



**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**  
**ФАКУЛТЕТ ПО ТРАНСПОРТА**  
катедра Железопътна техника

---

---

# **ДИПЛОМНА РАБОТА**

**ТЕМА: РАЗРАБОТВАНЕ НА ОН-ЛАЙН ИНФОРМАЦИОННА  
СИСТЕМА ЗА РЕЗЕРВАЦИЯ НА МЕСТА В ПЪТНИЧЕСКИЯ  
ЖЕЛЕЗОПЪТЕН ТРАНСПОРТ**

**ДИПЛОМАНТ: Евгений Танев Василев**  
**Специалност: Технология и управление на транспорта**  
**Фак. № 07024436**

Дипломант: .....  
/Евгений Василев/

Научен ръководител: .....  
/доц. д-р инж. Светла Стоилова/

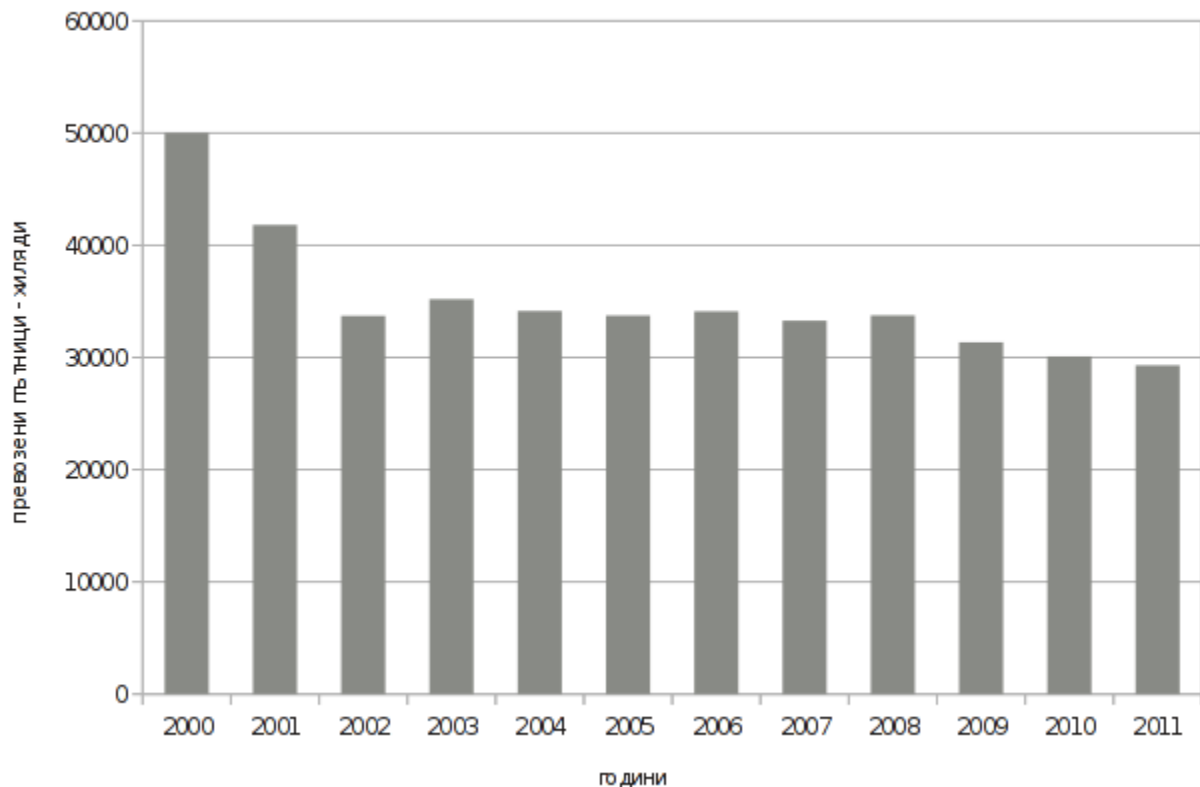
**София, 2012**

# Съдържание

1. Увод.....	2
2. Анализ на европейските практики в онлайн продажбата на електронни билети за железопътен транспорт.....	4
2.1. Онлайн портали на европейски железопътни оператори.....	6
2.2. Изисквания към он-лайн системата.....	9
2.3. Дефиниране на целта и задачите на дипломната работа. ....	11
3. Проектиране и изграждане на система за он-лайн продажба на електронни билети.....	12
3.1. База данни.....	12
3.2. Попълване на БД от изходните данни.....	19
3.3. Разписание.....	24
3.3. Потребителски модул.....	42
3.4. Административен модул.....	50
4. Ефективност на системата.....	58
5. Изводи и препоръки.....	60
5.1. Изводи:.....	61
5.1. Препоръки:.....	61
6. Използвана литература.....	62

# 1. Увод

Използването на пътнически железопътен транспортни в Европейският съюз нараства с около един процент всяка година<sup>1</sup>. Търсенето на този вид транспорт от клиентите се обуславя от неговото удобство, предсказуемост и не на последно място цена. За съжаление пътническият железопътен транспорт в България посреща тази тенденция с неадекватни мерки, които според официалната статистика от Националния статистически институт, води и до намаляване на превозите на пътници всяка година<sup>2</sup>.



Фигура 1: Превозени пътници от железопътен транспорт по години

Причините за спадът на пътническия поток са много и не винаги икономически. Голяма част от загубите на БДЖ са в следствие от недалновидна и нецеленасочена политика по управление на компанията, която все още се държи като монополист в един отворен пазар. Тази заблуда за монополно положение все още се поддържа жива от социалните пакети за пътуване: билети за възрастни граждани и учаци се, преференциалните билети за държавни служители и др. В тази ниша БДЖ наистина има традиционно силно присъствие. За съжаление тази дейност най-често е губеща дори в случаите, когато е субсидирана от държавата. Във средния и високия потребителски сегмент, компанията отстъпва на автобусните превози и личните автомобили. Причината за това е, че тези два сегмента не са толкова чувствителни

1 EU Transport in figures 2011, <http://ec.europa.eu/transport/publications/statistics/doc/2011/pocketbook2011.pdf>

2 Национален статистически институт - 2.1.2.2. Превозени пътници и извършена работа, <http://www.nsi.bg/otrasal.php?otr=6&a1=858&a2=859&a3=892&a4=894>

към цената на билета а определящи за техния избор са най-често по голямата гъвкавост, комфорта и удобството като удобството не се изразява само в удобството на самото пътуване а в цялостния процес на взаимодействие с компанията.

Първата стъпка от това взаимодействие е изборът и закупуването на билет за превоз. Това е стъпката в която клиентът трябва да бъде убеден да използва дадена услуга. Последващите стъпки от превозния процес са не по-малко важни но точно тук инвестицията се капитализира в конкретна покупка.

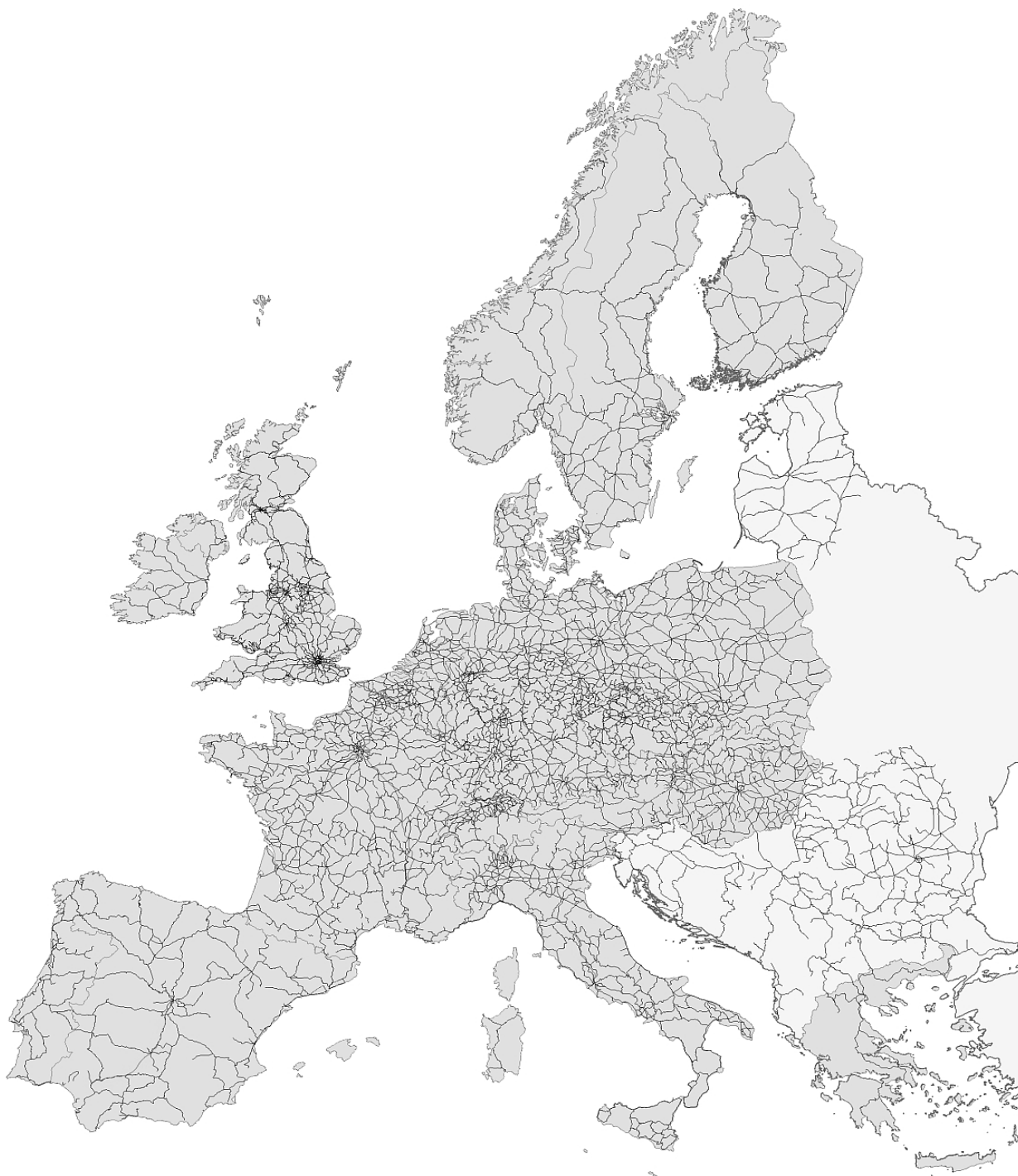
БДЖ прави опит през 2011 година да въведе хибридна система за резервация на билети чрез международните бюра "Рила"<sup>3</sup>, но процесът е тромав и по-скоро затруднява клиента, отколкото да му помага.

Реализацията на он-лайн система за продажба на електронни билети би дало стратегическо предимство на БДЖ в сегмента на сухопътните пътнически превози. Компанията, за разлика от по-малките транспортни фирми, има необходимия обем и разнообразие на транспортни услуги, които да съставят притегателното ядро за потенциални клиенти.

---

3 Катанска, Ц. *БДЖ разработи е-система за поръчка на билети, но без да спестява опашките*, Дневник, [http://www.dnevnik.bg/bulgaria/2010/06/22/921090\\_bdj\\_razraboti\\_e-sistema\\_za\\_poruchka\\_na\\_bileti\\_no\\_bez/](http://www.dnevnik.bg/bulgaria/2010/06/22/921090_bdj_razraboti_e-sistema_za_poruchka_na_bileti_no_bez/)

## 2. Анализ на европейските практики в онлайн продажбата на електронни билети за железопътен транспорт



*Фигура 2: Железопътна мрежа на Европа 2000г.*

Болшинството от европейските железопътни оператори предлагат на клиентите си възможности за закупуване и резервация на билети, он-лайн:

Таблица 1: Он-лайн присъствие на основните железопътните оператори в Европа

Държава	Оператор	Сайт	Онлайн разписание	Онлайн продажба на билети
Австрия	ÖBB	<a href="http://www.oebb.at/">http://www.oebb.at/</a>	да	да
Белгия	NMBS	<a href="http://www.b-rail.be/">http://www.b-rail.be/</a>	да	да
България	БДЖ	<a href="http://www.bdz.bg/">http://www.bdz.bg/</a>	да	не
Германия	Deutsche Bahn	<a href="http://www.bahn.com/">http://www.bahn.com/</a>	да	да
Гърция	OSE	<a href="http://trainose.gr/">http://trainose.gr/</a>	да	да
Дания	DSB	<a href="http://www.dsb.dk/">http://www.dsb.dk/</a>	да	да
Ирландия	CIÉ	<a href="http://www.irishrail.ie/">http://www.irishrail.ie/</a>	да	да
Испания	Renfe	<a href="http://www.renfe.com/">http://www.renfe.com/</a>	да	да
Италия	Trenitalia	<a href="http://www.trenitalia.com/">http://www.trenitalia.com/</a>	да	да
Люксембург	CFL	<a href="http://www.cfl.lu/">http://www.cfl.lu/</a>	да	да
Норвегия	NSB	<a href="http://www.nsb.no/">http://www.nsb.no/</a>	да	да
Полша	PKP	<a href="http://rozklad-pkp.pl/">http://rozklad-pkp.pl/</a>	да	не
Португалия	CP	<a href="http://www.cp.pt/">http://www.cp.pt/</a>	да	да
Румъния	CFR	<a href="http://www.cfr.ro/">http://www.cfr.ro/</a>	да	да
Словакия	ZSSK	<a href="http://www.zssk.sk/">http://www.zssk.sk/</a>	да	да
Словения	SŽ	<a href="http://www.slo-zeleznice.si/">http://www.slo-zeleznice.si/</a>	да	не
Сърбия	ŽS	<a href="http://www.serbianrailways.com/">http://www.serbianrailways.com/</a>	да	да
Финландия	VR	<a href="http://www.vr.fi/">http://www.vr.fi/</a>	да	да
Франция	SNCF	<a href="http://www.sncf.com/">http://www.sncf.com/</a>	да	да
Унгария	MÁV-START	<a href="http://www.mav-start.hu/">http://www.mav-start.hu/</a>	да	не
Холандия	NS	<a href="http://www.ns.nl/">http://www.ns.nl/</a>	да	да
Хърватска	HZ	<a href="http://www.hznet.hr/">http://www.hznet.hr/</a>	да	не
Черна гора	ZCG	<a href="http://www.zcg-prevoz.me/">http://www.zcg-prevoz.me/</a>	да	не
Чехия	Czech Railways	<a href="http://www.cd.cz/">http://www.cd.cz/</a>	да	да
Щвейцария	SBB	<a href="http://www.sbb.ch">http://www.sbb.ch</a>	да	да
Швеция	SJ	<a href="http://www.sj.se/">http://www.sj.se/</a>	да	да

## **2.1. Онлайн портали на европейски железопътни оператори**

Следва кратко представяне на всеки един от сайтовете:

### **2.1.1. Австрия**

Сайта на ÖBB предлага следните възможности за он-лайн закупуване на билети:

- Електронни билети, които се разпечатват на принтер;
- Електронни билети с електронни карти;
- Електронни билети с доставка по пощата;
- SMS билети при които цената се включва в сметката за мобилен телефон или се заплаща на специални терминали на гарите;

Освен различните видове билети, сайта предлага закупуването на карти за отстъпка и различни пакетни оферти за почивки и културни събития.

### **2.1.2. Белгия**

Порталът предлага електронни билети за печат и такива, които се зареждат в електронни карти. Освен билети от сайта могат да бъдат закупени и ограничен брой туристически пакети.

### **2.1.3. България**

Сайтът на БДЖ ЕАД към момента не предлага он-лайн продажба на билети. Разписанието дава информация за някои цени на билети а информацията в него е актуална. Компанията продава билети само в каси на гарите, във влаковете (с повишена цена) и в туристическите бюра Рила. Това силно ограничава достъпността на превозите за чуждестранни пътници.

### **2.1.4. Германия**

Deutsche Bahn предлагат електронни билети за разпечатване, MMS билети и доставка на билети по пощата. Сайта предлага също така оферти за хотели и кола под наем. Предлага голямо разнообразие от карти за намаление и пътуване до международни дестинации.

### **2.1.5. Гърция**

Порталът на Гърция предлага резервиране и заплащане на билети онлайн. За съжаление голяма част от сайта е само на гръцки език, което го прави неудобен за чужденци.

### **2.1.6. Дания**

Датският портал, предлага възможност за използване на различни видове транспорт при избор на маршрут. Интересна възможност е показването на индекса на вредни емисии на всяко пътуване и сравнението при пътуване с автомобил. За съжаление части от сайта са само на датски и въпреки че има ръководство за използване на английски език, използването е затруднено. Закупените електронни билети се разпечатват на принтер и се предоставят за проверка в превозното средство.

### **2.1.7. Ирландия**

В Ирландия се предлагат електронни билети за печат, които в някои случаи са по-евтини от редовните билети, издавани на каса.

### **2.1.8. Испания**

Сайта на Renfe е изцяло на испански. Сайта предлага закупуването на електронни билети он-лайн.

### **2.1.9. Италия**

Продават се електронни билети и свързани с пътуване пакетни оферти. Електронните билети се разпечатват на принтер.

### **2.1.10. Люксембург**

Продават се електронни билети за пътуване до съседни държави.

### **2.1.11. Норвегия**

Билети чрез смартфони. Ако изчакването при прехвърляне надхвърля 30 минути, клиентът получава компенсация. Срещу доплащане се предлагат места със електрическо захранване. Възможност за резервация на хотели. Закупените билети се използват през приложението за смартфон или се получават от автомати разположени на гарите.

### **2.1.12. Полша**

В Полша се предлага само он-лайн резервация на билети чрез сайта <http://www.polrail.com/>, който се явява фирма посредник.

### **2.1.13. Португалия**

В Португалия, могат да бъдат закупени он-лайн електронни билети за разпечатване на принтер. Интересен вариант е и закупуването на билети от банкомати.

### **2.1.14. Румъния**

В Румъния, он-лайн билети могат да бъдат закупени чрез портала <http://www.cfrcalatori.ro/>. За закупените по този начин билети, клиентите получават отстъпка от 5% от стойността на билета.

### **2.1.15. Словакия**

Има възможност за закупуване на он-лайн билети.

### **2.1.16. Словения**

На клиентите е предложено само разписание без възможност за он-лайн поръчка или резервация на билети.



### **2.1.17. Сърбия**

В Сърбия електронни билети могат да бъдат закупени от сайта <http://w3.srbrail.rs/eticketing/> но само за влакове по определени направления. Електронните билети се распечатват на принтер и служат като легитимен превозен документ.

### **2.1.18. Финландия**

Електронните билети във Финландия могат да бъдат распечатани или получени като SMS.

### **2.1.19. Франция**

Във Франция, он-лайн билети се продават през портала <http://www.voyages-sncf.com/>. Билетите могат да се распечатват на принтер или да бъдат доставени по пощата.

### **2.1.20. Унгария**

Предлага се само онлайн разписание и няколко туристически оферти.

### **2.1.21. Холандия**

Предлагат се електронни билети за распечатване

### **2.1.22. Хърватска**

В Хърватска не се предлага он-лайн продажба на билети.

### **2.1.23. Черна гора**

На сайта на „Железопътни превози - Черна гора“ може да се открие базова информация за нормативната уредба в страната и PDF файлове с маршрутите. Он-лайн билети не се предлагат.

### **2.1.24. Чехия**

Предлагат се няколко вида електронни билети, които могат да бъдат распечатани.

### **2.1.25. Швейцария**

В Швейцария се предлагат електронни билети за печат и мобилни билети за смартфони.

### **2.1.26. Швеция**

Предлага се он-лайн закупуване на електронни билети.

## **2.2. Изисквания към он-лайн системата**

След анализ на данните от порталите на националните железопътни оператори се налага изводът че една модерна система трябва да притежава всички или поне част от следните характеристики:

### **2.2.1. Многоезичност**

Немалък процент от потребителите на системи за онлайн продажба на билети са чужденци, за които в някои случаи системата е единствения вариант за достъп до транспортния пазар в страната. Поддръжката на съдържание на няколко езика е времеемко и ресурсоемко като освен това намалява гъвкавостта при поддръжката на актуална информация. Затова изборът на поддръжани езици трябва да се направи на база оценка на потенциалния пазар. Почти всички портали поддържат версия на английски език, който се е наложил де факто като стандарт в Интернет

### **2.2.2. Мобилна версия**

Използването на мобилни устройства бележи ръст в последните 5 години. Тази тенденция е нормална като се има предвид удобството и преносимостта на този тип устройства. Благодарение на широкото им разпространение в тази ниша се оформя все по-голям пазар. Това налага поддръжката на мобилни платформи от системата. Това може да стане по няколко различни начина.

- Основен сайт с адаптивен дизайн който да поддържа както мобилни, така и десктоп устройства. Предимствата на този метод е че се разработва и поддържа само един сайт което чувствително намалява разходите и сроковете за изработка и поддръжка. Недостатъците са че хибридният модел носи със себе си доста ограничения, които макар че в повечето случаи не са непреодолими, затрудняват разработката.
- Отделна версия за мобилни устройства. Този вариант дава свобода в разработката на двата отделни сайта, но прави поддръжката по-сложна и скъпа.
- Мобилни приложения. Плюсовете на този подход са че мобилните приложения обикновено са по-бързи, прехвърлят по-малко количество данни през преносната мрежа и могат да предоставят функционалност, недостъпна през WEB брауъра. Недостатъците са фрагментираната екосистема от мобилни платформи, за да е ефективна подобна стратегия трябва да се разработят приложения поне за iOS и Android платформите, които след това трябва и да се поддържат.
- Външни разработчици. Тъй като софтуерните разработки, рядко са приоритет на железопътните оператори, задачата по разработка на мобилна версия на системата може да се отстъпи на трети лица. Най-добрият начин за това е като се предостави програмен интерфейс към системата т.нар. API, който да може да се ползва свободно от разработчици, който да разработят собствен програмен продукт на база системата.

### **2.2.3. Разнообразни методи за плащане**

Успешното плащане е една от основните цели на системата, затова поддръжката на разнообразни кредитни и дебитни карти е важно за използваемостта ѝ. В България възможните варианти за разплащане са:

- ePay (<http://www.epay.bg/>)- поддържат болшинството от дебитни карти издавани в страната, изискват потребителя да има регистрация;
- eBg (<http://www.ebg.bg/>) - поддържат плащане с български дебитни карти и виртуални кредитни карти;
- EasyPay (<http://www.easypay.bg/>) - комбинирано онлайн задължаване и офлайн заплащане. Популярен вариант за потребители без банкови карти;
- Виртуален POS - предлага се от няколко банки, приемат се плащания с международни кредитни карти, обикновено не се изисква регистрация на потребителя;
- Банков превод - предлага се от всички банки;
- наложен платеж - широко използван метод за плащане в България, предлага се от куриерските фирми, има приложение само при доставка на билети;
- PayPal (<http://www.paypal.com>) - международен оператор на картови разплащания, удобен за използване от международни клиенти;
- SMS плащания - имат фиксирана цена за обработка на транзакция така че не са ефективни за много малки суми. Изискват подписване на договори със всички оператори и разработка на обща система с тях. Удобни за потребителите, тъй като изискват само мобилен телефон без претенции към операционната му система;

От този списък трябва да се избере не само един оператор а всички рентабилни опции, тъй като различните потребители разполагат с различни платежни средства а и имат различни предпочитания към използваните услуги.

#### **2.2.4. Превозни документи**

Създаването на валидируем превозен документ е една от основните цели на системата. В практиката се използват следните варианти на документи:

- Електронен билет - един от най-лесните за издаване документи, разчита на разпечатване на генерирана от системата бланка, която включва информация за пътуването, цената на билета и служебна информация, която да удостовери валидността на документа. Най-често за проверка се използват баркодове, разпечатани на билета, които се проверяват от кондуктора. Също така могат да се използват QR кодове или прост идентификационен номер който да се въведе ръчно. За да се използва метода, кондукторите трябва да са екипирани с устройства с памет баркод четци за да могат да проверят дали билетът е уникален, издаден за този влак и тази дата на пътуване. Информацията в устройствата може да се синхронизира след приключване на резервационния период и преди потеглянето на влака.
- Мобилен билет - мобилен билет се реализира чрез специално приложение за смартфон, което в се явява приемник на данните за билета. При този вариант се избягва хартиения носител а проверката се извършва най-често чрез QR-код.
- SMS билет - клиента получава уникален код от системата изпратен под формата на SMS, кодът се проверява ръчно от кондуктора;
- Електронни карти - обикновено магнитна картата, която идентифицира уникално клиента. При покупка на билет, системата указва че с картата може да се пътува за определена дата по дадено направление и тази информация се проверява от кондуктора;

### **2.2.5. Обвързка с други услуги и видове транспорт**

Нуждата от транспортна услуга рядко е самоцелна и много често е част от по-широки намерения на клиента. Предлагането на допълнителни услуги като резервация на хотели и наемане на автомобил както и обвързката с останалите видове транспорт създава голяма добавена стойност към системата. Цялостното обслужване на клиента създава в него доверие а също така може да доведе до по-ниска обща цена на услугата както и до по-висок оборот на оператора. Този похват се използва особено широко при въздушните превозвачи но няма причина да не е приложим и при сухопътните превози.

### **2.2.6. Достъп до пълна и актуална информация**

Колкото и фундаментално да изглежда това условие, все пак си заслужава да бъде спомената, защото често бива забравено или оставено на заден план. Потребителят предпочита да е информиран за всички детайли, особено ако ще използва дадена услуга за първи път. За целта е необходимо да има достъп до пълно и точно разписание както и всички варианти за билети които може да получи. Добре е да се представят плюсовете на всеки вид билет за да може, клиента да направи информиран избор.

### **2.2.7. Бонус програми**

Част от операторите предлагат намаление на цената на билета ако е закупен онлайн. Тази разлика се обуславя от по-ниските разходи за продажбата но не трябва да е единствения стимул за клиента. Бонус програми, натрупани километри пътуване, намаления за чести пътувания са само някой от вариантите за привличане на лоялни клиенти.

## **2.3. Дефиниране на целта и задачите на дипломната работа.**

Целта на дипломната работа е да се разработи система за резервация на места за пътнически железопътни превози.

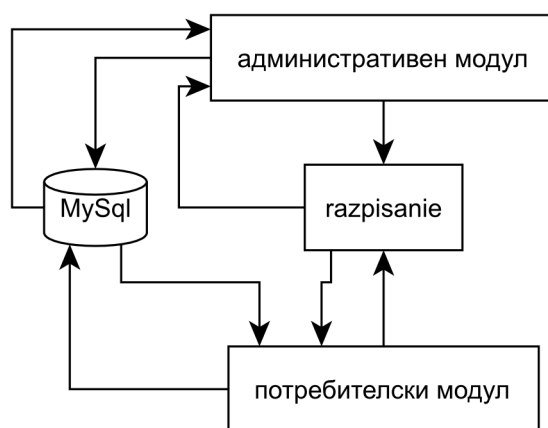
Задачите на дипломната работа са:

1. Да се направи анализ на системата за резервация на места в европейските железопътни администрации;
2. Да се разработи алгоритъм на информационна система за резервация на места;
3. Да се направи апробация на системата;
4. Да се направят изводи и препоръки.

### 3. Проектиране и изграждане на система за он-лайн продажба на електронни билети

Системата се състои от четири функционални модула. Потребителският и административен модул са WEB базирани реализирани на езикът за програмиране PHP, като е използвана и библиотеката CodeIgniter. Системата използва WEB сървър Apache2 но би могла да работи и с nginx, lighthttpd или друг сървър, който поддържа mod\_php или php-cgi.

Целите, които си поставя системата е да реализира базовата функционалност необходима за успешно обслужване на клиенти. Разбира се една такава система може да се усложнява и развива почти безгранично в зависимост от конкретната обстановка и плановете на мениджмънта.



Фигура 3: Функционални модули на системата

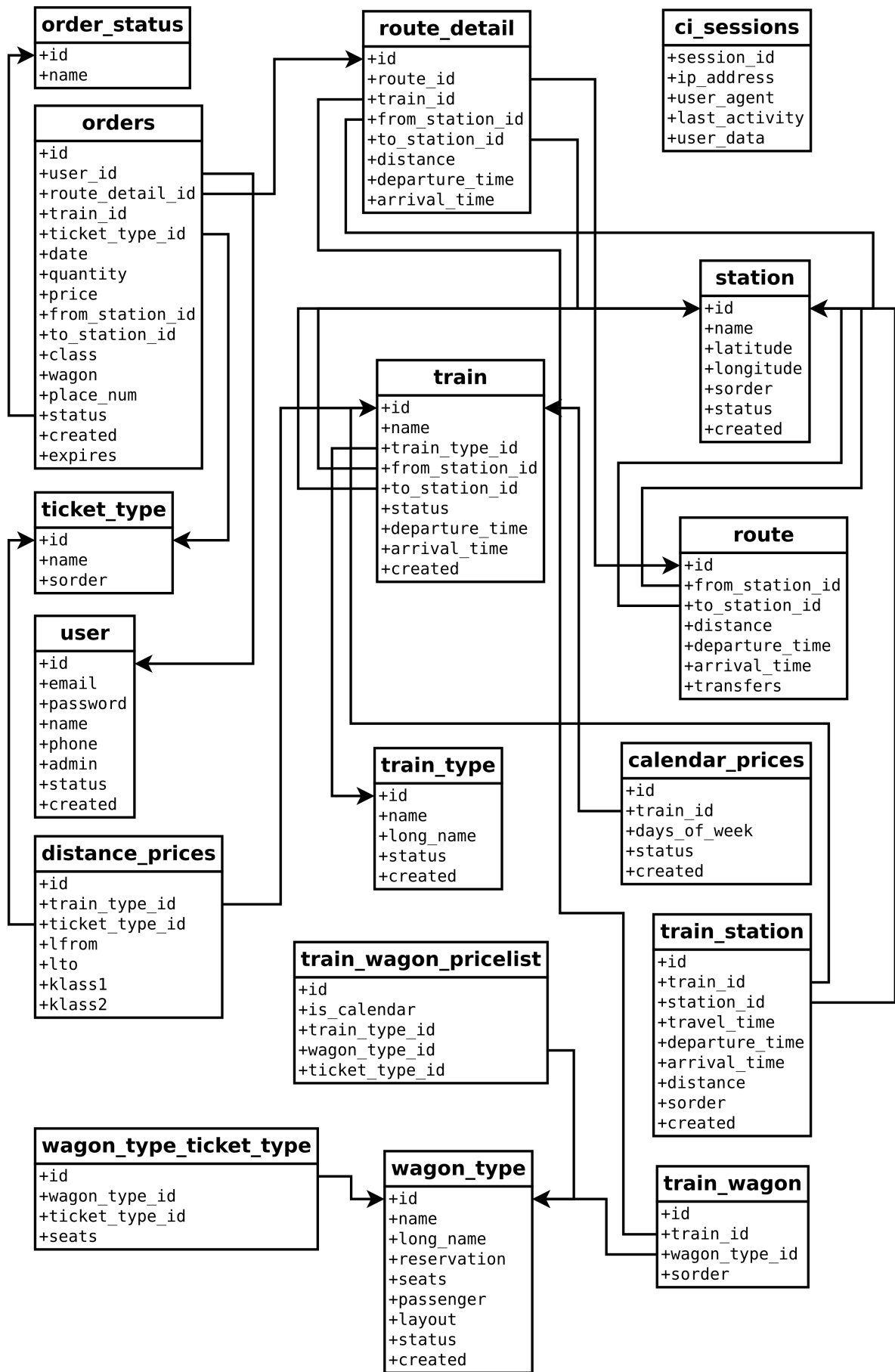
Модулите на системата трябва да са самостоятелни и да, които комуникират по утвърдени комуникационни протоколи избрани на база ефективност. Това би улеснило разширяването на системата при нужда от поемане на повече потребителски трафик.

Първата стъпка в разработката е проектиране на структурата на базата данни.

#### 3.1. База данни

Сайта използва MySQL база данни за съхранение и обработка на данните. Изборът на MySQL е обусловен от широката употреба на тази СУБД и сравнително лесното ѝ използване както от професионалисти така и от начинаещи. Системата позволява миграция и към друга СУБД, примерно PostgreSQL със сравнително малки модификации.

Данните са разпределени в следните таблици:



Фигура 4: Схема на базата данни

Следва кратко описание на структурата на всяка таблица:

### **calendar\_prices**

Списък на календарните влакове.

- **id** - идентификатор;
- **train\_id** - идентификатор на превозно средство;
- **day\_of\_week** - ден от седмицата в РНР формат (1 - понеделник, 7 - неделя);
- **status** - статус на записа;
- **created** - дата и час на добавяне на записа

### **ci\_sessions**

Таблица за съхранение на потребителските сесии.

- **session\_id** - идентификатор на сесията;
- **ip\_address** - IP адрес на потребителя;
- **user\_agent** - User-Agent на браузъра;
- **last\_activity** - последна активност на потребителя;
- **user\_data** - потребителски данни;

### **distance\_prices**

Тарифи за разстояния.

- **id** - идентификатор;
- **train\_type\_id** - идентификатор на типа превозно средство;
- **ticket\_type\_id** - идентификатор на типа тарифа;
- **lfrom** - долна граница на интервала по разстояние;
- **lto** - горна граница на интервала по разстояние;
- **klass1** - цена за билет клас 1;
- **klass2** - цена за билет клас 2;

### **order\_status**

Списък с възможни статуси на поръчка.

- **id** - идентификатор;
- **name** - име на статуса;

## orders

Клиентски поръчки и резервации.

- **id** - идентификатор;
- **user\_id** - идентификатор на потребител;
- **route\_detail\_id** - идентификатор на сегмент от маршрут;
- **train\_id** - идентификатор на превозно средство;
- **ticket\_type\_id** - идентификатор на тарифа;
- **date** - дата на пътуване;
- **quantity** - количество билети;
- **price** - единична цена;
- **from\_station\_id** - идентификатор на начална гара;
- **to\_station\_id** - идентификатор на крайна гара;
- **class** - клас билет;
- **wagon** - номер на вагон (0 ако е без резервация);
- **place\_num** - номер на запазено място (0 ако е без резервация);
- **status** - статус на поръчката;
- **created** - дата и час на поръчване;
- **expires** - дата и час на изтичане на резервацията;

## route

Изчислен маршрут между две гари.

- **id** - идентификатор;
- **from\_station\_id** - идентификатор на начална гара;
- **to\_station\_id** - идентификатор на крайна гара;
- **distance** - разстояние (в стотици метри);
- **departure\_time** - час на заминаване (в минути след 0:00 часа);
- **arrival\_time** - час на пристигане (в минути след 0:00 часа);
- **transfers** - брой прехвърляния по маршрута;



## route\_detail

Сегмент от изчислен маршрут.

- **id** - идентификатор;
- **route\_id** - идентификатор на маршрут;
- **train\_id** - идентификатор на превозно средство;
- **from\_station\_id** - идентификатор на начална гара;
- **to\_station\_id** - идентификатор на крайна гара;
- **distance** - разстояние (в стотици метри);
- **departure\_time** - час на заминаване (в минути след 0:00 часа);
- **arrival\_time** - час на пристигане (в минути след 0:00 часа);

## station

Списък гари.

- **id** - идентификатор;
- **name** - име на гарата;
- **latitude** - географска ширина;
- **longitude** - географска дължина;
- **sorder** - ред на сортиране;
- **status** - статус на записа;
- **created** - дата и час на създаване на записа;

## ticket\_type

Списък тарифи.

- **id** - идентификатор;
- **name** - име на тарифата;
- **sorder** - ред на сортиране;

## train

Превозни средства.

- **id** - идентификатор;
- **name** - име на превозното средство;
- **train\_type\_id** - тип на превозното средство;
- **from\_station\_id** - идентификатор на начална гара;
- **to\_station\_id** - идентификатор на крайна гара;
- **status** - статус на превозното средство;
- **departure\_time** - час на заминаване (в минути след 0:00 часа);
- **arrival\_time** - час на пристигане (в минути след 0:00 часа);
- **created** - дата и час на създаване на записа;

## train\_station

Точки от маршрута на влак.

- **id** - идентификатор;
- **train\_id** - идентификатор на превозно средство;
- **station\_id** - идентификатор на гара;
- **travel\_time** - време за пътуване (в минути);
- **departure\_time** - час на заминаване (в минути след 0:00 часа);
- **arrival\_time** - час на пристигане (в минути след 0:00 часа);
- **distance** - разстояние (в стотици метра);
- **sorder** - хронологичен ред;
- **created** - дата и час на създаване на записа;

## train\_type

Списък на типове превозни средства.

- **id** - идентификатор;
- **name** - кратко име на типа;
- **long\_name** - описание на типа;
- **status** - статус;
- **created** - дата и час на създаване на записа;

## **train\_wagon**

Подвижен състав на превозните средства.

- **id** - идентификатор;
- **date** - дата на състава (0 ако е базов);
- **train\_id** - идентификатор на превозно средство;
- **wagon\_type\_id** - идентификатор на тип вагон;
- **sorder** - пореден номер;

## **train\_wagon\_pricelist**

Конфигурации за комбинации влак/вагон/тарифа.

- **id** - идентификатор;
- **is\_calendar** - календарен (0-не, 1-да);
- **train\_type\_id** - идентификатор на тип превозно средство;
- **wagon\_type\_id** - идентификатор на тип вагон;
- **ticket\_type\_id** - идентификатор на тарифа;

## **user**

Регистрирани потребители.

- **id** - идентификатор;
- **email** - e-mail адрес на потребителя;
- **password** - парола в криптиран формат;
- **name** - име на потребителя;
- **phone** - телефон на потребителя;
- **admin** - администраторски права (0 -не, 100 - да);
- **status** - статус на профила;
- **created** - дата и час на създаване на записа;

## wagon\_type

Видове вагони.

- **id** - идентификатор;
- **name** - име на вагона;
- **long\_name** - описание на вагона;
- **reservation** - възможност за резервация на място (0-не, 1-да);
- **seats** - брой места във вагона;
- **passenger** - пътнически? (0-не, 1-да);
- **layout** - схема на местата във вагона;
- **status** - статус на вагона;
- **created** - дата и час на създаване на записа;

## 3.2. Попълване на БД от изходните данни

Част от изходните данни са в електронен формат, което ги прави идеални кандидати за автоматичен импорт в системата.

Файлът **2010-TIMETABL.TXT** съдържа информация за движението на всички влакове в текстов формат. Кодировката на файла е windows-1251, затова първо трябва да бъде преобразуван в utf-8 кодировка която се използва широко в съвременните мултиезични системи. За целта се използва служебната програма **iconv**:

```
$iconv -f cp1251 -t utf8 2010-TIMETABL.TXT > timetable.txt
```

Самият импорт се извършва от конзолен PHP скрипт за обработка на текста, като резултата от скрипта е SQL скрипт, който директно може да се изпълни на SQL сървъра:

Скрипта обработва файла за разписание и генерира SQL скрипт от него:

### import.php

```
<?php
mb_internal_encoding("UTF-8");
class Import {

    var $in_train = FALSE;
    var $train = array();
    var $stations = array();
    var $trains = array();
    var $train_types = array();

    function do_import($file_name){
        if (!file_exists($file_name)) {
            die('File '.$file_name.' not found!'.PHP_EOL);
        }
        $handle = fopen($file_name, 'r');
        if ($handle) {
            while (($buffer = fgets($handle, 4096)) !== false) {
                $this->processLine($buffer);
            }
        }
    }
}
```

```

    }
    if (!feof($handle)) {
        echo "Error: unexpected fgets() fail\n";
    }
    fclose($handle);
}
if ($this->train) {
    $this->saveTrain();
}
//print_r($this->trains);
}

function processLine($line) {
    if (trim($line)) {
        if ($line[0] != ' ') {
            if ($this->train) {
                $this->saveTrain();
            }
            $this->train = explode(' ', $line);
            //echo 'header' . $line . PHP_EOL;
        } else {
            $station['len'] = mb_substr($line, 0, 7);
            $station['speed'] = (int)mb_substr($line, 8, 4);
            $station['name'] = trim(mb_substr($line, 12, 15));
            $station['minutes'] = trim(mb_substr($line, 28, 2));
            $station['minutes2'] = mb_substr($line, 31, 2);
            $station['arrival_time'] = mb_substr($line, 34, 5);
            $station['wait'] = mb_substr($line, 40, 3);
            $station['departure_time'] = mb_substr($line, 44, 5);
            $station['unknown'] = mb_substr($line, 50, 4);
            $this->train['stations'][] = $station;
        }
    }
}

function getElement(&$array, $name) {
    $index = array_search($name, $array);
    if ($index === FALSE) {
        $array[] = $name;
        return count($array)-1;
    }
    return $index;
}

function timeToMinutes($time) {
    if ($time) {
        list($min, $sec) = explode(':', trim($time));
        return $sec+($min*60);
    }
    return 0;
}

function saveTrain() {
    $train = array(
        'train_type' => $this->getElement($this->train_types, $this->train[0]),
        'name' => $this->train[1],
    );
    $train['stations'] = array();
    foreach($this->train['stations'] as $station) {
        $name = mb_convert_case($station['name'], MB_CASE_TITLE);
    }
}

```

```

    $train['stations'][] = array(
        'station_id' => $this->getElement($this->stations, $name),
        'time' => $station['minutes']?$station['minutes']:0,
        'arrival_time' => $this->timeToMinutes($station['arrival_time']),
        'departure_time' => $this->timeToMinutes($station['departure_time']),
        'len' => $station['len']*10,
    );
}
$this->trains[] = $train;
}

function generateSQL(){
    echo "SET names utf8;\n";
    foreach($this->stations as $id => $station) {
        echo sprintf("INSERT INTO station (id, name) VALUES (%d, '%s');\n", $id+1, $station);
    }
    foreach($this->train_types as $id => $tt) {
        echo sprintf("INSERT INTO train_type (id, name) VALUES (%d, '%s');\n", $id+1, $tt);
    }
    foreach($this->trains as $train_id => $train) {
        $from_id = $train['stations'][0]['station_id'];
        $to_id = $train['stations'][count($train['stations'])-1]['station_id'];
        $departure_time = $train['stations'][0]['departure_time'];
        $arrival_time = $train['stations'][count($train['stations'])-1]['arrival_time'];
        echo sprintf("INSERT INTO train (id, name, train_type_id, from_station_id,
%d);\n",
            $train_id+1, $train['name'], $train['train_type']+1, $from_id+1,
            $to_id+1, $departure_time, $arrival_time);
        $num = 1;
        foreach ($train['stations'] as $station) {
            echo sprintf("INSERT INTO train_station (train_id, station_id, travel_time,
%d);\n",
                $train_id+1, $station['station_id']+1, $station['time'],
                $station['arrival_time'], $station['departure_time'], $station['len'], $num);
            $num++;
        }
    }
}

$file_name = '../_data/timetable.txt';
$i = new Import();
$i->do_import($file_name);
$i->generateSQL();

```

Информацията за подвижния състав се намира във файла **2010-TABLE1.RTF**. Директната обработка на Rich Text Format е затруднена, затова файлът се експортира в HTML формат, който след това се изчиства от HTML таговете и бива преобразуван до файла **processed.txt**, съдържащ информацията за всеки влак на самостоятелен ред. Файлът се обработва от скрипта **wagons.php**, който отново генерира SQL скрипт, готов за изпълнение на MySQL сървъра:

### wagons.php

```
<?php
mb_internal_encoding("UTF-8");
$re = '/([\d])([A-ZAB]+)/u';
$lokre = '/(лок[\d]+)/u';

$text = file_get_contents('../../_data/vagoni/processed.txt');

$arr = explode(PHP_EOL, $text);

$res = array();

$wagons = array();

function fix($t) {
    return str_replace(
        array('A', 'B'),
        array('A', 'B'),
        $t
    );
}

foreach ($arr as $row) {
    $t = explode('|', $row);
    if (isset($t[1])) {
        $train_num = $t[1];
        $tr = array();
        if (preg_match($lokre, $row, $m)) {
            $lok = $m[1];
            if (!in_array($lok, $wagons)) {
                $wagons[] = $lok;
            }
            $tr[$lok] = 1;
        }
        if (preg_match_all($re, $row, $m, PREG_SET_ORDER)) {
            foreach ($m as $mrow) {
                $w = fix($mrow[2]);
                if (!isset($tr[$w])) {
                    $tr[$w] = $mrow[1];
                }
                if (!in_array($w, $wagons)) {
                    $wagons[] = $w;
                }
            }
        }
        // print_r($tr);
    }
    $res[$train_num] = $tr;
}
}
```

```

sort($wagons);

foreach ($wagons AS $id => $row) {
    printf("INSERT INTO wagon_type (id, name) VALUES (%d, '%s');".PHP_EOL, ($id+1), $row);
}

foreach ($res as $train_num => $lwagons) {
    $n = 1;
    foreach ($lwagons as $type => $cnt) {
        for ($i = 1; $i <= $cnt; $i++) {
            $ndx = array_search($type, $wagons);
            printf("INSERT INTO train_wagon (train_id, wagon_type_id, sorder)
                VALUES (COALESCE((SELECT id FROM train WHERE name='%s'), 0), %d,
%d);".PHP_EOL,
                $train_num, $ndx+1, $n);
            $n++;
        }
    }
}
?>

```

Останалите данни: тарифната информация [2], GPS координати на гарите и подробностите за подвижния състав, се заложена ръчно в SQL скрипта **populate.sql**, който е с големина над 2000 реда.

Целият процес по изграждане и попълване на данните в MySQL е автоматизиран чрез следния файл:

### Makefile

```

DBHOST = localhost
DBUSER = username
DBDB = database
DBPASS = password

```

```
all: clean_database create_database populate
```

```
create_database:
```

```
mysql -u $(DBUSER) -h $(DBHOST) --password=$(DBPASS) $(DBDB) < create.sql
```

```
clean_database:
```

```
mysql -u $(DBUSER) -h $(DBHOST) --password=$(DBPASS) $(DBDB) < clean.sql
```

```
populate:
```

```
php import.php | mysql -u $(DBUSER) -h $(DBHOST) --password=$(DBPASS) $(DBDB)
php wagons.php | mysql -u $(DBUSER) -h $(DBHOST) --password=$(DBPASS) $(DBDB)
mysql -u $(DBUSER) -h $(DBHOST) --password=$(DBPASS) $(DBDB) < populate.sql
```

Процесът се стартира чрез командата **make** при което последователно се извършват следните задачи:

- Унищожаване на всички таблици в базата данни (ако съществуват);
- Създаване на всички таблици в базата данни;
- Стартиране на **import.php** и изпълнение на резултата от скрипта на MySQL сървъра.



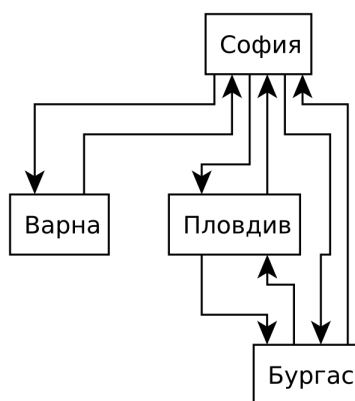
- Стартиране на **wagons.php** и изпълнение на резултата от скрипта на MySQL сървъра.
- Изпълнение на **populate.sql** на MySQL сървъра.

Задачите по унищожаване на данните, създаване на таблиците и попълване на данните могат да се стартират и индивидуално по желание.

### 3.3. Разписание

Модулът **razpisanie** изпълнява една от двете основни цели на системата, да изготви близки до оптималния, маршрути между две гари или спирки. Изискването към модула е да изпълнява търсенето възможно най-бързо и точно.

Железопътната система може да се представи като симетричен насочен граф където всяка гара е връх от графа а всеки маршрут между две гари като насочена дъга с тегло разстоянието между двете гари по маршрута:



Фигура 5: Примерен граф на железопътна мрежа

Има множество алгоритми за намиране на пътища в граф. В конкретния случай ще използваме модификация на метода описан в DRUMURI OPTIME ÎN GRAFURI ORAR[cozac]. По същество методът се състои в оптимизирано представяне на графа в паметта и търсене на близки до оптималния, пътища в него.

Входящите данни за модула са списък със следното съдържание:

- идентификатор на превозно средство;
- име на превозно средство;
- идентификатор на гара;
- име на гара;
- час на пристигане;
- час на заминаване;
- разстояние от началната точка на маршрута;

Часовете на пристигане и заминаване са представени чрез брой минути от 0:00 часа а разстоянието е в стотици метри. Целта на тези преобразувания е да използваме цели числа за

представяне на данните като по този начин спестяваме памет и изчислително време. За разделител между колоните използваме стойността на константата **SEPARATOR**, дефинирана в **constants.go**, в случая вертикална черта "|";

### Опростени данни за движение по осма линия

```
120|7620|1|София|0|435|0
120|7620|215|Враца|542|544|1015
120|7620|341|Видин|725|0|2652
121|7621|341|Видин|0|365|0
121|7621|215|Враца|528|530|1637
121|7621|1|София|635|0|2653
122|7622|1|София|0|745|0
122|7622|215|Враца|868|870|1036
122|7622|341|Видин|1060|0|2673
124|7623|341|Видин|0|765|0
124|7623|215|Враца|948|950|1637
124|7623|1|София|1075|0|2673
125|7624|1|София|0|985|0
125|7624|215|Враца|1082|1084|1015
125|7624|341|Видин|1250|0|2652
126|7625|341|Видин|0|968|0
126|7625|215|Враца|1165|1167|1637
126|7625|1|София|1275|0|2653
127|7630|1|София|0|1150|0
127|7630|215|Враца|1269|1271|1036
127|7630|350|Лом|1377|0|2031
128|7631|350|Лом|0|335|0
128|7631|215|Враца|451|454|995
128|7631|1|София|588|0|2031
```

При стартиране на модула се изпълнява функцията **StartServer**:

```
func StartServer() {
    // Създаване на нова глобална променлива за данните
    razpisanie = NewRazpisanie()
    // Зареждане на входящите данни
    razpisanie.LoadFromFile("data/paths.txt")
    // Обработка на данните
    razpisanie.Process()

    log.Println("Server operational")
    // Дефиниране на услугите на сървъра
    http.Handle("/route", http.HandlerFunc(routeHandler))
    http.Handle("/distance", http.HandlerFunc(distanceHandler))
    // Стартиране на сървъра
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

**Razpisanie** е основната функционална структура на модула. Затова променлива от тип **Razpisanie** е дефинирана като глобална за целия модул.

```
// Основна функционална структура
type Razpisanie struct {
    tempList TempList // Временен списък за данните при зареждане
    stations *Dictionary // Речник на гарите
    trains *Dictionary // Речник на превозните средства
    floydGraph FloydGraph // Матрица на теглата
    dfsGraph DFSGraph // Разширен граф
```

```

}
Зареждането на входящите данни се осъществява от функцията LoadFromFile:
// Зареждане на данните от файл
func (raz *Razpisanie) LoadFromFile(file_name string) {
    // Отваряне на файла за четене
    f, err := os.Open(file_name)
    if err != nil {
        // В случай на грешка прекъсване работата на модула
        panic(err)
    }
    // Заявка за затваряне на файла след приключване на работа с него
    defer f.Close()

    // Обект за буферирано четене от файлов дескриптор
    reader := bufio.NewReader(f)

    line := 1 // Брояч на редове
    for {
        // Прочитане на ред от файла
        rline, _, err := reader.ReadLine()

        if err != nil {
            break // Приключване на четенето
        }
        // Разделяне на реда на елементи
        array := strings.Split(string(rline), SEPARATOR)

        train_id, _ := strconv.Atoi(array[0]) // Идентификатор на превозно средство
        train_name := array[1] // Име на превозно средство
        station_id, _ := strconv.Atoi(array[2]) // Идентификатор на гара
        station_name := array[3] // Име на гара

        var row TRow // Нов ред-контейнер за временния списък

        row.arrival, _ = strconv.Atoi(array[4]) // Час на пристигане
        row.departure, _ = strconv.Atoi(array[5]) // Час на заминаване
        row.length, _ = strconv.Atoi(array[6]) // Дължина пътуването от началната точка

        // Добавяне на името и идентификатора на гара в речника за гари и
        // указване на вътрешния идентификатор във реда-контейнер
        row.station = raz.stations.Add(station_id, station_name)
        // Добавяне на името и идентификатора на превозно средство в речника за
        превозни
        // средства и указване на вътрешния идентификатор във реда-контейнер
        row.train = raz.trains.Add(train_id, train_name)

        // Добавяне на реда-контейнер във временния списък
        raz.tempList = append(raz.tempList, &row)
        line++ // Увеличаване на брояча на редове
    }
    log.Printf("Loaded %d lines from \"%s\"\n", line, file_name)
}

```

Имената и идентификаторите на гарите и превозните средства се съхраняват в два речника за да се използв по-малко памет за съхранението на данните. Това също така опростява инициализирането на структурите за търсене в последствие. В този момент данните вече са налични но не са в удобен за употреба формат. Обработката им се поема от функцията **Process**:

```

// Обработка на временния списък
func (raz *Razpisanie) Process() {
    // Създаване на структурата за матрица на теглата
    (*raz).floydGraph = *(NewFloydGraph((*raz).stations.Len()))
    // Създаване на структурата за разширен граф
    (*raz).dfsGraph = *(NewDFSGraph((*raz).stations.Len()))

    // Инициализация на променливите
    var (
        last_station int = -1
        last_train   int = -1
        last_distance = 0
        distance     = 0

        node1 *DFSNode = nil
        node2 *DFSNode = nil
    )

    // Обработка на входящите данни
    for _, row := range (*raz).tempList {
        if row.length == 0 {
            last_distance = 0
        }
        distance = row.length - last_distance
        //Floyd
        if last_station != -1 && row.train == last_train {
            if row.length == 0 {
                last_distance = 0
            }
            // Добавяне на разстоянието в матрицата на теглата
            (*raz).floydGraph.Add(last_station, row.station, distance)
            last_distance = row.length
        }
        //DFS
        if row.arrival != 0 {
            // Добавяне на нов връх от тип "пристигане"
            node1 = (*raz).dfsGraph.AddTrainStation(row.train, row.station, row.arrival, ARRIVAL)
        } else {
            node1 = nil
        }
        if node1 != nil && node2 != nil {
            // Добавяме дъга за продължаване на пътуването
            node2.addArc(node1, distance, CONTINUE)
        }
        if row.departure != 0 {
            // Добавяне на нов връх от тип "заминаване"
            node2 = (*raz).dfsGraph.AddTrainStation(row.train, row.station, row.departure,
DEPARTURE)
        } else {
            node2 = nil
        }
        if node1 != nil && node2 != nil {
            // Добавяне дъга за продължаване на пътуването
            node1.addArc(node2, 0, CONTINUE)
        }

        last_station = row.station
        last_train   = row.train
    }
}

```

```

// Обработка на матрица на теглата
(*raz).floydGraph.Resolve()
// Обработка на разширен граф
(*raz).dfsGraph.Init()
}

```

За реализация на търсенето се използват два алгоритъма от теория на графите: алгоритъм на **Фloyd-Уоршал**[3] и **търсене в дълбочина на граф**[3]

В конкретния случай, алгоритъмът на **Фloyd-Уоршал** се използва за определяне на минималните разстояния между всеки две точки в граф. Тези разстояния ще използваме по-късно при търсенето на маршрути.

Дефинира се матрица, представена от двумерен масив (в конкретната имплементация на езика **Go** се използват отрязъци за да се оптимизира количество заделена памет):

```

type FloydGraph [][]int

```

Следва инициализация на матрицата с размер броят на гарите и запълване на всички клетки със константната стойност **INFINITY**, дефинирана във файла **constants.go**.

```

// Създаване на нова матрица на теглата и инициализиране
// на всички клетки с INFINITY
func NewFloydGraph(num_stations int) *FloydGraph {
// създаване на първото измерение на матрицата
fg := make(FloydGraph, num_stations)
for x := range fg {
// създаване на второто измерение
fg[x] = make([]int, num_stations)
for y := range fg[x] {
// инициализация на клетките
fg[x][y] = INFINITY
}
}
return &fg
}

```

Всички известни разстояния между гари се добавят в матрицата:

```

// Обработка на входящите данни
for _, row := range (*raz).templList {
if row.length == 0 {
last_distance = 0
}
distance = row.length - last_distance
//Floyd
if last_station != -1 && row.train == last_train {
if row.length == 0 {
last_distance = 0
}
// Добавяне разстоянието в матрицата на теглата
(*raz).floydGraph.Add(last_station, row.station, distance)
last_distance = row.length
}

// ....

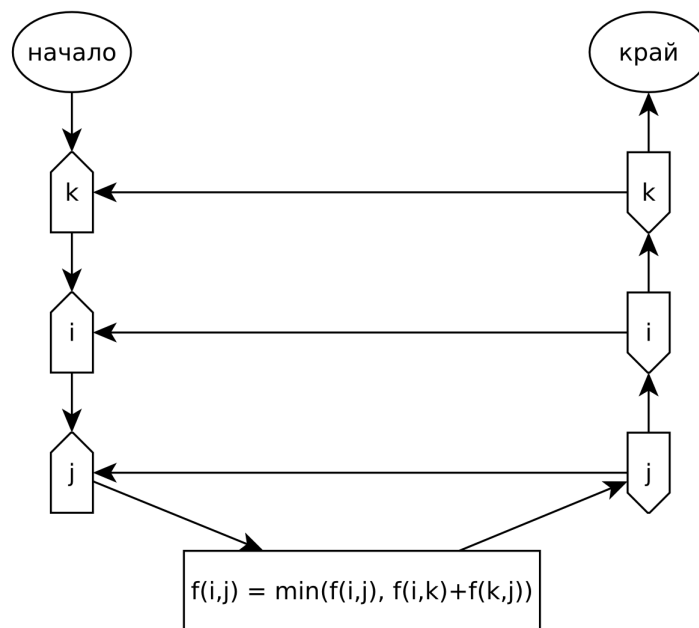
last_station = row.station
last_train = row.train
}

```

Следва изпълнение на самият алгоритъм на **Флойд-Уоршал**:

```
// връщане на по-малкото от две цели числа
func MinInt(v1, v2 int) int {
    if v1 <= v2 {
        return v1
    }
    return v2
}

// Определяне на минимални пътища между върхове на граф
// по алгоритъм на Флойд-Уоршал
func (fg *FloydGraph) Resolve() {
    n := len(*fg) // Размерността на матрицата
    for k := 0; k < n; k++ {
        for i := 0; i < n; i++ {
            for j := 0; j < n; j++ {
                if i == j {
                    (*fg)[i][j] = 0 // Разстоянието от гарата до самата нея винаги е 0
                } else {
                    (*fg)[i][j] = MinInt((*fg)[i][j], (*fg)[i][k]+(*fg)[k][j])
                }
            }
        }
    }
}
```



Фигура 6: Алгоритъм на Флойд-Уоршал

Състоянието на матрицата преди и след обработката са показани съответно в **Таблица 2** и **Таблица 3**:

Таблица 2: Матрица на теглата преди изчисляване на разстоянията

	София	Враца	Видин	Лом
София	999999	1036	999999	999999
Враца	1036	999999	1637	995
Видин	999999	1637	999999	999999
Лом	999999	995	999999	999999

стойностите са в стотици метри

Таблица 3: Матрица на теглата след изчисляване на разстоянията

	София	Враца	Видин	Лом
София	0	1036	2673	2031
Враца	1036	0	1637	995
Видин	2673	1637	0	2632
Лом	2031	995	2632	0

стойностите са в стотици метри

След обработката на матрицата, изчислените разстояния между всеки две гари са достъпни чрез функцията **GetDistance**:

```
// Разстояние между гари x и y
func (fg *FloydGraph) GetDistance(x, y int) int {
    return (*fg)[x][y]
}
```

За същинското търсене на маршрути се използва алгоритъмът търсене в дълбочина (**Deep First Search**).

Търсенето в дълбочина на граф е алгоритъм измислен още през 19 век като метод за откриване на пътища в лабиринт. Алгоритъмът е рекурсивен и се състои в последователно обхождане на всички дъги на даден връх докато не се достигне до желания краен връх.

Предимството на този алгоритъм е относително добрата му скорост на търсене и лесната приложимост при търсене в критическа секция.

Графът е представен от следните структури:

```
// Разширен граф
type DFSGraph []*NodeColumn

// Колона на гара
type NodeColumn []*DFSNode

// Връх
type DFSNode struct {
    station int // гара
    train int // влак
}
```

```

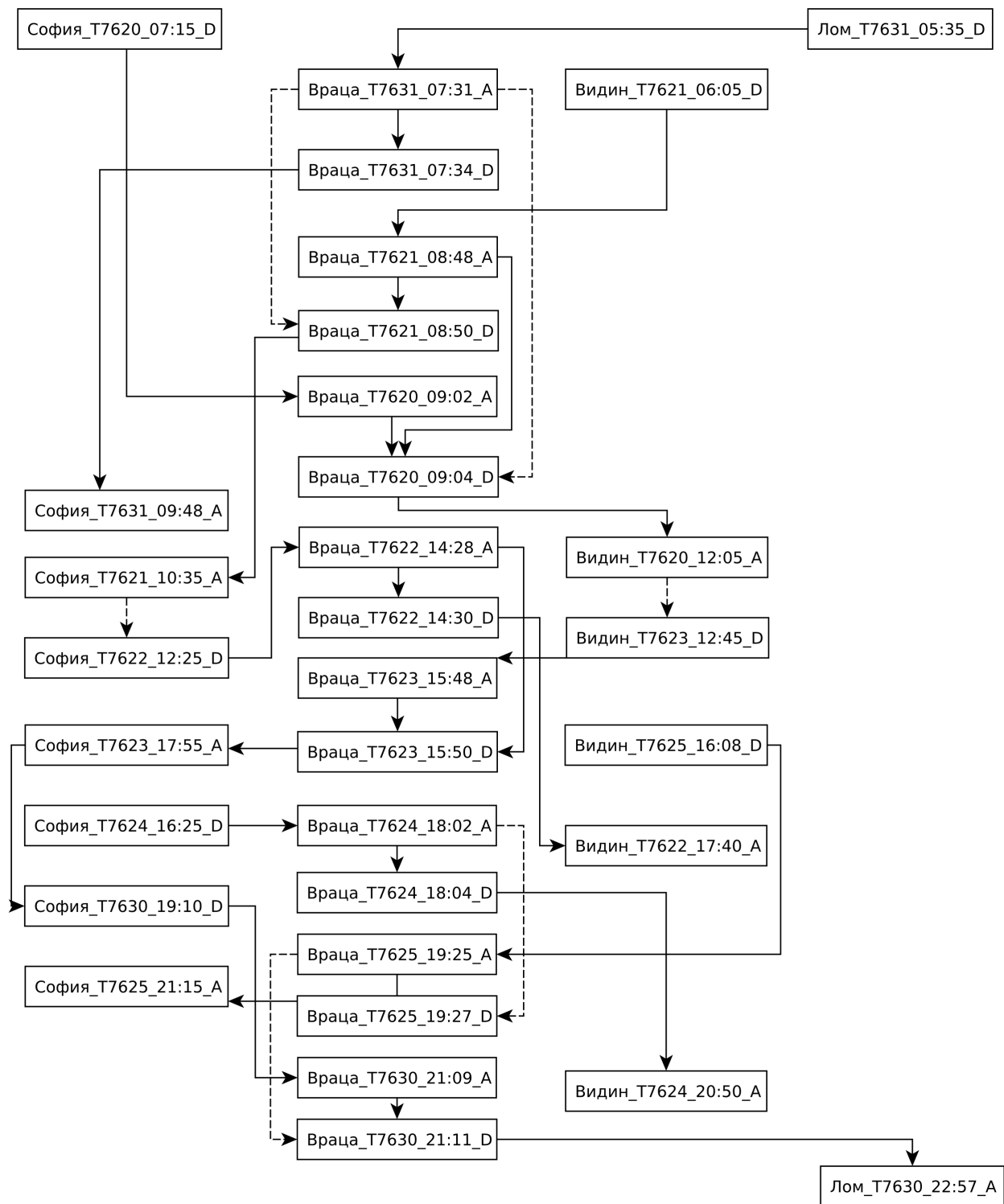
time  int  // време(минути)
ntype bool // тип на върха
arcs  []*DFSArc // списък дъги
}

// Дъга
type DFSArc struct {
    next *DFSNode // връх
    length int // тегло (дължина)
    atype bool // тип на дъгата
}

```

Графът се състои от колони като всяка една колона представлява гара. Във всяка колона са подредени възходящо по време върховете а всеки връх съдържа в себе си списък с връзките към други върхове от графа. Схема на примерните данни обработени във вида на разширен граф е представена на **фигура 7**:





Фигура 7: Разширен граф на движението по осма линия

На фигурата, нормалното движение на влаковете е представено с пълтни стрелки а трансферните дъги са означени с пунктир. Имената на полетата съдържат име на гарата, номер на влака, час и А – за пристигане и D – за заминаване.

Върховете в разширения граф биват два вида:

- Пристигане;

- Потегляне.

Те са представени от една булева променлива: **n<sub>type</sub>** със стойности дефинирани във файла **constants.go**:

```
const (
    // ...
    DEPARTURE = true // заминаване
    ARRIVAL   = false // пристигане
    // ...
)
```

Дъгите също биват два вида:

- Движение на влака
- Трансферни

Те също са представени от една булева променлива: **a<sub>type</sub>** със стойности дефинирани във файла **constants.go**:

```
const (
    // ...
    CONTINUE = true // движение на влака
    TRANSFER = false // трансфер
    // ...
)
```

При създаване на разширеният граф, се създават колони, като всяка от тях съдържа върхове принадлежащи на са една и съща гара:

```
// Създаване на нов разширен граф
func NewDFSGraph(num_stations int) *DFSGraph {
    dfs := make(DFSGraph, num_stations)
    for i := 0; i < num_stations; i++ {
        // Създаване на колона за всяка гара
        dfs[i] = NewNodeColumn()
    }
    return &dfs
}
```

Информацията в разширения граф се попълва паралелно с тази в матрицата на теглата, при обработката на входящите данни:

```
// Обработка на входящите данни
for _, row := range (*raz).templList {
    if row.length == 0 {
        last_distance = 0
    }
    distance = row.length - last_distance

    // ....

    if row.arrival != 0 {
        // Добавяне на нов връх от тип "пристигане"
        node1 = (*raz).dfsGraph.AddTrainStation(row.train, row.station, row.arrival, ARRIVAL)
    } else {
        node1 = nil
    }
    if node1 != nil && node2 != nil {
        // Добавяне на дъга за продължаване на пътуването
    }
}
```

```

        node2.addArc(node1, distance, CONTINUE)
    }
    if row.departure != 0 {
        // Добавяне на нов връх от тип "заминаване"
        node2 = (*raz).dfsGraph.AddTrainStation(row.train, row.station, row.departure,
DEPARTURE)
    } else {
        node2 = nil
    }
    if node1 != nil && node2 != nil {
        // Добавяне дъга за продължаване на пътуването
        node1.addArc(node2, 0, CONTINUE)
    }

    last_station = row.station
    last_train = row.train
}

```

За запълване на графа се използват два метода:

– Добавяне на нов връх:

```

// Създаване на нов връх
func NewNode(station, train, time int, ntype bool) *DFSNode {
    return &DFSNode{station, train, time, ntype, nil}
}

```

```

//Добавяне на връх в графа
func (dfs *DFSGraph) addNode(station int, node *DFSNode) {
    nc := (*dfs)[station]
    nnc := append(*nc, node)
    (*dfs)[station] = &nnc
}

```

```

// Добавяне на връх
func (dfs *DFSGraph) AddTrainStation(train int, station int, time int, ntype
bool) *DFSNode {
    node := NewNode(station, train, time, ntype) //Създаваме нов връх
    dfs.addNode(station, node) // Добавяме върхът в графа
    return node
}

```

– Добавяне на дъга свързваща два върха:

```

// Създаване на нова дъга
func NewArc(next *DFSNode, length int, atype bool) *DFSNode {
    return &DFSNode{next, length, atype}
}

```

```

// Добавяне на дъга между два върха
func (node *DFSNode) addArc(next *DFSNode, length int, atype bool) {
    // Добавя нов дъга към списъка с дъги на текущия връх
    node.arcs = append(node.arcs, NewArc(next, length, atype))
}

```

След запълването на графа с върхове и дъгите за движение на влака, следва изграждане на трансферните дъги. Трансферна дъга е дъга свързваща два върха от една колона в разширения граф за която са изпълнени условията за прехвърляне на пътници между влакове:

```

// Проверка за възможен трансфер между влакове
func (node *DFSNode) canTransferTo(next *DFSNode) bool {
    return node.ntype == ARRIVAL && // Влак 1 пристига
        next.ntype == DEPARTURE && // Влак 2 заминава
        node.train != next.train && // Влак 1 <-> Влак 2
        next.time-node.time >= MIN_TRANSFER_TIME && // Времето между двете събития е по-
        голямо или равно на минималното трансферно време
        next.time-node.time <= MAX_TRANSFER_TIME // Времето между двете събития е по-
        малко или равно на максималното трансферно време
}

// Инициализиране на трансферните дъги
func (dfs *DFSGraph) Init() {
    for _, nc := range *dfs {
        sort.Sort(nc) // Сортиране на върховете във всяка гара по време
        c := nc.Len()
        for i, node := range *nc {
            for j := i; j < c; j++ {
                // Ако е възможен трансферът между два върха
                if node.canTransferTo((*nc)[j]) {
                    // Създаване на трансферна дъга
                    node.addArc((*nc)[j], 0, TRANSFER)
                }
            }
        }
    }
}

```

За сортиране на всяка колона с върхове се използва вграденият алгоритъм на езика Go като предварително е имплементиран интерфейса за сортиране по следния начин:

```

// Брой елементи в колоната
func (nc *NodeColumn) Len() int {
    return len(*nc)
}

// Компаратор
func (nc *NodeColumn) Less(i, j int) bool {
    return (*nc)[i].time < (*nc)[j].time // Сравняваме по време
}

// Разменяне на местата на два върха
func (nc *NodeColumn) Swap(i, j int) {
    (*nc)[i], (*nc)[j] = (*nc)[j], (*nc)[i]
}

```

След сортирането на върховете, за всеки връх от колоната се осъществява проверка дали е възможно прехвърляне от един връх към всеки от следващите го в колоната. Ако има възможност за прехвърляне се създава трансферна дъга от текущия връх до върха за прехвърляна.

На този етап данните за модула са инициализирани и готови за употреба.

Следва на стартиране на WEB сървъра:

```

func StartServer() {
    // Създаване на нова глобална променлива за данните
    razpisanie = NewRazpisanie()
}

```

```

// Зареждане на входящите данни
razpisanie.LoadFromFile("data/paths.txt")
// Обработка на данните
razpisanie.Process()

log.Println("Server operational")
// Дефиниране на услугите на сървъра
http.Handle("/route", http.HandlerFunc(routeHandler))
http.Handle("/distance", http.HandlerFunc(distanceHandler))
// Стартиране на сървъра
log.Fatal(http.ListenAndServe(":8080", nil))
}

```

Сървърът предлага две услуги **route** и **distance** чрез HTTP протокол на порт 8080;

Услугата **distance** връща минималното разстояние между две гари от графа. Тъй като всички разстояния между гари са изчислени, от обработката на матрицата на теглата с алгоритъмът на Флойд-Уоршал, реализацията на услугата е тривиална:

```

// Реализация на услугата distance
func distanceHandler(w http.ResponseWriter, r *http.Request) {
    // Идентификатор на начална гара
    fromid, _ := strconv.Atoi(r.FormValue("from"))
    // Идентификатор на крайна гара
    toid, _ := strconv.Atoi(r.FormValue("to"))
    // Вътрешен идентификатор на начална гара
    from := razpisanie.stations.FindId(fromid)
    // Вътрешен идентификатор на крайна гара
    to := razpisanie.stations.FindId(toid)

    log.Printf("Distance from: %d\tto: %d", from, to)
    // Разстоянието се получава директно от матрицата на теглата
    p := razpisanie.FindDistance(from, to)
    // Връщане на резултата към клиента
    fmt.Fprint(w, p)
}

func (raz *Razpisanie) FindDistance(from, to int) int {
    return (*raz).floydGraph.GetDistance(from, to)
}

```

Услугата **route** предоставя оптималния и близки до оптималния (ако са възможни) маршрути между две гари. Услугата се реализиран от следната функция:

```

// Реализация на услугата route
func routeHandler(w http.ResponseWriter, r *http.Request) {
    // Идентификатор на начална гара
    fromid, _ := strconv.Atoi(r.FormValue("from"))
    // Идентификатор на крайна гара
    toid, _ := strconv.Atoi(r.FormValue("to"))
    // Вътрешен идентификатор на начална гара
    from := razpisanie.stations.FindId(fromid)
    // Вътрешен идентификатор на крайна гара
    to := razpisanie.stations.FindId(toid)

    log.Printf("Search from: %d\tto: %d", from, to)
    // Търсене на маршрути
    p := razpisanie.FindRoutes(from, to)
    // Изграждане на JSON формат на резултата
    fmt.Fprint(w, Encode(p))
}

```

За осъществяване на търсенето, се изисква подаването на два параметъра чрез **POST** заявка: - **from** - идентификатор на начална гара; - **to** - идентификатор на крайна гара;

Двата идентификатора се преобразуват във вътрешни идентификатори и се подават на функцията **FindRoutes**:

```
// Търсене на близки до оптималния маршрути
func (raz *Razpisanie) FindRoutes(from, to int) *Paths {
    // Минимално разстояние между две гари
    min_distance := (*raz).floydGraph.GetDistance(from, to)
    var p *Paths = nil // Нов контейнер за маршрути
    // Започва проверката от маршрути без прехвърляне като се увеличават възможните
    // прехвърляния с единица докато не се получи възможен по смисъла на
    // ограниченията маршрут(и)
    for max_transfers := 0; max_transfers <= MAX_TRANSFERS; max_transfers++ {
        // Търсене в дълбочина на разширен граф
        p = (*raz).dfsGraph.FindRoutes(from, to, int(float64(min_distance)*DISTANCE_COEF),
&(*raz).floydGraph, max_transfers)
        // В случай на резултат търсенето се прекратява
        if len((*p).paths) > 0 {
            // Връщане на откритите резултати
            return p
        }
    }
    return p
}
```

Тъй като в общия случай, маршрутите между два върха на графа са много, като само малка само част от тях са рентабилни, търсенето се лимитира от няколко ограничаващи условия:

- Възможно най-малко прехвърляния между превозни средства. За налагането на това ограничение се изхожда от презюмцията че прехвърлянията са неудобни за пътника и болшинството клиенти при предоставени два маршрута с еднакви други параметри биха предпочели този с по-малко прехвърляне. Максималният брой прехвърляния е константа, дефинирана в **constants.go**:

```
MAX_TRANSFERS = 4 // Максимален брой прехвърляния
```

- Дължина на маршрута по-малка или равна на минималната дължина между двете гари умножена по коефициент за разстояние:

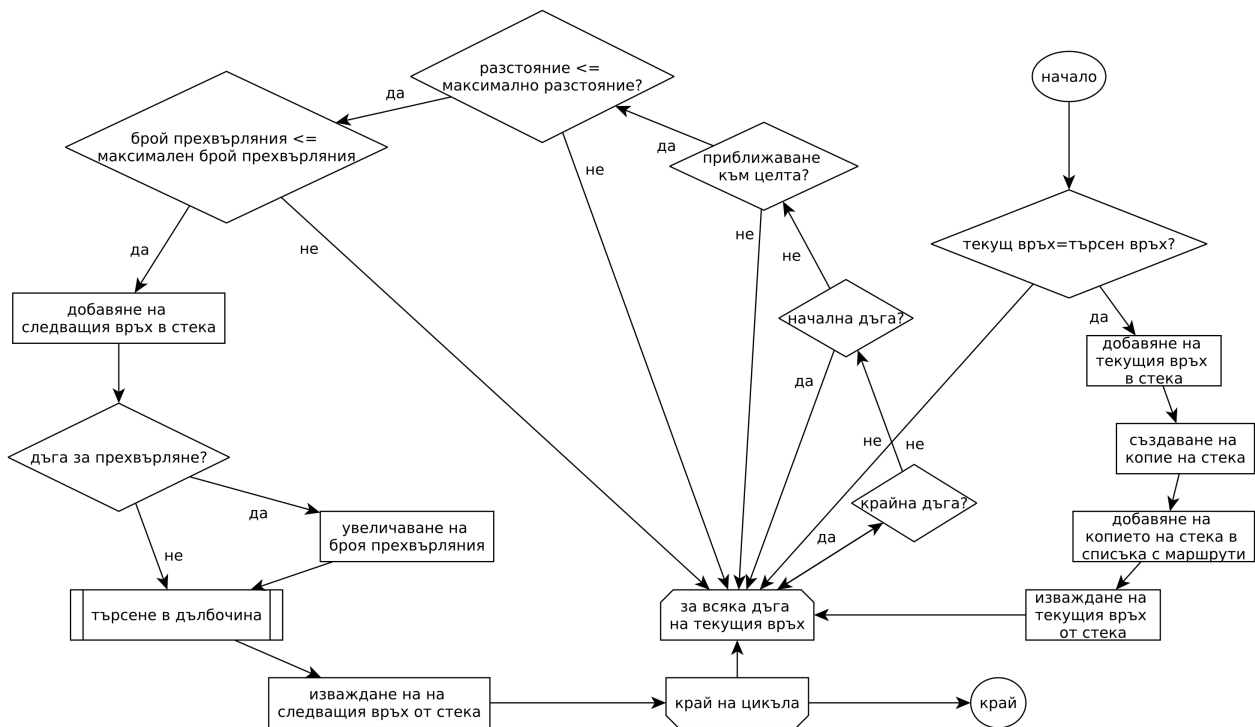
```
DISTANCE_COEF = 1.2 // Коефициент на разстояние
```

В този случай приемаме, че маршрути с дължина 20% над минималната, все още са рентабилни за пътника;

Същинското търсене се осъществява от функцията **FindWay**, която е модифицирано търсене в дълбочина с използване на стек:

```
// Търсене в дълбочина на граф
func (p *Paths) FindWay(node *DFSNode, end_station, length, transfers,
max_transfers int) {
    // При достигане на крайната гара
    if node.station == end_station {
        // Добавяне на крайната гара в стека
    }
}
```





Фигура 8: Търсене в дълбочина

В стека се съхранява комбинация от указатели към връх и дъга до него за не се губи информация за разстоянието между два върха, което се носи от дъгите.

```

type StackNode struct {
    node *DFSNode // Връх
    arc *DFSArc // Дъга
}
// Създаване на нов елемент за стека
func NewStackNode(node *DFSNode, arc *DFSArc) *StackNode {
    return &StackNode{node, arc}
}

```

В списъка с маршрутите се добавя копие на текущия стек:

```

// Добавяне на стек към списъкът с маршрути
func (p *Paths) AddPath() {
    //Създаване на копие на текущия стек
    ns := (*p).stack.Copy()
    p.paths = append(p.paths, ns)
}

```

След приключване на търсенето, структурата **Paths** съдържа списък с откритите маршрути между началната и крайна гара:

```

type Paths struct {
    max_distance int // Максимално разстояние
    floyd *FloydGraph // Матрица на теглата
    stack *Stack // Текущ стек
    paths []*Stack // Списък маршрути
}

```

Така получените данни са в суров вид и съдържат пълните пътища между двете точки. За целите на сайта са нужни само интервалите между значещите гари (начална, крайна и



трансферна). Данните се подготвят за клиента от функцията **Encode** като се ползват експортни структури чрез които резултатът автоматично се конвертира в JSON формат:

```
// Експортна структура
type JSONStack []JSONPath

// Маршрут
type JSONPath struct {
    From_Station_Id int    `json:"from_station_id"`
    To_Station_Id   int    `json:"to_station_id"`
    Distance        int    `json:"distance"`
    Transfers       int    `json:"transfers"`
    Departure_Time  int    `json:"departure_time"`
    Arrival_Time    int    `json:"arrival_time"`
    Details         []JSONDetail `json:"details"`
}

// Сегмент от маршрута
type JSONDetail struct {
    Train_Id        int    `json:"train_id"`
    From_Station_Id int    `json:"from_station_id"`
    To_Station_Id   int    `json:"to_station_id"`
    Distance        int    `json:"distance"`
    Departure_Time  int    `json:"departure_time"`
    Arrival_Time    int    `json:"arrival_time"`
}

// Подготвяне на маршрутите за връщане към клиента
func Encode(p *Paths) string {
    // Създаване и запълване нова експортна структура
    js := p.CreateJSONStack(razpisanie.stations, razpisanie.trains)
    // Преобразуване на структурата в JSON запис
    b, err := json.Marshal(js)
    if err != nil {
        log.Println(err)
        b = []byte("{\"err': 'Marshal Failure'}")
    }
    // Връщане на JSON текста
    return string(b)
}

// Създаване и запълване нова експортна структура
func (p *Paths) CreateJSONStack(stations, trains *Dictionary) JSONStack {
    var js JSONStack
    for _, s := range p.paths {
        // Добавяне към експортната структура на всеки маршрут
        js = append(js, s.DumpJSON(stations, trains))
    }
    return js
}

// Експорт на заглавна част на маршрута
func (s *Stack) DumpJSON(stations, trains *Dictionary) JSONPath {
    var jp JSONPath
    // Начална гара
    jp.From_Station_Id = (*stations)[(*s)[0].node.station].id
    // Крайна гара
    jp.To_Station_Id = (*stations)[(*s)[len(*s)-1].node.station].id
    // Разстояние и брой прехвърляния
    jp.Distance, jp.Transfers = s.GetDistanceAndTransfers()
    // Час на потегляне за маршрута
    jp.Departure_Time = (*s)[0].node.time
}
```

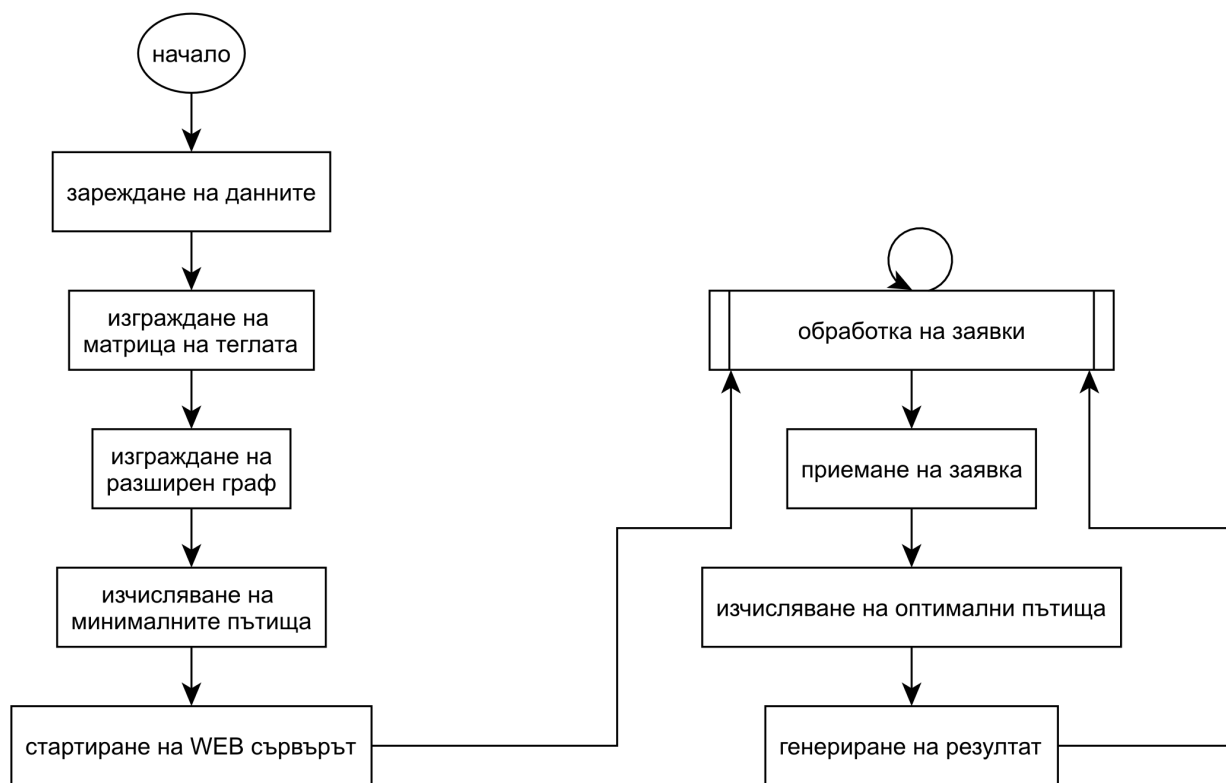
```

// Час на пристигане за маршрута
jp.Arrival_Time = (*s)[len(*s)-1].node.time
// Експорт на сегментите на маршрута
jp.Details = (*s).DumpRouteDeatilsJSON(stations, trains)
return jp
}

// Експорт на сегментите на маршрута
func (s *Stack) DumpRouteDeatilsJSON(stations, trains *Dictionary) []JSONDetail
{
var jds []JSONDetail
var jd JSONDetail
jd.Distance = 0 // Разстояние за сегмента
jd.To_Station_Id = 0 // Крайна гара за сегмента
jd.Arrival_Time = 0 // Час на пристигане за сегмента
// Идентификатор на влак за сегмента
jd.Train_Id = (*trains)[(*s)[0].node.train].id
// Начална гара за сегмента
jd.From_Station_Id = (*stations)[(*s)[0].node.station].id
// Час на потегляне за сегмента
jd.Departure_Time = (*s)[0].node.time
// Обхождане на всички елементи на маршрута
for _, path := range *s {
// Промяна на крайната гара
jd.To_Station_Id = (*stations)[path.node.station].id
// Промяна на часа на пристигане
jd.Arrival_Time = path.node.time
// Ако има следващ връх
if path.arc != nil {
// Добавяне на разстоянието на дъгата към сегмента
jd.Distance += path.arc.length
// Ако дъгата е за прехвърляне
if path.arc.atype == TRANSFER {
// Добавяне на сегмента към резултата
jds = append(jds, jd)
// Инициализиране на променливите за новия сегмент
jd.Distance = 0
jd.Train_Id = (*trains)[path.node.train].id
jd.From_Station_Id = (*stations)[path.node.station].id
jd.Departure_Time = path.node.time
}
}
}
// Добавяне на последния сегмент
jds = append(jds, jd)
return jds
}

```

Модулът работи като API сървър, отворен за заявки от потребителската част на сайта. Цялостния модел на работа на модула е представен на **фигура 7**:



Фигура 9: Обща схема на работа на модул *razpisanie*

### 3.3. Потребителски модул

Потребителският модул на системата е предназначен за крайните клиенти на превозната услуга. За създаването му са използвани следните технологии:

- PHP (<http://php.net/>)- език за програмиране;
- CodeIgniter (<http://codeigniter.com/>)- framework;
- Apache2 (<http://httpd.apache.org/>) - уеб сървър;
- Twitter Bootstrap (<http://twitter.github.com/bootstrap/>) - CSS/JavaScript библиотека поддържаща адаптивна структура;

Модулът позволява на потребителя следните възможности:

- Търсене на маршрути между всяка двойка гари/спирки от разписанието за определена дата:

Билети Вход Регистрация 0 / 0.00 лв.


**Търсене**

Начална гара

Крайна гара

Дата на пътуване

[Търсене](#)



2012

Фигура 10: Начална страница – полета за търсене на маршрут

- Избор на подходящ маршрут от списък:

Билети Вход Регистрация 0 / 0.00 лв.

**София - Велико Търново : 25.06.2012**

№	Заминава	Пристига	Категория	Прекачвания	Времетраене
1	10:05	14:51	БВ, ПВ	1	04:46
<b>Гара/Спирка</b>		<b>Влак</b>	<b>Категория</b>	<b>Заминава</b>	<b>Пристига</b>
София » Горна Оряховица		2613	БВ	10:05	14:12
Горна Оряховица » Велико Търново		40103	ПВ	14:12	14:51
<a href="#">Билети</a>					
2	13:20	18:05	БВ, МБВ	1	04:45
3	15:40	20:20	БВ, КПВ	1	04:40
<b>Гара/Спирка</b>		<b>Влак</b>	<b>Категория</b>	<b>Заминава</b>	<b>Пристига</b>
София » Горна Оряховица		4612	БВ	15:40	19:42
Горна Оряховица » Велико Търново		40217	КПВ	19:42	20:20
<a href="#">Билети</a>					

2012

Фигура 11: Избор на вариант за пътуване

- Преглед на гарите/спирките, през които преминава превозното средство:

Билети
Вход
Регистрация
🛒 0 / 0.00 лв.

## Влак КПВ40217

---

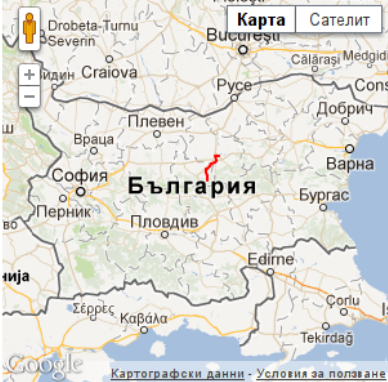
### Крайградски пътнически влак

**Подвижен състав**

- Електрически локомотив серия 43 [лок43]
- Вагон 2 клас [В]
- Вагон 2 клас [В]
- Вагон 2 клас [В]

### Маршрут

Гара/Спирка	Пристигане	Заминаване
Горна Оряховица	-	20:00
Самоводене	20:08	20:09
Трапезица	20:16	20:17
Велико Търново	20:20	20:21
Дълга Лъка	20:24	20:25
Дебелец	20:30	20:31
Рп Соколово	20:39	20:40
Ганчовец	20:46	20:46
Дряново Сп.	20:50	20:51
Дряново	20:54	20:55
Бачо Киро	21:00	21:01
Царева ливада	21:06	21:07
Стайновци	21:14	21:15
Трявна	21:20	21:26
Божковци	21:31	21:31
Плачковци	21:36	-



Карта Сателит  
Картата показва маршрута на влака през България, с ключови спирки като Горна Оряховица, Велико Търново, Дряново, Бачо Киро, Царева ливада, Стайновци, Трявна, Божковци и Плачковци.

2012

Фигура 12: Информационна страница за влак



- Резервация на различни видове билети и избор на запазено място в превозното средство, където това е възможно;

Билети
Вход
Регистрация
🛒 0 / 0.00 лв.

### Данни за пътуването

От гара: София  
До гара: Велико Търново  
За дата: 06.07.2012  
Разстояние: 300.9 км

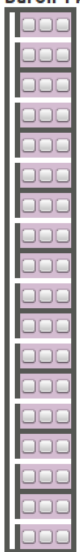
## Влак: БВ2613

### Билети без запазени места


Тарифа	Клас	Цена (лв)	Брой
Стандартен билет	2	14.60	<input type="text" value="0"/>
Стандартен билет	1	18.30	<input type="text" value="0"/>
Отиване и връщане	2	26.30	<input type="text" value="0"/>
Отиване и връщане	1	32.90	<input type="text" value="0"/>
Малки групи	2	12.40	<input type="text" value="0"/>
Малки групи	1	15.60	<input type="text" value="0"/>

### Билети със запазени места


Вагон 1 клас




Вагон 2 клас



Вагон 2 клас



Вагон 2 клас



Резервация

Фигура 14: Избор на билет/място

- Възможност за заплащане на резервираните билети онлайн.

Билети
Вход
Регистрация
🛒 2 / 29.20 лв.

Билет	Брой	Единична цена (лв)	Общо (лв)
<b>Билет от София до Горна Оряховица за 25.06.2012</b> Влак: <b>БВ2613</b> ; Заминава: <b>10:05</b> ; Пристига: <b>14:12</b> ; Клас: <b>2</b> ; Вагон: <b>3</b> ; Място: <b>1</b> ;	1	14.60	14.60
<b>Билет от София до Горна Оряховица за 25.06.2012</b> Влак: <b>БВ2613</b> ; Заминава: <b>10:05</b> ; Пристига: <b>14:12</b> ; Клас: <b>2</b> ; Вагон: <b>3</b> ; Място: <b>2</b> ;	1	14.60	14.60
		<b>Сума</b>	<b>29.20</b>

Обнови поръчката
Поръчка

2012

Фигура 15: Потвърждаване на поръчката

Билети
Моят профил
Изход
🛒 0 / 0.00 лв.

Начало / 
 Моят профил / 
 Очакващи плащане

### Очакващи плащане

	От гара	До гара	Влак	Клас	Вагон	Място	Статус	Брой	Цена	Сума
<input checked="" type="checkbox"/>	София	Горна Оряховица	БВ2613	2	3	1	Очаква плащане	1.00	14.60	<b>14.60</b>
<input checked="" type="checkbox"/>	София	Горна Оряховица	БВ2613	2	3	2	Очаква плащане	1.00	14.60	<b>14.60</b>

Плати избраните

2012

Фигура 16: Плащане на избрани билети



- Възможност за регистрация в портала от където да се следят всички закупени от клиента билети;

Билети Вход Регистрация  0 / 0.00 лв.

### Регистрация

E-mail

Име

Телефон


Парола

Парола отново

Съгласен съм с условията за ползване на сайта

2012

*Фигура 17: Регистрация на потребител*

Билети Вход Регистрация  2 / 29.20 лв.

Трябва да влезете в профила си за да продължите

### Вход

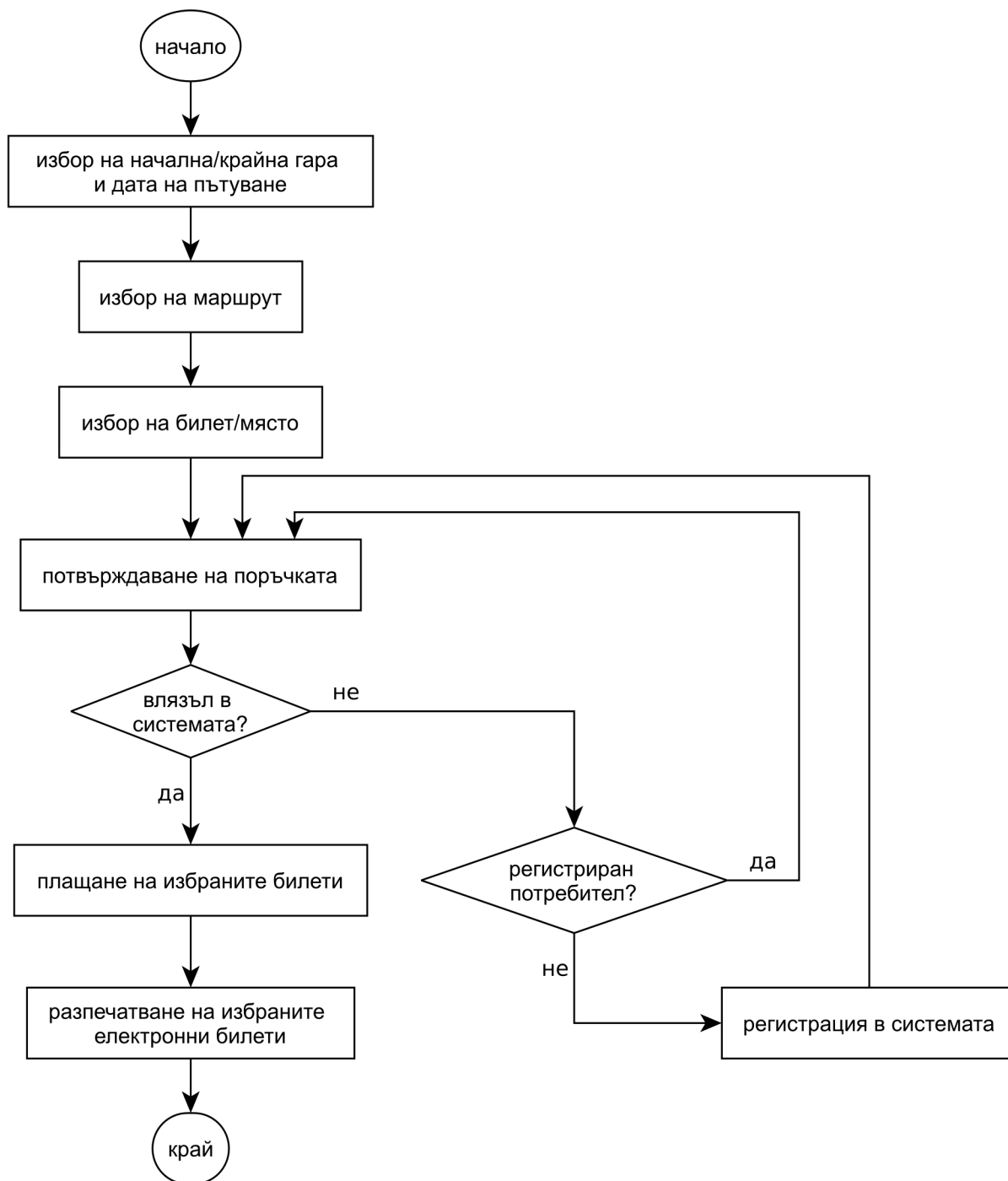
E-mail

Парола

2012

*Фигура 18: Вход в системата*

Основният процес на монетизация на сайта е закупуването на превозни документи. Това се осъществява чрез следния процес:



Фигура 19: Процес на избор и закупуване на електронен билет за пътуване

## 3.4. Административен модул

Административният модул дава възможност за манипулация на следните елементи:

### 3.4.1. Гари/спирки

Всяка гара или спирка в системата се представя от име и географски координати . Координатите се използват при показване на информация за гара както и за маршрутите на влаковете.

Билети					Моят профил	Изход	0 / 0.00 лв.
Начало / Администрация / Гари					Нова гара		
Гари							
ID	Гара	Ширина	Дължина	Активна			
511	Абланица	24.7129	43.0305	1			
415	Аврамово	23.8503	42.0447	1			
212	Айтос	27.2419	42.7031	1			
7	Алдомировци	22.979	42.8851	1			
500	Александрово	24.9384	43.2658	1			
537	Анево	24.7222	42.6514	1			
271	Антон	24.2675	42.7316	1			
468	Арковна	27.1812	43.0401	1			
660	Асеновград	24.87	42.0136	1			
224	Асеново	26.0217	43.2619	1			
497	Асеновци	25.0849	43.3266	1			
177	Аспарухово	27.3161	42.9822	1			
638	Атолово	26.7554	42.595	1			
358	Байково	26.8578	43.456	1			
433	Балкан	23.3668	43.0559	1			
673	Банковица	23.0018	42.3632	1			
650	Банкя	23.1513	42.7058	1			
425	Банско	23.4845	41.8461	1			
714	Баня	24.8352	42.5496	1			
32	Батановци	22.9514	42.5952	1			
1 2 3 > Last >							

2012

Фигура 20: Списък налични гари

### 3.4.2. Влакове

Администрацията позволява въвеждането на нови влакове, както и редакцията на вече съществуващи. Всеки влак се характеризира с име и тип.

Билети Моят профил Изход 🛒 0 / 0.00 лв.

---

[Начало](#) / [Администрация](#) / [Гари](#) / Редакция на гара Айтос

#### Редакция на гара Айтос

Име	<input type="text" value="Айтос"/>
Широчина	<input type="text" value="27.2419"/>
Дължина	<input type="text" value="42.7031"/>
Активна	<input type="text" value="да"/>

2012

*Фигура 21: Редакция на гара*

Начало / Администрация / Влакове

Нов влак

## Влакове

ID	Влак	Пристигане	Заминаване	От гара	До гара	Активен	Подв. състав	Маршрут	Участ. скорост	Приходи
1	МБВ292	22:20	20:40	София	Димитровград Жс	1	Подв. състав	Маршрут	Участ. скорост	Приходи
2	МБВ293	07:15	05:52	Димитровград Жс	София	1	Подв. състав	Маршрут	Участ. скорост	Приходи
3	МБВ360	12:40	09:15	Кулата	София	1	Подв. състав	Маршрут	Участ. скорост	Приходи
4	МБВ361	10:20	07:00	София	Кулата	1	Подв. състав	Маршрут	Участ. скорост	Приходи
5	МБВ362	23:20	20:02	Кулата	София	1	Подв. състав	Маршрут	Участ. скорост	Приходи
6	МБВ363	20:25	17:05	София	Кулата	1	Подв. състав	Маршрут	Участ. скорост	Приходи
7	МБВ382	03:40	19:40	София	Гюргево Север	1	Подв. състав	Маршрут	Участ. скорост	Приходи
8	МБВ383	05:55	22:05	Гюргево Север	София	1	Подв. състав	Маршрут	Участ. скорост	Приходи
9	МБВ462	16:25	03:50	Кулата	Гюргево Север	1	Подв. състав	Маршрут	Участ. скорост	Приходи
10	МБВ462/2	16:25	03:50	Кулата	Гюргево Север	0	Подв. състав	Маршрут	Участ. скорост	Приходи
11	МБВ463	02:25	14:25	Гюргево Север	Кулата	1	Подв. състав	Маршрут	Участ. скорост	Приходи
12	МБВ464	11:40	06:50	Димитровград	Горна Оряховица	1	Подв. състав	Маршрут	Участ. скорост	Приходи
13	МБВ465	22:02	17:48	Горна Оряховица	Димитровград	1	Подв. състав	Маршрут	Участ. скорост	Приходи
14	МБВ490	13:10	11:40	София	Димитровград Жс	1	Подв. състав	Маршрут	Участ. скорост	Приходи
15	МБВ491	01:10	16:17	Димитровград Жс	Капъ Куле	1	Подв. състав	Маршрут	Участ. скорост	Приходи
16	МБВ491/1	01:59	16:17	Димитровград Жс	Капъ Куле	0	Подв. състав	Маршрут	Участ. скорост	Приходи
17	МБВ492	10:35	04:05	Капъ Куле	София	1	Подв. състав	Маршрут	Участ. скорост	Приходи
18	МБВ964	15:20	06:50	Димитровград	Русе	1	Подв. състав	Маршрут	Участ. скорост	Приходи
19	МБВ965	23:38	15:25	Русе	Димитровград	1	Подв. състав	Маршрут	Участ. скорост	Приходи
20	МБВ1003	08:55	05:18	Русе	Варна	1	Подв. състав	Маршрут	Участ. скорост	Приходи

1 2 3 > Last >

Фигура 22: Списък влакове

### 3.4.3. Маршрути на влакове

Маршрутът на всеки влак се състои от поредица от име на гара, час на пристигане, час на заминаване и разстояние от началото на маршрута. Маршрутите могат да бъдат допълвани и променяни.

Билети Моят профил Изход 🛒 0 / 0.00 лв.

Начало / Администрация / Влакове / Маршрути

#### Маршрут на влак МБВ292

Гара	Пристигане (час:мин)	Заминаване (час:мин)	Разстояние (км)
София	-	20:40	0
Волуяк	20:50	20:50	7.8
Костинброд	20:56	20:57	14.7
Петърч	21:03	21:03	21.6
Сливница	21:08	21:08	27.6
Сливница Сп.	21:10	21:11	29.1
Алдомировци	21:18	21:18	35.2
Драгоман	21:25	21:30	42.3
Драгоил	21:38	21:38	49.7
Калотина	21:43	21:43	54.1
Калотина Запад	21:46	22:11	55.9
Граница Кан	22:13	22:13	56.7
Димитровград Жс	22:20	-	63.2
	-	-	0

Фигура 23: Маршрут на влака

### 3.4.4. Подвижен състав на влак

Позволява указване на състава на влака. Тази информация се използва при продажбата на билети (Приложение: фиг. 14).

Билети Моят профил Изход 0 / 0.00 лв.

[Начало](#) / [Администрация](#) / [Влакове](#) / [Вагони](#)

#### Вагони за влак BV1621

Вагон 1	лок43 ▼
Вагон 2	В ▼
Вагон 3	В ▼
Вагон 4	АВ ▼
Вагон 5	АВ ▼
Вагон 6	--- ▼

Фигура 24: Избор на подвижен състав

### 3.4.5. Превозни средства

Тази администрация позволява добавяне и редактиране на елементите на подвижния състав. Всяко превозно средство се описва с код, име, брой места за пътници и схема, която се показва при избора на запазени места (Приложение: фиг. 18).

Синтаксиса за дефиниране на схемата на превозното средство е:

- " " - коридор;
- "-" или "|" - стена;
- "1" или "2" - съответен клас място;





### 3.4.6. Тарифи

Позволява редакция на тарифните коефициенти за превоз. (Приложение: фиг. 17)

Билети				Моят профил	Изход	🛒 0 / 0.00 лв.
Начало / Администрация / Цени						
RБВ ПВ БВ МБВ						
<b>Цени</b>						
От	До	Клас 2	Клас 1			
1	10	1.80	2.30			
11	20	2.40	3.00			
21	30	3.10	3.90			
31	40	3.60	4.50			
41	50	4.10	5.10			
51	60	4.50	5.60			
61	70	4.90	6.10			

Фигура 26: Редакция на тарифа

Освен промяната на данни, администрацията дава достъп до няколко основни статистически справки:

- Минимално разстояние между две гари (получава се от матрицата на теглата на модул **razpisanie**);

Билети				Моят профил	Изход	🛒 0 / 0.00 лв.
Начало / Администрация / Минимално разстояние между две гари						
Минималното разстояние от <b>София</b> до <b>Пловдив</b> е <b>155.8</b> километра.						
<b>Търсене</b>						
Начална гара						
<input type="text" value="София"/>						
Крайна гара						
<input type="text" value="Пловдив"/>						
<input type="button" value="Търсене"/>						

2012

Фигура 27: Минимално разстояние между две гари

- Натоварване на гари;

Начало / Администрация / Натоварване на гари

### Натоварване на гари

ID	Влак	Брой влакове за денонощие
1	София	357
141	Пловдив	259
93	Русе Разпредел.	177
2	Волюяк	146
142	Пор Изток	145
125	Искър	143
207	Владимир Павлов	134
41	Илиянци	130
168	Карнобат	127
102	Димитровград	123
126	Казичене	122
727	Подуене Разпред	122
109	Стара Загора	119

Фигура 28: Натовареност на гари

- Участъкова скорост по отсечки за влак;

Начало / Администрация / Влакове / Участъкови скорости

### Участъкови скорости

Влак	От гара	До гара	Разстояние (km)	Времетраене (min)	Участъкова скорост (km/h)
МБВ292	София	Волюяк	7.8	10	46.80
МБВ292	Волюяк	Костинброд	6.9	6	69.00
МБВ292	Костинброд	Петърч	6.9	6	69.00
МБВ292	Петърч	Сливница	6.0	5	72.00
МБВ292	Сливница	Сливница Сп.	1.5	2	45.00
МБВ292	Сливница Сп.	Алдомировци	6.1	7	52.29
МБВ292	Алдомировци	Драгоман	7.1	7	60.86
МБВ292	Драгоман	Драгоил	7.4	8	55.50
МБВ292	Драгоил	Калотина	4.4	5	52.80
МБВ292	Калотина	Калотина Запад	1.8	3	36.00
МБВ292	Калотина Запад	Граница Кан	0.8	2	24.00
МБВ292	Граница Кан	Димитровград Жс	6.5	7	55.71

2012

Фигура 29: Участъкова скорост на влак

- Месечни приходи за влак;

Възможно е добавянето на допълнителни справки при необходимост.

## 4. Ефективност на системата

Така изготвената примерна система позволява бързо и лесно закупуване на желаните билети от потребителя. За проверка на ефективността на търсене на маршрути от модула **razpisanie** се използва следния скрипт:

### benchmark.php

```
<?php

function test($from, $to) {
    $url = 'http://localhost:8080/route';

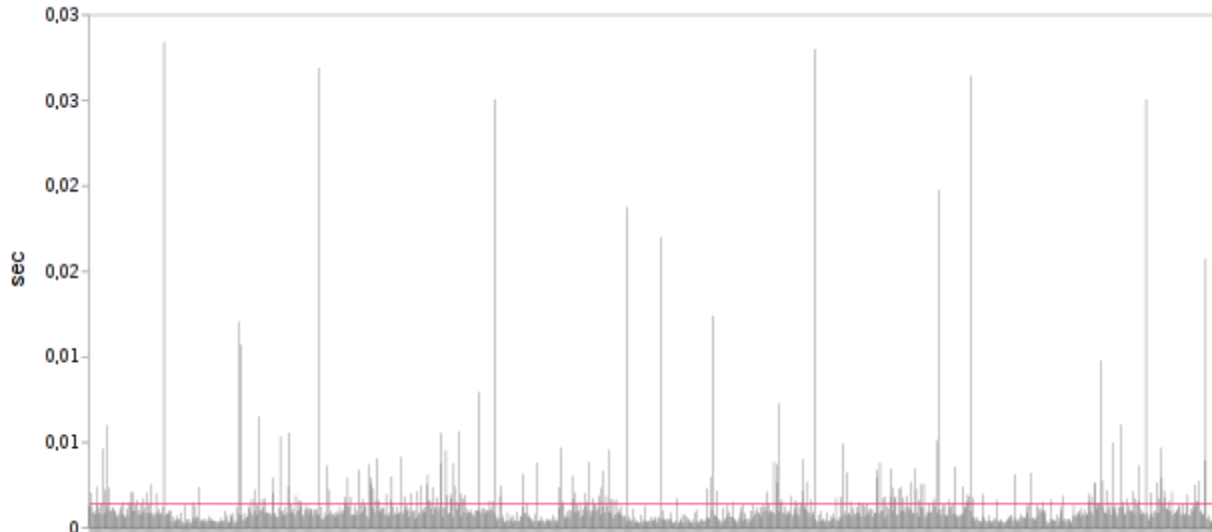
    $post = sprintf('from=%d&to=%d', $from, $to);

    $ch = curl_init();

    curl_setopt($ch,CURLOPT_URL,$url);
    curl_setopt($ch,CURLOPT_POST, 2);
    curl_setopt($ch,CURLOPT_POSTFIELDS,$post);
    curl_setopt($ch,CURLOPT_RETURNTRANSFER, true);
    $result = curl_exec($ch);
    curl_close($ch);
    return $result;
}

for ($i=0; $i< 1000; $i++) {
    $from = rand(1,700);
    $to = rand(1,700);
    $time_start = microtime(true);
    test($from,$to);
    echo microtime(true) - $time_start.PHP_EOL;
}
?>
```

Скриптът извършва 1000 търсения на маршрути между две произволни гари и измерва времето за всяко търсене. Резултатите от теста са показани на **фигура 30**:



Фигура 30: Време за търсене на маршрути между произволни гари

Средното време за търсене е 1.4 милисекунди като в това време се включва и времето за HTTP комуникация.

По отношение на първоначалното стартиране на сървъра, времето от стартиране до пълна оперативна готовност е 4 секунди:

```
2012/07/03 14:41:28 Loaded 12552 lines from "data/paths.txt"
2012/07/03 14:41:32 Server operational
```

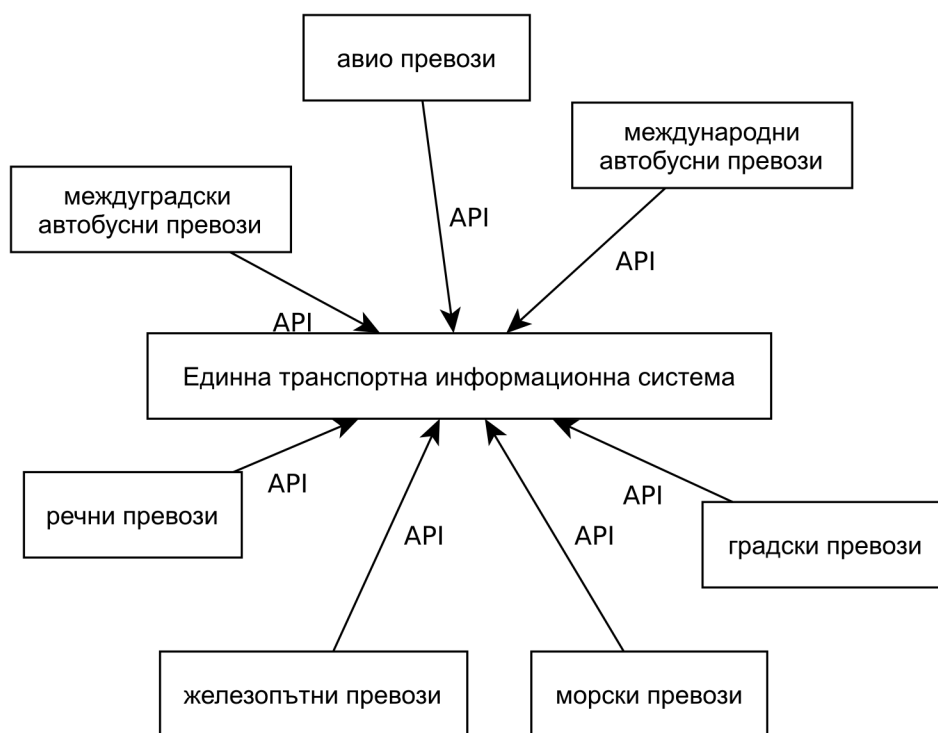
Тестовете са проведени на конфигурация:

Процесор	Intel® Core™ i3 CPU 3.20GHz
Памет	8 GB
Операционна система	Debian Wheezy
Ядро	linux-image-3.2.0-2-686-pae #1 SMP Mon Jun 11 18:27:04 UTC 2012 i686 GNU/Linux
Компилатор	go1.0.1

## 5. Изводи и препоръки

Така описаната система подлежи на постоянно развитие и модификация. Едно допълнително приложение на системата би могло да бъде интегрирането и в т.нар. киоск системи. Примерна конфигурация на такава система би могла да бъде окомплектована с АТМ функционалност както и с фискален принтер за директен печат на билети.

Друго направление за употреба на системата би могло да бъде интегрирането и със сродни системи за други видове транспорт. Това ще добави още по голяма стойност към услугата. Възвращаемостта от подобна система може да надхвърли в пъти инвестицията направена за изработка или закупуване на подобна система. Оперативните разходи по системата са минимални и голяма част от тях се правят и при сега наличия портал.



Фигура 31: Принципна схема на интегрирана система за предоставяне на транспортни услуги

Една такава децентрализирана система би комуникирала с отделните системи на доставчиците на транспортни услуги чрез унифициран програмен интерфейс (API), който минимално би трябвало да включва команди за:

- Маршрути между две точки за интервал от време;
- Свободни позиции за маршрут;
- Възможност за резервация на позиция;

Ползите от подобни системи са:

- Удобство и предсказуемост за клиентите;

- Унификация на критериите;
- По-добри условия за конкуренция;
- По-лесно "продаване" на транспортната услуга;
- Възможности за оптимизация и планиране;

Информационни системи се използват в транспорта и днес. Това което се забравя обикновено е че една информационна система никога не е завършена, защото тя трябва да отразява реалния свят, който винаги се променя.

### **5.1. Изводи:**

1. Разработването на онлайн система за резервация на билети е необходимо за подобряване на конкурентоспособността на БДЖ АД;
2. Изработката на подобна базова система е възможно да се постигне в кратки срокове (около половин година) на постижима за компанията цена;
3. Пазарът на он-лайн услуги в България е в растеж и в светлината на предстоящото електронно правителство, все по голяма част от операциите, традиционно извършвани на място, ще имат своят електронен аналог;
4. Обвързката на железопътната система на България с европейската железопътна система, освен на физическо, трябва да стане и на информационно равнище;

### **5.1. Препоръки:**

1. Планирането и изграждането на он-лайн система за резервация на билети трябва да започне в най-кратки срокове, защото страната ни вече е изостанала от общоевропейските тенденции.
2. Стратегията за предлагане на електронни услуги трябва да е дългосрочна и съпроводена с периодично преразглеждане на приоритетите, за да отговаря адекватно на съвременните изисквания.

## 6. Използвана литература

Списък с използваната литература

- [1] Cozac Ion, *DRUMURI OPTIME ÎN GRAFURI ORAR*,  
[http://www.upm.ro/facultati\\_departamente/stiinte\\_litere/conferinte/situl\\_integrare\\_european\\_a/Lucrari2/Ioan%20Cozac.pdf](http://www.upm.ro/facultati_departamente/stiinte_litere/conferinte/situl_integrare_european_a/Lucrari2/Ioan%20Cozac.pdf)
- [3] БДЖ ЕАД, Тарифни известия, брой 1, 11.01.2012
- [2] Наков, Преслав, *Основи на компютърните алгоритми*, Top Team Co., 1999