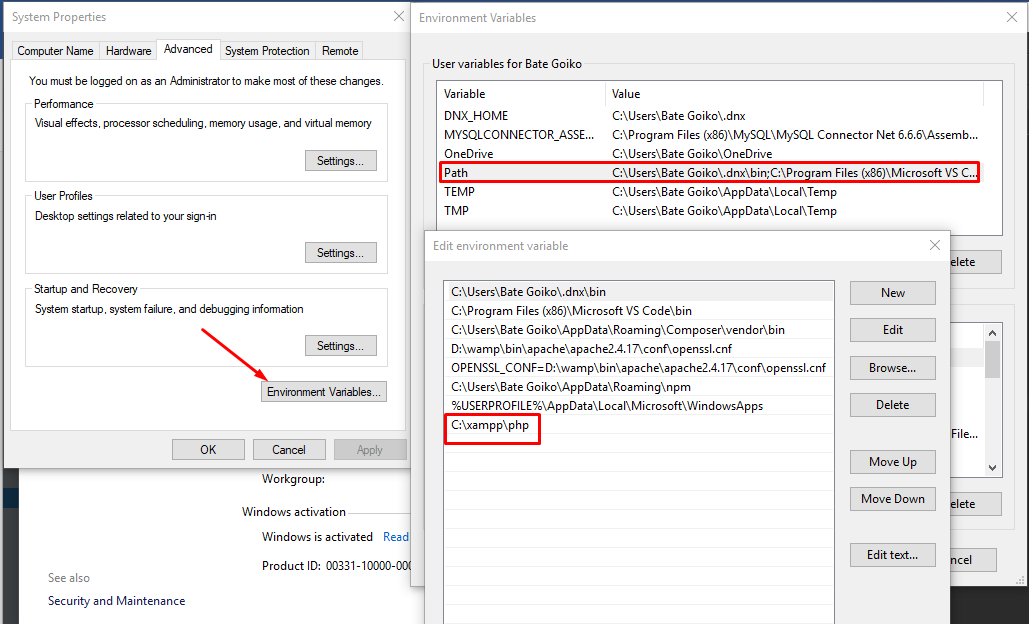
* Проектът трябва да се копира в публичната директория на сървъра. Обикновено е **/var/www/**.
* Инсталиране на всички библиотеки нужни за работата на софтуерната рамка Laravel. За целта трябва в директорията на проекта, чрез терминала да се пусне следната команда: **composer install**
* След инсталиране на проекта, трябва на сървъра да се укаже коя е публичната директория на проекта. В случая тя е **public/**.
* След настройване на проекта трябва да се конфигурира връзката с базата от данни. За целта в главната директория на проекта се намира файл наречен **.env.example**. С отворения терминал достъпваме главната директория на проекта и пускаме следната команда: **cp .env.example .env**

Тази команда ще копира съдържанието на .env.example в .env файла.

* След създаване на .env файла, трябва да се редактира и да се въведат конфигурационните данни :
  + **APP\_ENV** – служи за указване на това дали системата е инсталирана в режим на разработване. Ако е в режим на разработване всички грешки ще се показват на сайта. Двата режима са : local – режим на разработване и production – при този режим на сайта няма да се показват грешки
  + **DB\_CONNECTION** – служи за указване на СУБД. Обикновено се използва mysql
  + **DB\_HOST** – указва сървъра, където се намира базата от данни. Обикновено е 127.0.0.1
  + **DB\_PORT** – указва порта, чрез който ще се свързва с базата от данни.
  + **DB\_DATABASE** – името на използваната база от данни
  + **DB\_USERNAME** – потребителско име за достъп до базата от данни
  + **DB\_PASSWORD** – парола за достъп до базата от данни.
* След като се конфигурира системата и се създаде база от данни, трябва да се създаде схемата на базата. За целта се пуска следната команда: **php artisan migrate**
* След създаване на схемата на базата, трябва да се добавят записите за ролите на потребителите, статусите на превозните средства, статусите за поръчките. За целта се пуска следната команда:**php artisan db:seed**

#### 6.1.2.2. Инсталиране в Winows среда

* За да се инсталиране PHP, apache, MySQL трябва да се инсталира програмата XAMPP. Тя съдържа всичко нужно за стартиране на уеб базирано приложение.
* След инсталиране на XAMPP, трябва да укажем пътя до PHP файла на операционната система. За целта отваряме настройките на променливите за средата (environment variables) – **My Computer -> Properties -> Advanced system settings -> Environment variables**. Тук в променливите за потребителя, редактираме записа **Path** и на края на списъка добавяме пътя до PHP с префикс „;“. Обикновено е **C:\xampp\php -> ;C:\xampp\php**. На фигура 7.1 е показано как трябва да изглежда настройката.
* След настройката на сървъра, трябва да се инсталира composer. За целта, от сайта на производителя <https://getcomposer.org/download/>, се изтегля инсталационния файл и се инсталира на компютъра.
* Кодът на проекта трябва да се постави в htdocs директорията на XAMPP. Обикновено пътят до папката е **C:\xampp\htdocs**.
* С помощта на командния прозорец пускаме командата **: composer install**
* Копираме файла **.env.example** в **.env** и конфигурираме системата както е показано в точка 6.2.2.1



*Фигура 6.1 Добавяне на пътя до РНР*

* След като се конфигурира системата и се създаде база от данни, трябва да се пусне командата **php artisan migrate** за създаване на схемата на базата
* След създаване на схемата трябва да се пусне командата **php artisan db:seed** за добавяне на администраторския потребител, записите за ролите на потребителите, статусите на превозните средства, статусите за поръчките.
* След конфигуриране на системата трябва да се укаже на сървъра къде се намира проекта. За целта трябва да се редактира **hosts** файла и да се добави ред с тестов домейн който да сочи до локалния IP адрес. Пример: **127.0.0.1 avtopark.tv**. За да може да се редактира **hosts** файла, потребителя на операционната система трябва да има администраторски права и с помощта на Notepad, стартиран като администратор, редактираме файла, който се намира в директорията **C:\Windows\System32\drivers\etc\** (фиг. 6.2).



*Фигура 6.2 Редактиране на host файла*

* След като се редактира hosts файла, трябва да се укаже на apache коя е основната директория на проекта. За целта редактираме файла **httpd-vhosts.conf**, който се намира в **C:\xampp\apache\conf\extra** директорията и добавяме следните настройки:

<VirtualHost \*:80>

DocumentRoot "C:\xampp\htdocs\AvtoparkRuse\public"

ServerName avtopark.tv

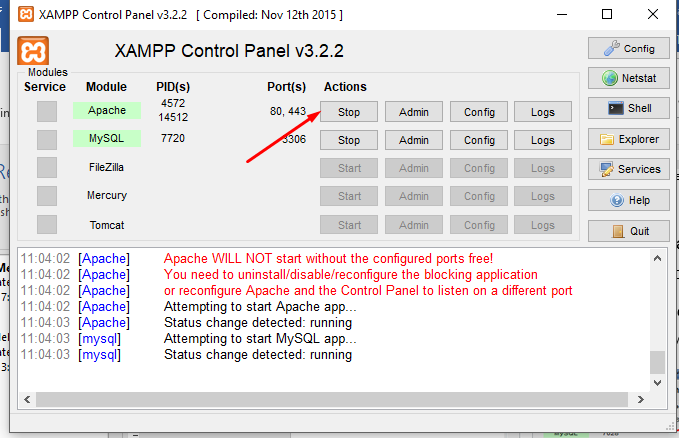
ServerAlias www.avtopark.tv

# ...

</VirtualHost>

Тук се указва на сървъра, че основната директория от където се стартира проекта е директорията **public.**

* След направените промени, рестартираме apache сървъра през контролния панел на XAMPP (фиг. 6.3).



*Фигура 6.3 Рестартиране на Apache сървъра*

## 6.2. Административния модул

Достъп до административния модул има потребителят администратор, който се създава с инсталирането на системата. Той може да създава и управлява останалите категории потребители, включително и други админстратори.

Подразбиращите се данни за първия потребител са:

* Имейл – **tisho26bg@gmail.com**
* Парола – **123456**

След вход в системата, администраторът може да редактира данните си.

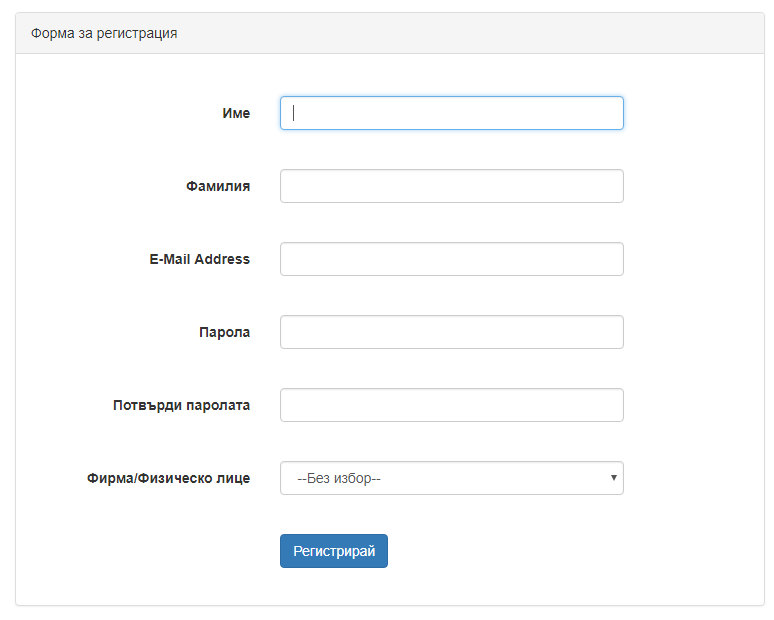
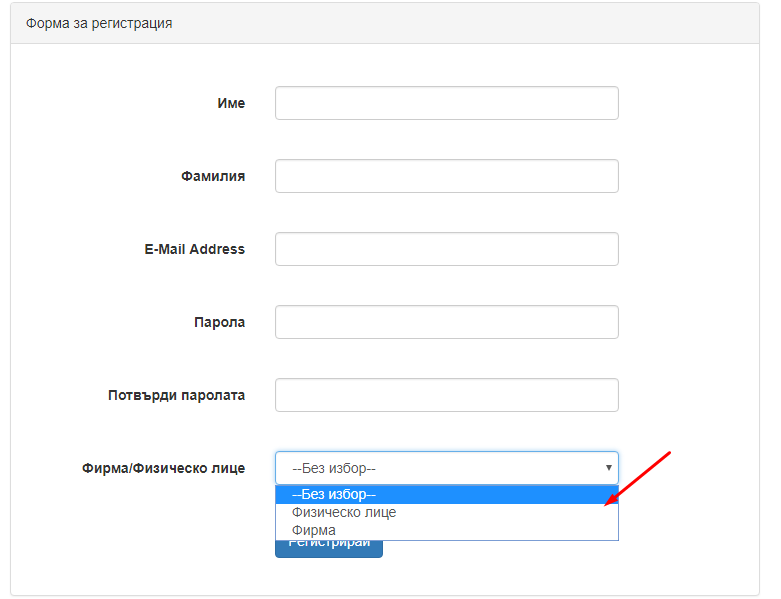
## 6.3. Ръководство за потребител „Клиент”

### Всички потребители на системата трябва да разполагат единствено с някой от съвременните браузъри. Въвежда се <https://avtoparkruse.top> и се показва екрана за оторизация. Клиентът може да се регистрира или да влезе в системата с вече съществуващи потребителско име и парола.

### 6.3.1. Регистрация в системата

На фигура 6.4 е показана формата за регистрация в системата. Достъп до нея е от връзка „Register“. За да бъде успешна регистрацията е необходимо всички полета да бъдат попълнени, двете въвеждания на пароли да съвпада, а данните в полетата да отговарят на следните изисквания:

* Символите в полетата **Име** и **Фамилия** могат да бъдат най-много 255.
* Символите в полето **E-mail Address** могат да бъдат най-много от 255 символа, да е уникално за системата и да представлява валиден имейл адрес.
* Символите в полето **Парола** трябва да са минимум шест на брой и да съвпадат с тези в полето **Потвърди паролата.**

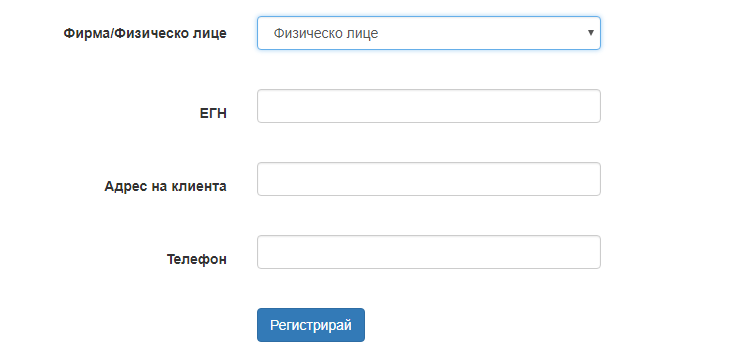
 

*Фигура 6.4 Форма за регистрация Фигура 6.5 Избор на статус на клиента*

Клиентът може да бъде както фирма, така и физическо лице и това се определя от полето **Фирма/Физическо лице** (фиг. 6.5)

Ако клиентът избере **Физическо лице**, се активират три допълнителни полета (фиг. 6.5), които той трябва да попълни, за да завърши регистрацията си :

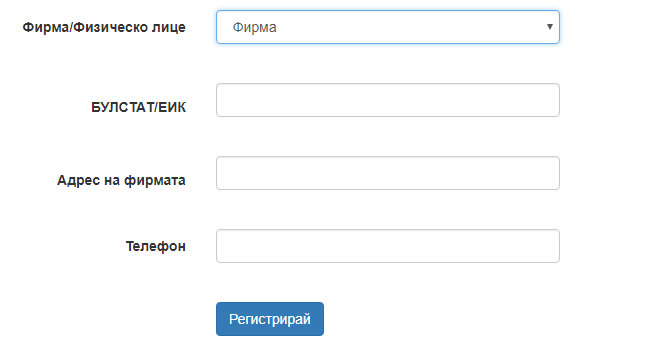
* Полето **EГН**, трябва да съдържа само цифри и да се състои от точно 10 символа.
* Полето **Адрес на клиента,** може да съдържа най-много 255 символа.
* Полето **Телефон,** трябва да съдържа валиден телефонен номер.



*Фигура 6.5 Клиент – Физическо лице*

Ако клиентът е **Фирма**, се активират полета (фиг. 6.6), които той трябва да попълни за да завърши регистрацията си :

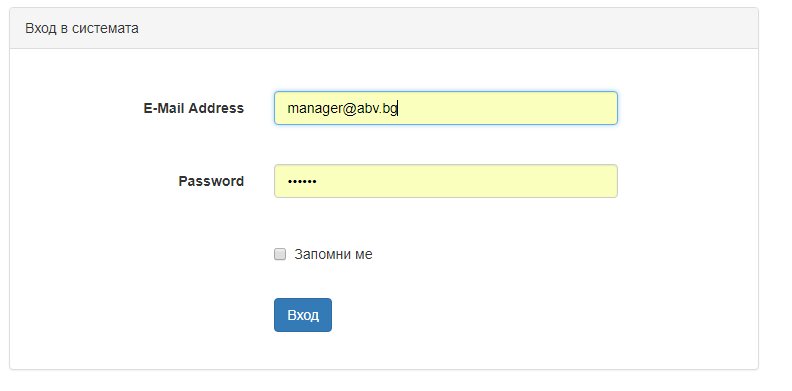
* Полето **БУЛСТАТ/ЕИК**, е задължително.
* Полето **Адрес на фирмата,** може да съдържа най-много 255 символа.
* Полето **Телефон,** трябва да е валиден телефонен номер.



*Фигура 6.6 Клиент – Фирма*

### 6.3.2. Вход в системата

На фигура 6.7 е показана формата за вход в системата. Изисква се потребителят да въведа имейл и парола, като и двете полета са задължителни. Ако потребителят иска да остане вписан в системата, трябва да се избере “Запомни ме”, след това, може да натисне бутона „Вход“, което ще го впише в системата.

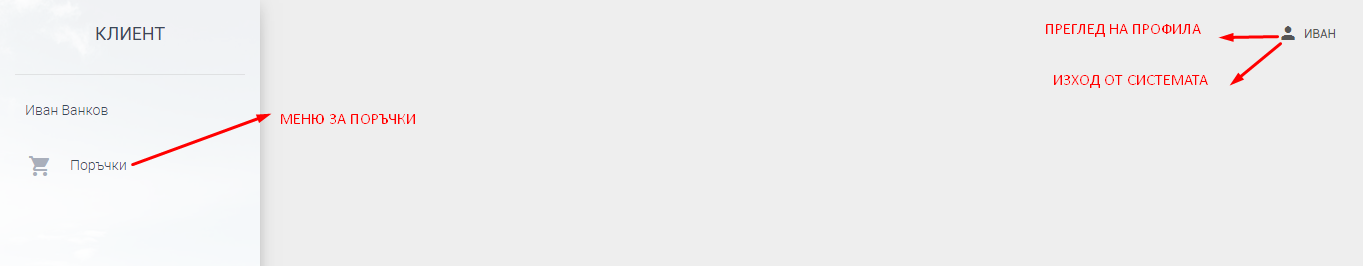


*Фигура 6.7 Вход в системата*

### 6.3.3. Основна страница след вписване в системата като клиент

На фигура 6.8 е показана основната страница, която се визуализира след вписване в системата. В тази страница се намира менюто за поръчки. Също така има падащо меню, от което клиентът може да влезе в профила си или да се отпише от системата.

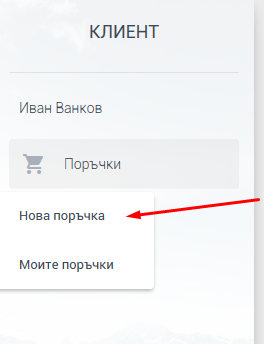
За да подаде заявка за поръчка, потребителят трябва да натисне бутона **Поръчки**, след което от падащото меню да избере **Нова поръчка**.



*Фигура 6.8 Основна страница след вписване на клиент в системата*

#### 6.3.3.1. Подаване на заявка за поръчка

Ако клиентът избере да направи нова поръчка (фиг. 6.9), в основния изглед, се отваря формата за създаване на нова поръчка, показана на фигура 6.10.



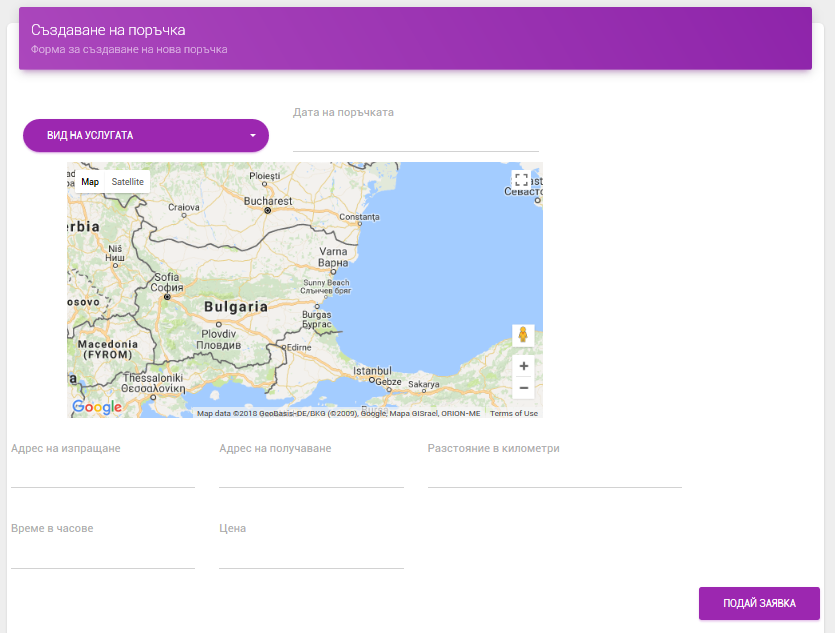
*Фигура 6.9 Меню с избор за създаване на нова поръчка, или преглед на всички поръчки*

Системата предлага три основни вида транспортни услуги, показани на таблица 1:

*Таблица 1 Видове услуги*

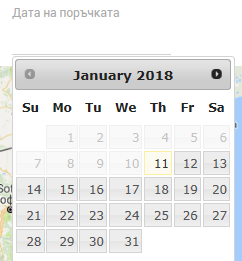
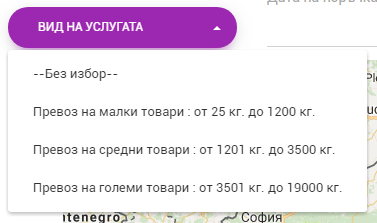
|  |  |  |
| --- | --- | --- |
| **УСЛУГА** | **МИНИМАЛНО ТЕГЛО** | **МАКСИМАЛНО ТЕГЛО** |
| Превоз на малки товари | 25кг. | 1200кг. |
| Превоз на средни товари | 1201кг. | 3500кг. |
| Превоз на големи товари | 3501кг. | 1900кг. |

Клиентът избира съответния вид услуга, в зависимост от товара, който трябва да бъде превозен.



*Фигура 6.10 Форма за създаване на нова поръчка*

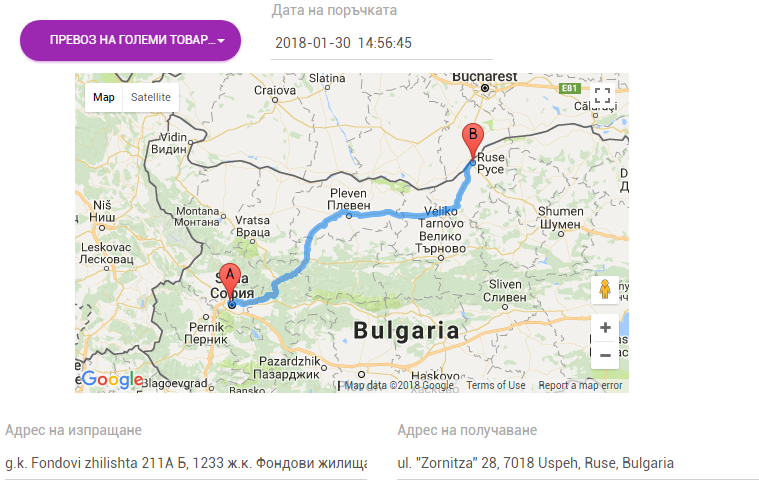
За да подаде заявка за поръчка, клиента трябва да избере вида на услугата, която да ползва (фиг.6.11), дата на поръчката (фиг. 6.12) и маршрут (фиг. 6.13).

  
*Фигура 6.11 Избор на услуга Фигура 6.12 Избор на дата*

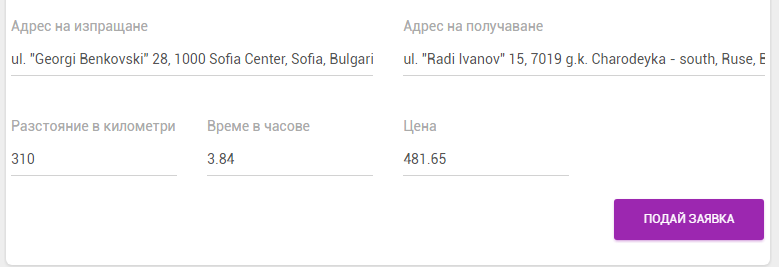
Маршрутът може да бъде зададен по два начина :

* Като се използва картата (фиг.6.10). С натискане на мишката се задават началния и крайния адрес на поръчката(т.А и т.В).
* Като се въведат ръчно данните в полетата **Адрес на изпращане** и **Адрес на получаване.**

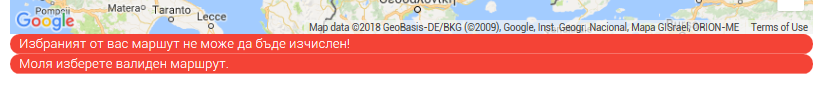
Ако маршрутът е коректен и може да се намери път между двата адреса, системата попълва полетата **Разстояние в километри**, **Време в часове** и **Цена** (фиг. 6.14). В противен случай се извежда съобщение за грешка (фиг. 6.15) и клиентът трябва да зададе нов маршрут.



*Фигура 6.13 Определяне на маршрута на поръчката*

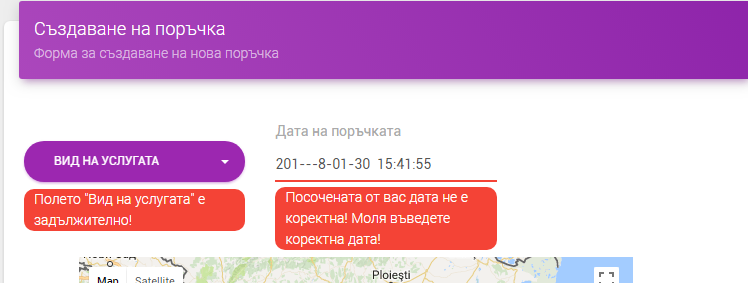


*Фигура 6.14 Време, разстояние и цена*



*Фигура 6.15 Съобщение за невалиден маршрут*

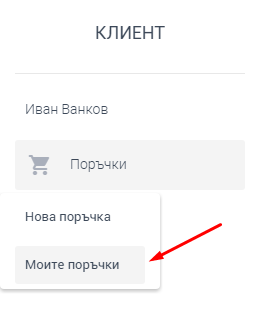
След като попълни всички данни във формата, клиентът изпраща заявката си с натискане на бутон **Подай заявка.** Ако всички полета са попълнени коректно поръчката се изпраща за обработка, а клиентът пренасочен към страницата „Мои поръчки“. При некоректно попълнени или липсващи данни в полетата на заявката, системата генерира съответно съобщение (фиг. 6.16).



*Фигура* 6.16 *Съобщение за грешни или липсващи данни в заявката*

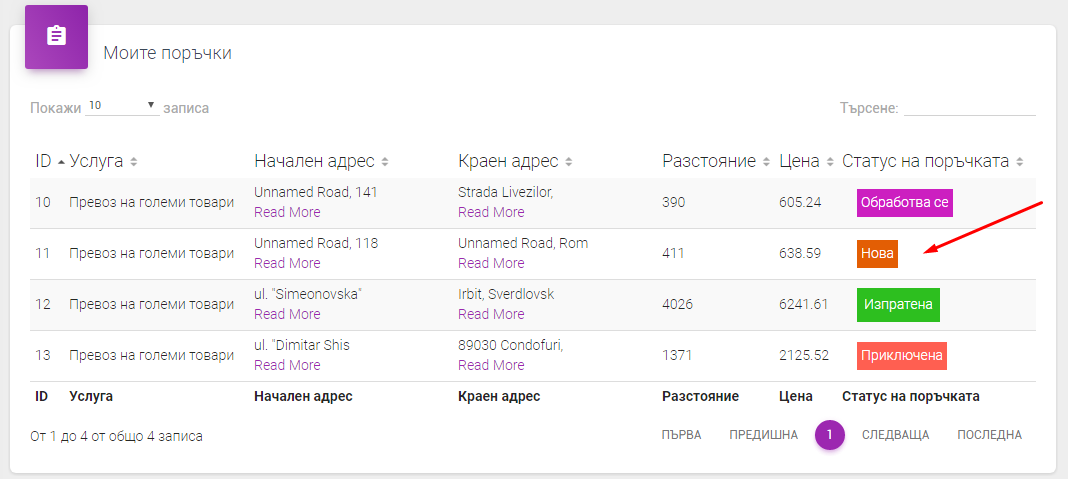
#### 6.3.3.2. Преглед на всички поръчки

Клиентът може да направи справка за своите поръчки като избере опцията **Моите поръчки** от меню **Поръчки** (фиг. 6.17).



*Фигура* 6.17 *Преглед на поръчките*

В таблицата освен основните параметри на поръчката се показва и **Статус на поръчката** (фиг. 6.18).



*Фигура 6.18 Преглед на всички поръчки направени от клиент*

Всяка поръчка може да има четири статуса :

* **Нова** – всяка заявка за поръчка направена от клиент, автоматично получава статус **Нова**. Това е поръчка, постъпила като заявка при мениджъра на системата, но все още не е обработена.
* **Обработва се** – това е поръчка, която е обработена от мениджъра на системата. Той е прегледал всички параметри на поръчката, потвърдил е, че за исканата от клиента дата има свободно превозно средство, което може да изпълни тази поръчка и е назначил шофьор, който да извърши поръчката.
* **Изпратена** - поръчката е тръгнала от началния адрес.
* **Приключена** – поръчката е стигнала крания си адрес. Превозното средство извършило поръчката е разтоварено и е свободно да поеме друга поръчка.

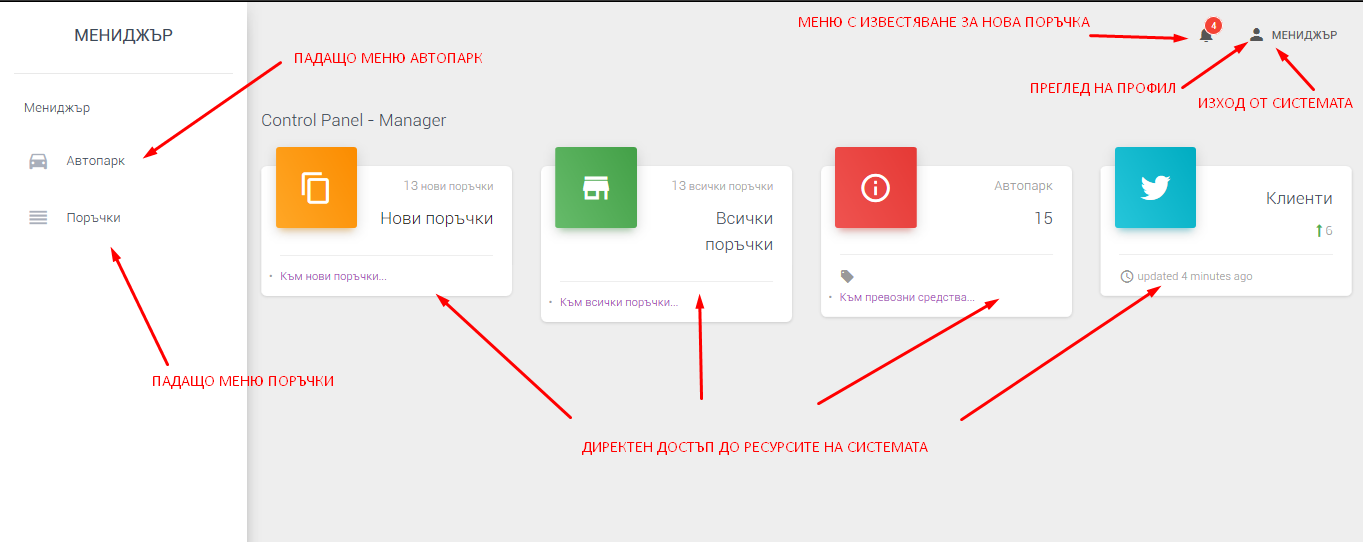
За удобство и лесно различаване, статусите са в различни цветове.

## 6.4. Ръководство за потребител „Мениджър”

### Потребител мениджър има зададено потребителско име и парола от администратора, с които да получи достъп до подсистема Мениджър.

### 6.4.1. Основна страница след вписване в системата като мениджър

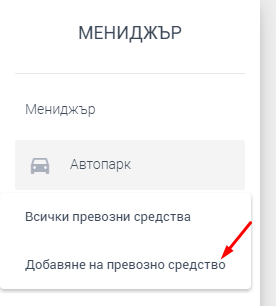
На фигура 6.19 е показана основната страница след влизане в системата като мениджър. В тази страница се показват връзки към ресурсите на системата, меню с известия за нови поръчки, меню за управление на автопарка и меню за управление на поръчките.



*Фигура 6.19 Основен екран на системата за потребител Мениджър.*

### 6.4.2. Добавяне на ново превозно средство в системата.

Ако мениджърът иска да добави ново превозно средство в системата той трябва да избере **Добавяне на ново превозно средство**, от падащото меню **Автопарк** (фиг. 6.20).



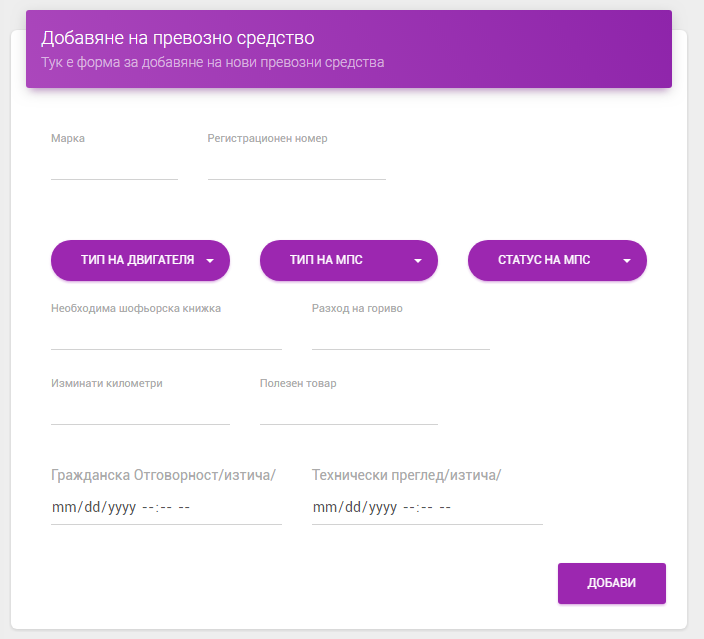
*Фигура 6.20 Добавяне на ново превозно средство*

В основния изглед се появява формата за добавяне на превозно средство (фиг. 6.21).

Фирмата използваща системата, разполага със собствен атопарк. В него влизат три основни типа превозни средства : лекотоварни автомобили, бусове и камиони. Връзките между превозните средства и услугите предоставяни от фирмата са показани в таблица 2.

*Таблица 2 Връзки превозни средства – услуги*

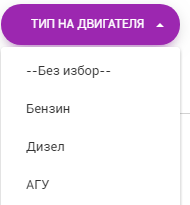
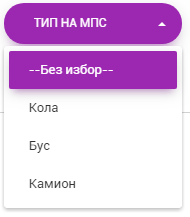
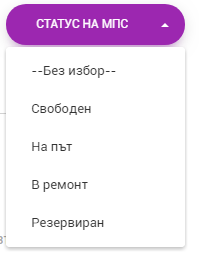
|  |  |  |
| --- | --- | --- |
| **Превозни средства** | **Услуга** | **Полезен товар** |
| Лекотоварни автомобили | Превоз на малки товари | до 1200 кг. |
| Бусове | Превоз на средни товари | до 3500кг |
| Камиони | Превоз на големи товари | до 19000кг |



*Фигура 6.21 Форма за добавяне на превозно средство*

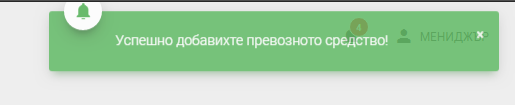
За да е успешно добавянето на превозно средство, трябва всички полета от формата да бъдат попълнени. Също така полетата трябва да отговарят на следните изисквания :

* Съдържанието на полето **Марка** трябва да се състои най-много от 255 символа.
* Съдържанието на полето **Регистрационен номер** трябва да се състои най-много от 255 символа.
* От падащото меню **Тип на двигателя**, мениджърът избира какъв тип двигател притежава превозното средство(фиг. 6.22).
* От падащото меню **Тип на МПС,** мениджърът избира какъв е вида на превозното средство(фиг. 6.23).
* От падащото меню **Статус на МПС**, мениджърът избира с какъв статус постъпва превозното средство в автопарка (фиг. 6.24). Понеже това е ново превозно средство се предполага, че е технически годно, за това му се слага статус **Свободен**, т.е. е може да бъде използвано за превоз на товари.
* В полето **Необходима шофьорска книжка** се записва каква шофьорска книжка трябва да притежава водача, който ще управлява превозното средство.
* В полето **Разход на гориво** се записва какъв е средния разход на превозното средство.
* За всяко ново превозно средство в полето **Изминати километри** се записват 0 км. След всяка поръчка системата добавя автоматично в това поле, изминатите километри. Това го правим за да знаем колко точно километра е изминало превозното средство от както е постъпило в автопарка.
* В полето **Полезен товар** се записва колко е полезния товар на превозното средтво.
* Полетата **Технически преглед /изтича/**  и **Гражданска отговорност / изтича/**, трябва да са валидни дати.

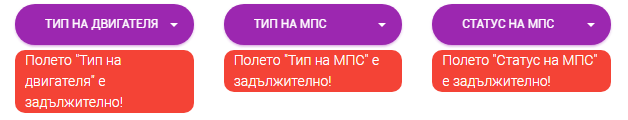
*Фигура 6.22 Фигура 6.23 Фигура 6.24*

След като попълни всички данни във формата, мениджъра може да добави превозното средство в системата си чрез натискане на бутона **Добави**.Аковсички полета са попълнени коректно, превозното средство се добавя в системата и на екрана се показва надпис с потвърждение на това (фиг. 6.25). Мениджъра бива пренасочен към страницата за преглед на всички превозни средства.



*Фигура 6.25 Успешно добавяне на превозно средство*

Ако има празни или не коректно попълнени полета, системата връща съобщение за грешка, като показва кои полета са не валидни (фиг. 6.26).

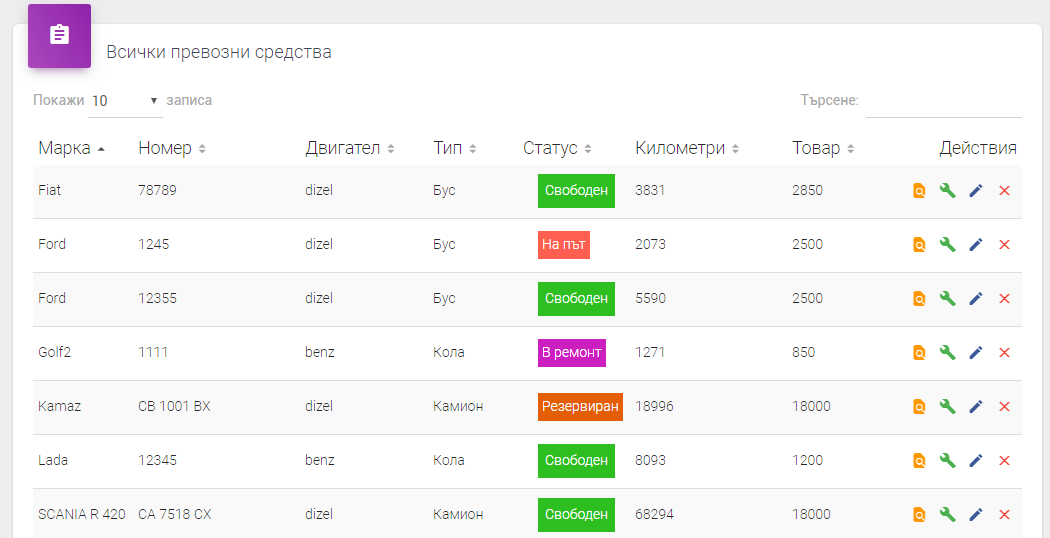


*Фигура 6.26 Съобщение за грешни или не коректно попълнени полета*

### 6.4.2. Управление на автопарка

За да прегледа всички превозни средства регистрирани в системата, мениджърът трябва да избере опцията **Всички превозни средства**  от падащото меню **Автопарк** (фиг. 6.20).

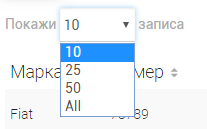
В основния изглед се отваря таблица с данните за превозните средства регистрирани в системата (фиг. 6.27).



*Фигура 6.27 Превозни средства регистрирани в системата*

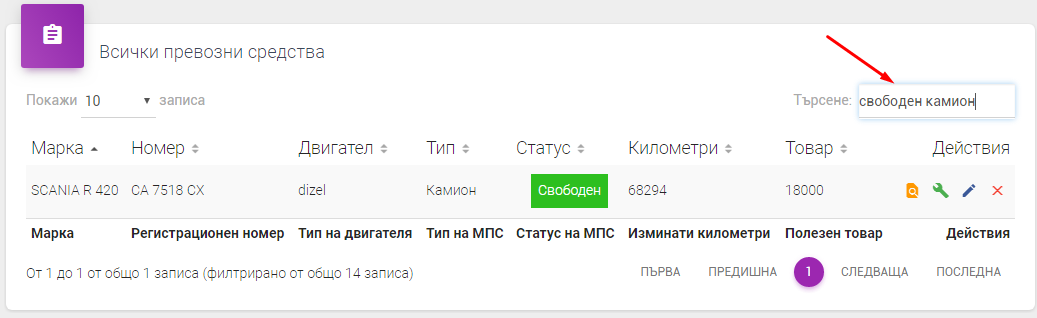
В нея са показани всички данни за превозните средства : марка, регистрационен номер, двигател, тип на превозното средство, изминати километри до този момент и статус на превозното средство към настоящия момент.

За улесняване работата на мениджъра, в хедъра на таблицата са добавени две полета **Търсене** и **Покажи \_\_\_\_ записа** . Мениджърът може да избира колко записа иска да вижда на една страница (фиг. 6.28).



*Фигура 6.28 Показване на определен брой записи на една страница*

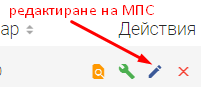
С помощта на търсачката, мениджърът може да търси превозни средства по желан от него критерий, например всички свободни камиони регистрирани в системата към настоящия момент (фиг. 6.29).



*Фигура 6.29 Използване на търсачката за намиране на свободни камиони*

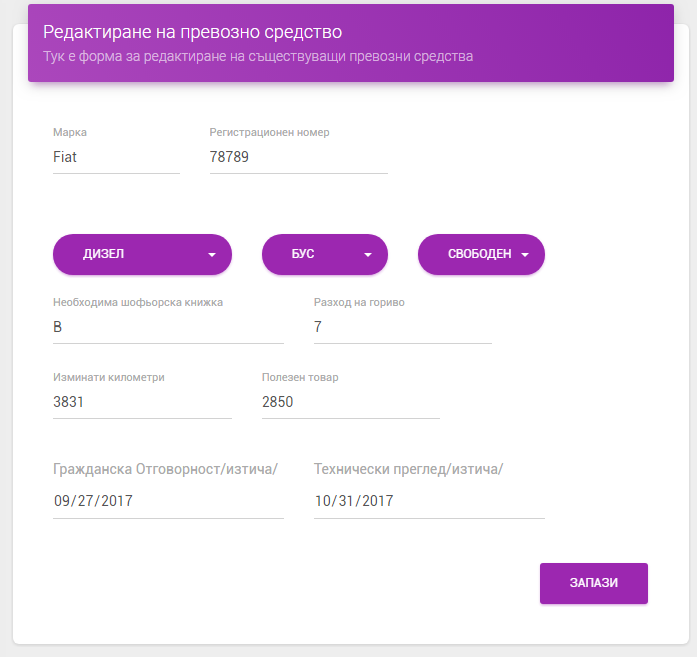
#### 6.4.2.1. Редактиране на превозно средство

Редактирането на превозно средство, става с помощта на синия бутон, намиращ се в менюто за **Действия** с превозни средства (фиг. 6.30).



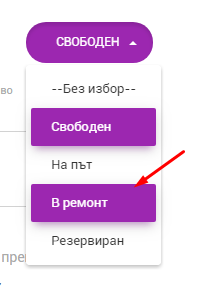
*Фигура 6.30 Редактиране на превозно средство*

След натискането на бутона, в основния изглед се отваря формата за редактиране на превозно средство (фиг. 6.31). Тя е идентична с тази за създаване.



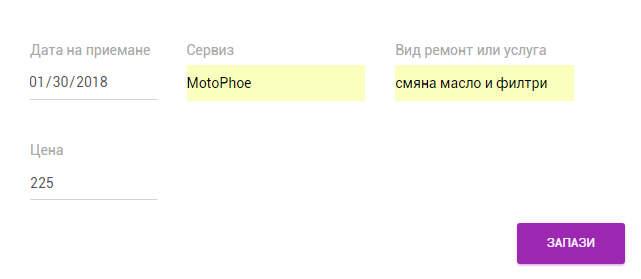
*Фигура 6.31 Форма за редактиране на превозно средство*

От тук могат да се редактират всички данни за превозното средство, включително и неговия статус. Ако се налага едно превозно средство да бъде ремонтирано по някаква причина, неговият статус се променя от падащото меню **Статус** (фиг.6.32)



*Фигура 6.32 Промяна на статус на превозно средство*

След избирането на тази опция, във формата се появяват четири допълнителни полета : **Дата на приемане**, **Сервиз**, **Вид ремонт или услуга** и **Цена**, които следва да бъдат попълнени (фиг. 6.33).



*Фигура 6.33 Допълнителни полета при изпращане на МПС в сервиз*

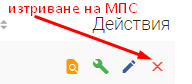
След натискането на бутона **Запази**, статуса на превозното средство се променя на „В ремонт“ и то не може да бъде използвано, докато не бъде ремонтирано. Когато ремонта приключи, статуса се променя отново от менюто показано на фигура 6.32. Избира се статус **Свободен.** Във формата се появява едно ново поле **Дата на излизане от сервиз** (фиг. 6.33), то трябва да бъде попълнено и да се натисне бутона **Запази**.



*Фигура 6.33 Дата на излизане на МПС от сервиз*

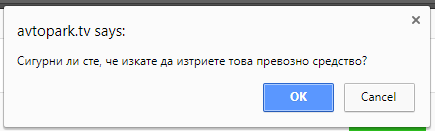
#### 6.4.2.2. Изтриване на превозно средство

Изтриването на превозно средство, става с помощта на червения бутон, намиращ се в менюто за **Действия** с превозни средства (фиг. 6.34).



*Фигура 6.34 Изтриване на превозно средство*

След натискане на бутона за изтриване, се появява диалогов прозорец за потвърждаване на изтриването (фиг. 6.35).

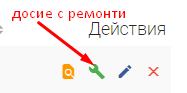


*Фигура 6.35 Диалогов прозорец за потвърждаване за изтриване на превозно средство*

След потвърждаване, данните за превозното средство се изтриват от базата данни.

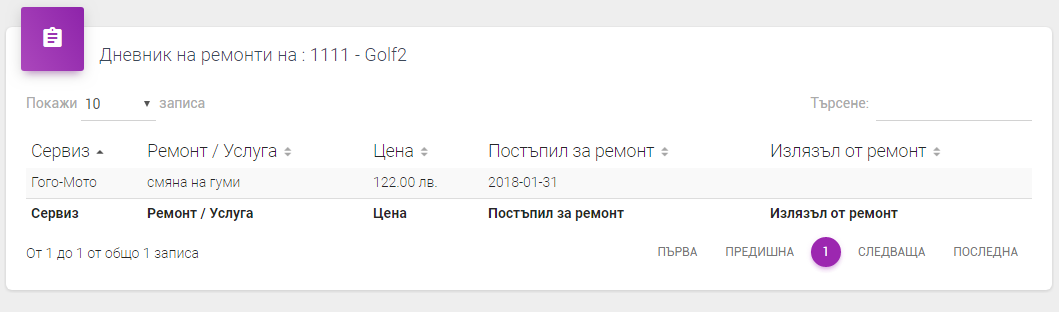
#### 6.4.2.3. Преглед на досие за ремонти на превозно средство

Всяко превозно средство разполага с досие за ремонти. В него се записва необходимата информация. За да се отвори досието, трябва да се натисне зеленият бутон намиращ се в менюто за **Действия** с превозни средства (фиг. 6.36).



*Фигура 6.36 Досие с ремонти*

След натискане на бутона, в основния изглед се отваря таблица, в която са описани всички ремонти (фиг. 6.37).



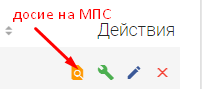
*Фигура 6.37 Дневник за ремонти на МПС*

В конкретният случай, показан на фигура 6.37, се вижда, че превозното средство е постъпило за ремонт в сервиз „Гого-Мото“, за смяна на гуми, на 31.01.2018. Полето „Излязъл от ремонт“ е празно защото ремонта не е приключил все още. Когато това стане, мениджърът ще промени статуса на превозното средство, както е описано в точка **6.4.2.1.** и датата ще бъде записана в таблицата.

#### 6.4.2.4. Преглед на досие за превозно средство

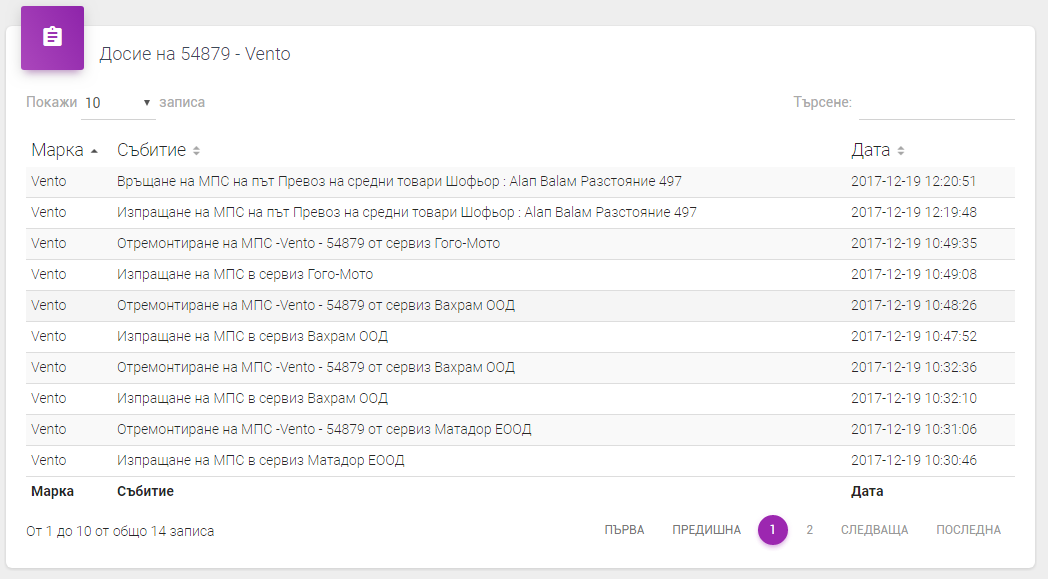
Това досие е подобно на описаното в точка 6.4.2.3. Разликата е в това, че в него се записва всяко едно събитие, свързано с конкретното превозно средство.

За да се отвори това досие, трябва да се натисне жълтият бутон, намиращ се в менюто за **Действия** с превозни средства (фиг. 6.38).



*Фигура 6.38 Досие на превозно средство*

След натискане на бутона, в основния изглед се отваря таблица, в която са описани всички дейности свързани с изпраното превозно средство (фиг. 6.39).



*Фигура 6.39 Досие на превозно средство*

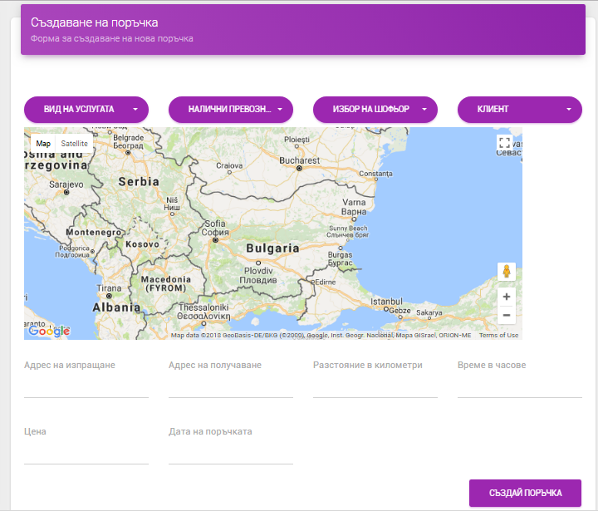
#### 6.4.2.5. Приемане и обработване на поръчки

Приемането и обработването на поръчки, е другата основна функция, която има мениджъра на системата. Системата е предвидена така, че да могат да се подават заявки за поръчки по два начина.

1. **Подаване на заявка за транспортна услуга от клиент, регистриран в системата, но нямащ достъп до нея**

Всеки клиент регистриран в системата, може да подаде заявка за поръчка и тя да да бъде разгледана от мениджъра. Ако клиент няма достъп до интернет, или няма възможност да влезе в системата, то той може да се обади по телефона и да направи своята поръчка. В този случай ениджърът трябва да избере опцията **Нова поръчка** от падащото меню **Поръчки**.

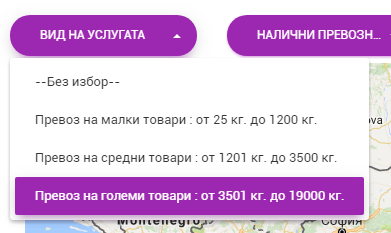
В основния изглед се отваря форма за създаване на нова поръчка (фиг. 6.40).



*Фигура 6.40 Създаване на нова поръчка*

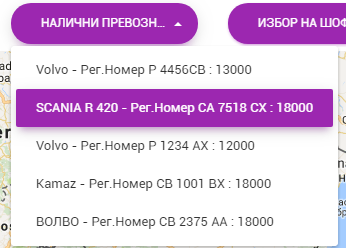
Мениджърът трябва да попълни следните полета :

* **Вид на услугата** – услугите са разделени на три категории (фиг. 6.41).



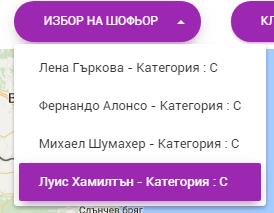
*Фигура 6.41 Избор на услуга*

* **Налични превозни средства** – след като бъде избрана услугата, в полето **Налични превозни средства**, се показват само тези превозни средства, които са технически изправни, не са заети, и имат техническите характеристики да изпълнят тази услуга (фиг. 6.42).



*Фигура 6.42 Избор на превозно средство*

* **Избор на шофьор** – след като се изберат вида на услугата, и превозното средство в полето **Избор на шофьор** се появяват само тези шофьори, които са свободни, и имат нужната правоспособност (фиг. 6.43).



*Фигура 6.43 Избор на шофьор*

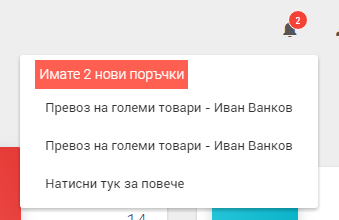
* **Клиент** – в полето **Клиент** се въвежда името на клиента, направил поръчката.
* **Определяне на маршрута на поръчката** – задаването на маршрута на поръчката се прави по същият начин, описан в точка **6.3.3.1.**
* **Дата на поръчката** – в това поле се попълва датата, на която клиента иска да бъде извършена неговата поръчка.

Полетата **Разстояние, Време в часове** и **Цена**, се попълват автоматично от системата. Ако всички данни са въведени коректно, след натискане на бутона **Създай поръчка**,поръчката се записва в базата дании и мениджърът бива препратен към страницата с всички поръчки.

1. **Приемане на заявка от клиент, изпратена чрез системата.**

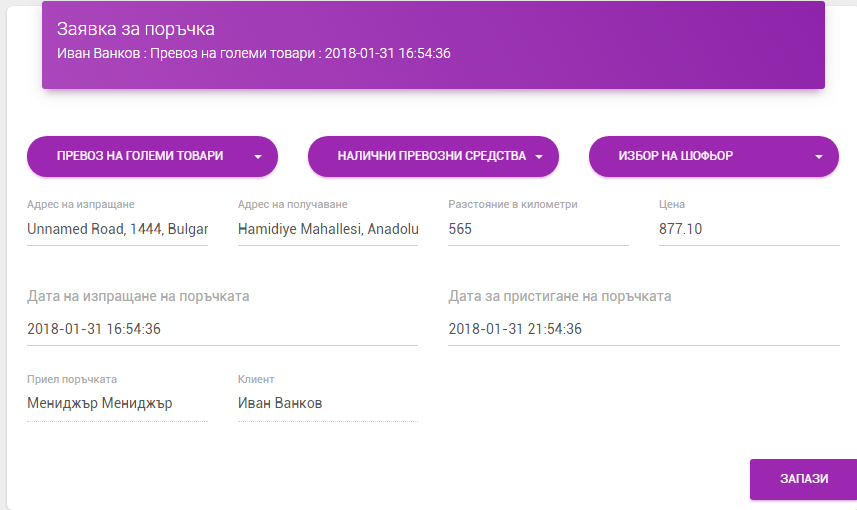
Както видяхме в точка 6.3.3.1. Подаване на заявка за поръчка, регистрираните в системата клиенти могат да подават заявки за поръчки. В следващите редове, ще обясним, какво точно се случва с тези заявки и как те биват обработени.

Когато клиент успешно подаде своята заявка, мениджъра получава известие за това. На фигура 6.44 е показано менюто с известия за нови поръчки.



*Фигура 6.44 Известия за нови поръчки*

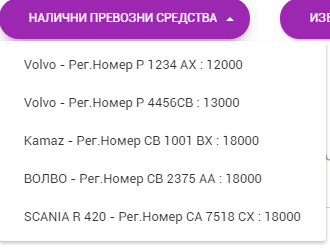
Опциите, които виждаме в падащото меню са хипервръзки към съответните поръчки. Когато бъде избрана една поръчка, в основния изглед се появява форма, в която са записани всички данни изпратени от клиента(фиг. 6.45).



*Фигура 6.45 Форма за приемане на поръчка*

В полето **Вид на услугата**, е записано „Превоз на големи товари“, защото клиента е избрал, че иска този вид услуга.

В падащото меню **Налични превозни средства**, следва да се покажат всички свободни превозни средства, които могат да извършват този вид услуга. Мениджърът избира едно от всички свободни, например SCANIA R 420 (фиг. 6.46).



*Фигура 6.46 Налични свободни превозни средства*

По същият начин се процедира и с полето **Избор на шофьор**.

Ако всички данни са попълнени коректно, след натискане на бутона **Запази**, трябва да излезе надписът показан на фигура 6.47 и мениджъра се пренасочва към страницата с всички поръчки.



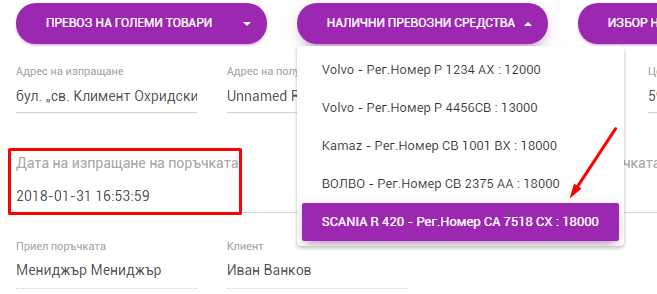
*Фигура 4.47 Успешно добавяне на поръчка*

Виждаме, че вече има само едно известие за нови поръчки (фиг. 6.48)



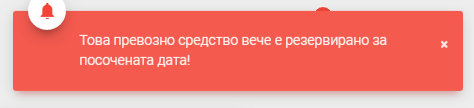
*Фигура 6.48*

Нека обработим и другата поръчка. Избираме поръчката от падащото меню, показано на фигура 6.44. Отваряме се същата форма както на фигура 6.45 и попълваме данните в нея. В полето **Избор на превозно средтсво**, обаче продължаваме да виждаме SCANIA R 420 (фиг. 6.49).



*Фигура 6.49*

Това е така защото, когато мениджърът запази една поръчка, той само „резервира“, превозното средство за датата на тази поръчка, то може да бъде използвано за други поръчки преди тази дата. И двете поръчки направени от клиента са за една и съща дата и за един и същ вид услуга, различен е само маршрута. Какво ще стане ако за втората поръчка изберем вече резервираната SCANIA R 420. На екрана се появява диалогов прозорец, предупреждаващ че за посочената дата, това превозно средство е резервирано (фиг. 6.50). Мениджърът остава на същата страница и трябва да избере друго превозно средство.

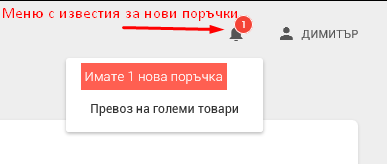


*Фигура 6.50 Диалогов прозорец с предупредителен надпис*

## 6.5. Ръководство за потребител „Шофьор”

### 6.5.1. Основна страница след вписване в системата като клиент

На фигура 6.51 е показана основната страница, която се визуализира след вписване в системата като „Шофьор“. В тази страница се намира менюто за известия за поръчки, предназначени за шофьора. Също така има падащо меню, от което клиентът може да влезе в профила си или да се отпише от системата.

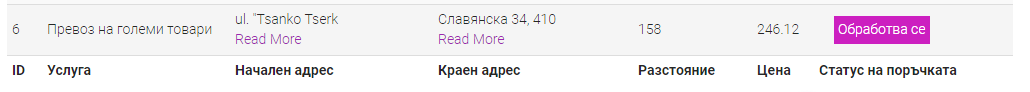


*Фигура 6.51*

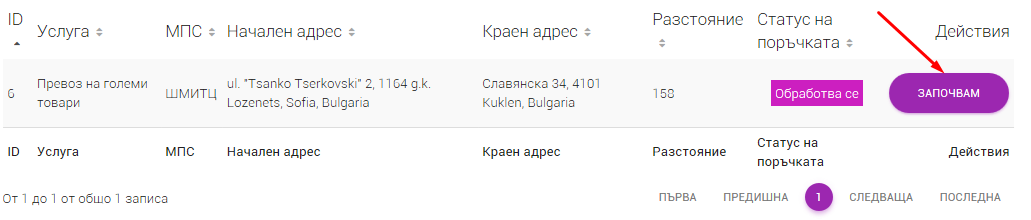
Следва описание на целия процес по обработването на една заявка за транспортна услуга:



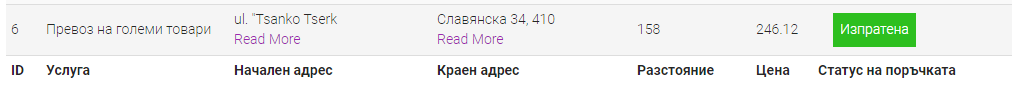
*Фигура 6.52 Заявка за поръчка от клиент*



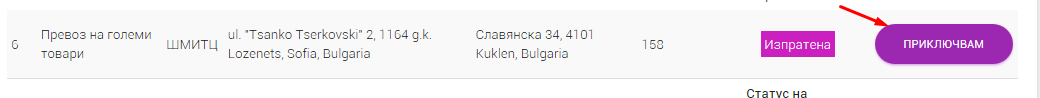
*Фигура 6.52 Поръчката е приета от мениджъра и е определен шофьор за нея*

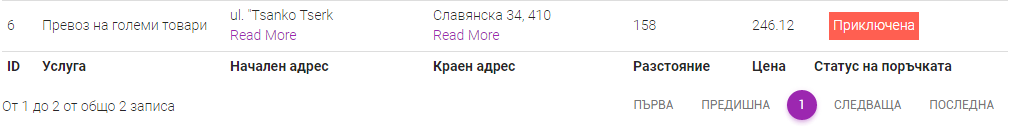


*Фигура 6.53 Шофьорът започва с изпълнението на възложената му поръчка*



*Фигура 6.54 Промяна на статуса на поръчката*

*Фигура 6.55 Приключване на поръчката*



*Фигура 6.56 Статус на поръчката „Приключена“*

* Подаване на заявка от клиент. На фигура 6.52 виждаме статуса на поръчка изпратена от клиент. Тя е „Нова“, защото все още не е обработена от мениджъра.
* Обработване на поръчката от мениджъра. На фигура 6.52 е показана вече обработена поръчка.
* Непосредствено преди да започне изпълнението на една поръчка, шофьорът трябва да натисне бутона **Започвам**(фиг. 6.53). Когато направи това, клиента може да види през своя профил, че статуса на поръчката се е променил на „Изпратена“(фиг 6.54).
* Приключване на поръчка – за да декларира, че поръчката е изпълнена, шофьорът трябва да натисне бутона **Приключвам**(фиг. 6.55). Това автоматично променя статуса на тази поръчка на „Приключена“(фиг 6.56).

# 7. Основни резултати, изводи и препоръки

Разработената информационна система представлява напълно завършена система за контрол и управление на автомобилен парк. Тя е изградена от четири модула за всяка категория потребител – Администратор, Клиент, Мениджър и Шофьор.

Постигнати са поставените цели:

* Разработено е приложение, което е платформено независимо и може да се инсталира на споделен хостинг;
* В системата могат да се регистрират фирми и физически лица като клиенти;
* Потребителят с административни права може да управлява останалите потребители;
* Създаден е модул „Мениджър“ за управление на автомобилния парк и обработката на поръчки;
* Създаден е модул „Клиент“, чрез който клиентите да могат да се регистрират, да правят он-лайн заявки за извършване на услуги и да следят състоянието им;
* Създаден е модул „Шофьор“ за водачите на превозни средства в
* автопарка;
* Системата притежава интуитивен потребителски интерфейс с изчистен дизайн;

Системата покрива изискванията за качество на съвременна и функционална система от този тип. Предвидени са методи и средства за въвеждане на коректни данни и предпазване от волни и неволни грешки. Системата е проектирана с изчистен дизайн и интуитивен интерфейс, като това я прави лесна за употреба от потребители с различно ниво на компютърна грамотност.

Голям плюс на тази система би било и разработването и включване на GPS модул, чрез който да се подобри маршрутната ефективност и диспечерската дейност. Всичко това би довело до по-ефективна работа на мениджъра, а проблемите в трафика могат да се следят по-лесно, за да се прогнозират и компенсират или избегнат евентуалните закъснения.

Всяка система може да се усъвършенства с добавяне на нови функционални възможностиния. В конкретния случай могат да се предвидят следните възможности:

* Групиране на товари по еднакъв или покриващ се маршрут;
* Изграждане на модул за GPS управление и мониторинг на автопарка;
* Изграждане на модул за SMS известяване на клиентите;
* Разработване на мобилна версия на потребителския интерфейс и др.

# 8. Използвана литература

[1]. Aberdeen conference - <http://www.marketwired.com/press-release/harte-hanks-trillium-software-announces-2010-annual-emea-user-conference-nyse-hhs-1320715.html>;

[2]. Trans.Eu - <https://www.trans.eu/bg/>;

[3]. my.tacho.bg - <https://www.tacho.bg/mytacho/>;

[4]. Транспорт Плюс – <https://www.transport-plus.com>;

[5]. Model-View-Controller - [https:/bg.wikipedia.org/wiki/Model-View-Controller](https://bg.wikipedia.org/wiki/Model-View-Controller);

[6]. Софтуерна рамка - [https://bg.wikipedia.org/wiki/Софтуерна\_рамка](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80%D0%BD%D0%B0_%D1%80%D0%B0%D0%BC%D0%BA%D0%B0);

[7]. HTML5 - <https://bg.wikipedia.org/wiki/HTML5>;

[8]. CSS - <https://bg.wikipedia.org/wiki/CSS>;

[9]. JavaScript - <https://bg.wikipedia.org/wiki/JavaScript>;

[10]. JQuery - <https://bg.wikipedia.org/wiki/JQuery>;

[11]. Composer - <https://bg.wikipedia.org/wiki/Composer_(софтуер)>;

[12]. Git - <https://bg.wikipedia.org/wiki/Система_за_контрол_на_версиите>;

[13]. Google Maps API - <https://developers.google.com/maps/>;

[14]. UML - <https://bg.wikipedia.org/wiki/UML>.