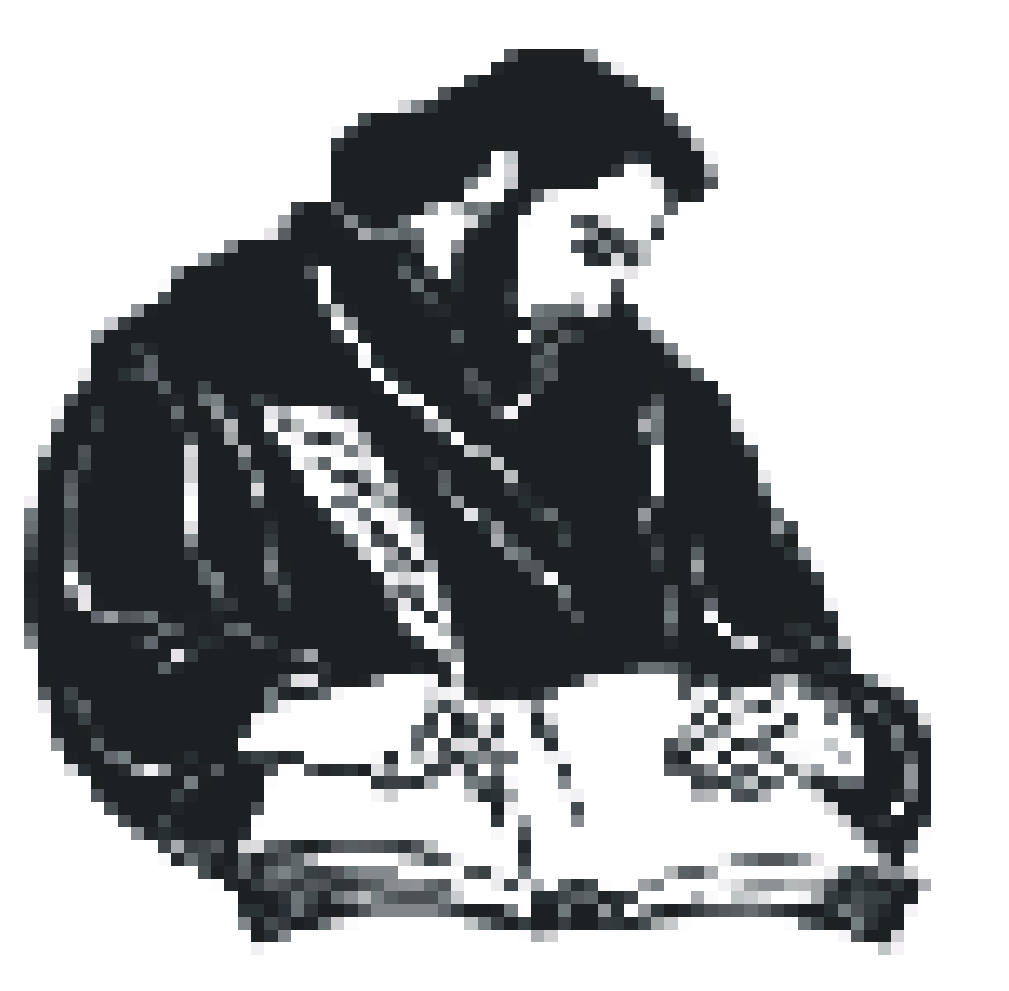
** Пловдивски университет „Паисий Хилендарски”**



Факултет по Математика и Информатика

Катедра “Компютърна Информатика”

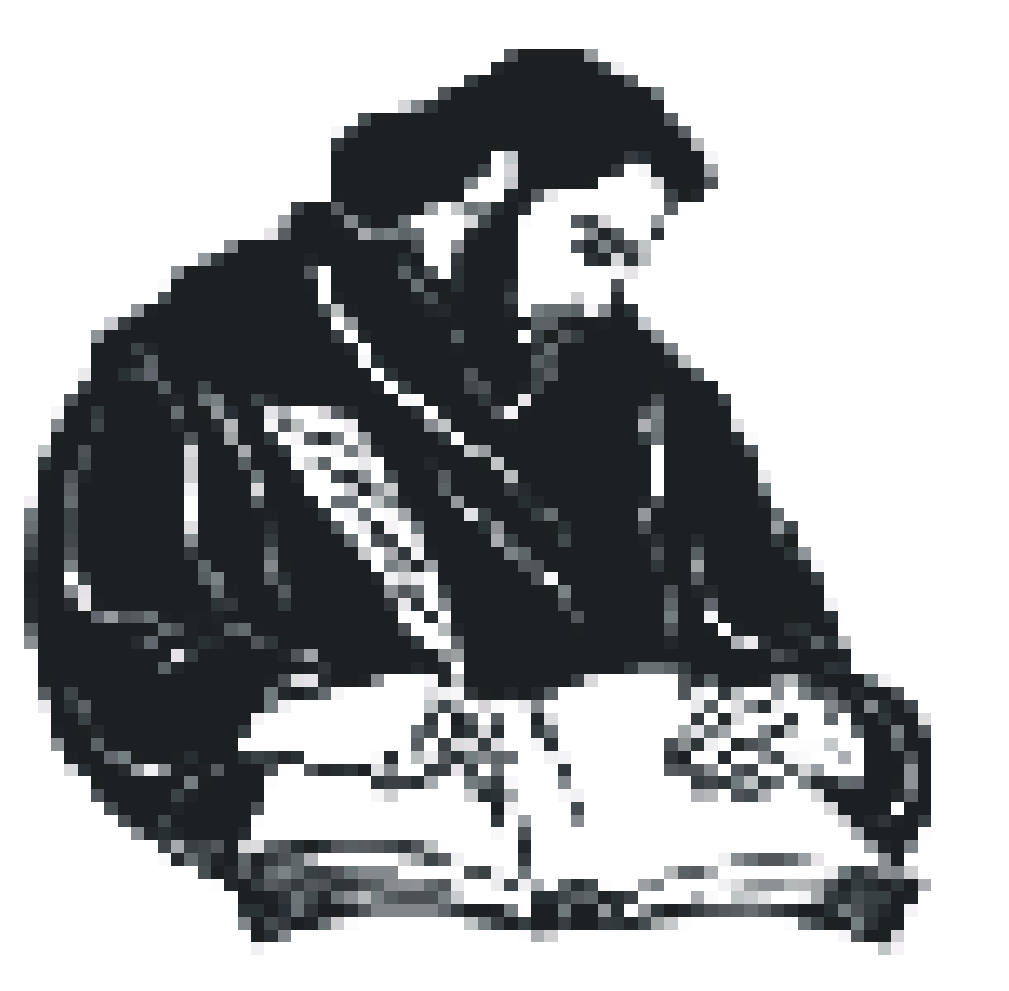
**ДИПЛОМНА РАБОТА**

Информационен портал за новини от България и света

**Дипломант: Кристиан Александров Витозев  
Факултетен номер: 1201262010**

**Научен ръководител: доц. д-р Светослав Енков**

**Пловдив  
2018 г.**

** Plovdiv University “Paisii Hilendarski”**



Facultiy of mathematics and informatics

**GRADUATION THESIS**

Web portal for world news

**Student: Kristian Aleksandrov Vitozev**

**Student number: 1201262010**

**Science supervisor: Assoc. Prof. Ph.D Svetoslav Enkov**

**Plovdiv  
2018**

## 1. Увод

Средствата за масова комуникация са се превърнали в съществена част от ежедневието на всеки един човек в днешно време. В най-чистата си форма, медиите ни дават информация за неща, които са се случили или предстои да се случат. Ключовата характеристика на една новина е нейната обективност, за съжаление историята помни много случаи на т.нар. “медийно затъмнение”, което цели да скрие пълната или частична истина за дадено събитие от потенциалната аудитория.

За много хора интернет е неизчерпаем източник на информация, забавление, както и невероятна възможност за правене на бизнес. Потребителите прекарват голяма част от ежедневието си в интернет и сайтовете за новини постепенно изместват вестниците като основен източник от който хората получават информация за случващото се около тях. В интернет пространството съществуват множество новинарски сайтове, което поставя пред обществото нов проблем, а именно т.нар. “фалшиви новини”. Фалшивите новини представляват невярна информация, подвеждаща аудиторията, чиято единствена цел е придобиване на голяма популярност и генериране печалби на авторите.

Проектът разработен в тази дипломна работа е независим сайт за споделяне на новини, който се стреми проактивно (чрез действието на всеки един потребител) да се бори с проблема, който пораждат фалшивите новини. В сайта потребителите могат да намерят категоризирана информация, публикувана от нашия професионален авторски екип. Читателите могат да се регистрират и да споделят мнение под формата на коментари за дадена новина, както и да дадат своята лична оценка дали новината, която щреглеждат е достоверна или не, като тази информация последствие ще бъде видима за останалите потребители на сайта. По този начин целим създаването на сигурна среда, която хората ще могат да използват за научаване на последните новини, без да има нужда да проверяват редица източници за достоверността информационния поток.

## 1. Resume

The news have become an essential part of people's everyday life. The media gives us information about things that have happened or are about to happen. The key feature of a news story is its objectivity, but unfortunately the story remembers many cases of the so-called " Media eclipse", which aims to hide the full, or partial truth of an event from the potential audience.

Many people find the Internet as endless source of information, entertainment and perfect environment for running a business. People spend the most of their time on the Internet and that’s why, the news websites are gradually shifting newspapers as main source of information. In nowadays, there are many news websites on the web, and they expose to the public the problem of "fake" news. Fake news is a type of yellow journalism or propaganda that consist of deliberate misinformation or propaganda.

The project itself is an independent website where people can read news and also to use proactive approach for "fighting" the problem of fake news. Users can find categorized information published by our professional team of authors. They can register and share their opinion in form of comments. Registered users can log into their accounts and vote for different comments as well.  
In this way, we aim to create a secure environment where people can find the latest news, without having to check a large amount of data sources before that.

## 

[**1. Увод**](#_j4kbeeesfu6z) **2**

[**1. Resume**](#_edde22ve2dvz) **3**

[**2. Цели и задачи**](#_uju8qqp0m79l) **5**

[2.1 Структура на дипломната работа](#_eact5rjvrkuj) 7

[**3. Проектиране и програмна реализация**](#_cqnroqht9579) **8**

[3.1 Обзор на информационните технологии и средства](#_5ze71cpymizn) 8

[3.2 Структура на приложението.](#_93o6mg8f76aq) 16

[3.2.1 Обща архитектура](#_mtmwpemzykj6) 16

[3.2.2 База данни](#_ewax59yesirc) 18

[3.2.3 Структура на файловете](#_lo6ol5we9mdi) 23

[3.3. Описание на програмната реализация](#_usgyx0kez031) 26

[**4. Ръководство за потребителя**](#_juxanetcandc) **34**

[4.1. Административна част](#_j0dbuu1d85g7) 34

[4.2. Публична част](#_ezn5usah0k99) 43

[**5. Заключение**](#_iabwzgu82v66) **49**

[5.1. Обобщение на постигнатите резултати](#_17fgddsznnjw) 49

[5.2. Възможности за бъдещето развитие и усъвършенстване](#_5sdny9vh023e) 49

[**6. Използвана литература**](#_4rwj0zkfn5jl) **51**

[**7. Списък със съкращенията**](#_3o9nwfppm284) **52**

[**8. Индекс на фигурите**](#_wsys62z4bzw5) **53**

## 

## 2. Цели и задачи

Основната цел на настоящият дипломен проект е създаването на новинарски уеб сайт, в който ще може да се добавят новини. Новините трябва да бъдат класифицирани чрез различни методи с цел по-бърз достъп до желаното съдържание. Важен е акцента върху проблема с фалшивите новини. За решението му е избрал колаборативен подход чрез който всеки един потребител взима участие в “изграждането” на автентичността на дадена новина.

Сайтът ще има както публична, така и административна част, позволяваща на потребители с определена роля (например “редактори” или “администратори”) да управляват различните типове съдържание. Можем да заключим, че сайтът в основата си ще представлява базов CMS.(1)

За да постигнем интуитивен интерфейс, ще използваме различни UX (2) техники, чрез които ще се акцентира върху важна информация, която би могла да представлява интерес за нашите бъдещи потребители. В началната страница посетителите ще могат да видят “слайдове” с актуална информация по важни новини от последните часове. В основното навигационно меню ще бъдат изброени различните категории, като по този начин хората ще имат по-бърз достъп до информацията, която търсят. Потребителите като гости ще могат да преглеждат новините и, за да коментират, ще е необходимо да си създадат профил в сайта.

Регистрираните потребители освен да коментират, ще могат да реагират на всеки един коментар чрез бутоните “харесвам” и “не харесвам”, а също така да гласуват дали новината, която гледат според тях е фалшива или не. След като са гласували ще могат да видят по какъв начин са гласували и останалите потребители. Сайтът ще разполага и със статични страници където посетителите могат да получат повече информация за сайта портала.

Администраторската част ще дава възможност на потребители с определена роля да управляват съдържанието от следните типове:

**Новини** - въвеждане, редактиране и изтриване

**Категории** - въвеждане, редактиране и изтриване

**Тагове** - въвеждане, редактиране и изтриване

**Слайдъри** - въвеждане, редактиране (управление на слайдовете към тях), изтриване;

**Коментари** - одобрение и изтриване

**Потребители** - въвеждане, редактиране и изтриване

**Настройки** - базови настройки свързани с работата на сайта

Информацията в сайта ще бъде класифицирана по няколко критерии:

**Категории** - това е основния тип класификатор, като всяка новина може да принадлежи към само една категория.

**Тагове**  - това е по-общия тип класификатор, всяка една новина може да принадлежи към множество тагове.

Потребителите ще могат да видят списък със всички новини, както в дадена категория, така и маркирана с даден таг.

Използвайки изчистен потребителски интерфейс и доказани UX шаблони, чиято основна задача е да намалят когнитивното напрежение, ние предоставяме на нашите потребители сайт, който те биха използвали за да научат последни новини от заобикалящия ги свят.

#### 2.1 Структура на дипломната работа

Дипломната работа се състои от увод, цели и задачи, проектиране и реализация, ръководство за потребителя, заключение, списък на съкращенията, списък на фигурите и използвана литература.

**Уводът** описва проблемната област, върху която се акцентира в дипломната работа.

**Целите и задачите** включват основни дефиниции, кратко описание на функционалностите и подходи, които са използвани по време на реализацията на дипломната работа.

**Проектиране и програмна реализация** се изброяват използваните технологии и архитектури върху които е базирана текущата разработка. Добавени са извадки (под формата на код) за основни функционалности, които са реализирани в рамките на проекта.

**Ръководство за потребителя** включва описание за начина по който трябва да се използва от потребителите, съответно за публичната и администраторската му част.

**Заключението** включва обобщение на реализацията, както и насоки за бъдещето развитие и усъвършенстване на проекта.

## 3. Проектиране и програмна реализация

#### 3.1 Обзор на информационните технологии и средства

**Клиент - сървър архитектура** емрежова архитектура, която отделя клиента от сървъра, като най-често информацията се предава чрез даден TCP/IP (3) протокол, например HTTP.

Най-елементарният типов клиент-сървър архитектура се състои от две части — от една страна е сървъра, от друга страна е клиента. Комуникацията между двете страни е задължителна.

В нашият случай, архитектурата работи по следния начин:

1. Клиента отваря сайта и браузърът изпраща заявка през HTTP протокола към сървъра.

2. Сървъра получава заявката, обработва я за да разбере по какъв начин трябва да я обработи.

3. Обработването може да включва изпълнение на програмен код, комуникация с БД и т.н.

4. След като приключи обработката, сървъра изпраща отговор към клиента, който съдържа резултата от заявката.

**База данни** - релационните СУБД (системи за управление на база данни) се базират върху основни принципи от релационната алгебра. Първоначалната концепция за релационните бази данни е разработена от Едгар Код, който създава релационния модел докато работи в IBM. При този модел данните се съхраняват във взаимо-свързани таблици, изградени от колони, които могат да съдържат различен тип данни. Всеки нов запис в таблицата представлява нов ред в нея. Както подсказва името, между таблиците могат да бъдат изградени силни “релации”, които гарантират автономоност на данните съхранявани в тях.

**Основни термини свързани с релационните бази данни**

**Първичен ключ -** това е колона, която има уникална стойност измежду всички записи в таблицата. Също така, тази колона дефинирана като PK може да се използва за създаваеето на релация към текущата таблица от друга таблица.

**Външен ключ** - чрез използването на външни ключове (foreign keys) ние се подсигуряваме, че данните в таблиците, които са свързани по някакъв начин ще бъдат консистентни при извършване на различни операции върху тях. Поле дефинирано като FK, винаги трябва да сочи към PK от друга таблица.

**Индекс** - това е вътрешна структура от данни, която ни осигурява бърз достъп до данни за да избегнем сканирането на цялата таблица (т.нар. “full table scan”) при изпълнението на различни заявки към базата данни.

**Релационни отношения**

**1:1 -** един запис от дадена таблица е свързан със само един запис от втората таблица

**1:n** - един запис от дадена таблица може да бъде свързан със много други записи от втората таблица

**n:n** - един запис от дадена таблица може да бъде свързан с много записи от втората таблица, и съответно един запис от втората таблица може да бъде свързан с множество записи от първата таблица.

**SQL** - е съкращение от Structured Query Language. SQL е проектиран за комуникация с бази от данни. За разлика от другите езици като Java, PHP и т.н, езикът има ограничен набор от инструменти, но за сметка на това е достатъчно ефективен за работата, която трябва да върши.

**Какви са предимствата на SQL?**

* SQL е език, който работи с различни DBMS, които го поддържат. Тоест, ако човек научи SQL ще може да го използва както с MySQL, така и с PostgreSQL.
* SQL е лесен за научаване.
* SQL е изключително мощен инструмент за извършване на различни операции свързани с бази от данни.

**MySQL** - е най-популярната релационна система за управление на SQL бази данни с отворен код. Тя се разпространява и поддържа от Oracle Corporation. MySQL позволява съхранението на данни от най-различен вид, както и различен набор от инструменти за създаване на високопроизводителни БД, които могат да издържат на огромни натоварвания. Голяма част от езиците за програмиране днешно време съдържат програмни интерфейси за комуникация с БД.

**WAMP** - съвкупност от софтуерни продукти, необходими за реализирането и стартирането на уеб проект. Включва в себе си Apache, MySQL, PHP и работи на операционна система Windows. Съществува и еквивалент за Linux-базирани операционни системи, който се нарича LAMP.

**PHP** - скриптов език за програмиране с отворен код, който се използва предимно за уеб-програмиране. За изпълнението на програмния код се грижи интерпретатор, който по време на изпълнение на скрипта (т.нар. “run-time”) обработва входящите данни, изпълнява инструкциите описани в изходния код и връща изходящите данни обратно. За разлика от Java (заедно с Java Web Server), PHP не идва с вграден сървърен софтуер, поради тази причина най-често се използва Apache като такъв.

В проекта сме използвали PHP 7, като основните разлики между него и предходните стабилни версии на езика са следните:

1. Бързодействие - според различни независими източници, последната версия на PHP изпълнява кода почти два пъти по-бързо от колкото предходните.
2. По-добър механизъм за прихващане на грешки и неочаквано поведение на кода. Част от т.нар. “фатални грешки” в предишните версии вече могат да бъдат уловени чрез подобреният механизъм за прихващането на грешки и да бъдат обработени адекватно зависимост от ситуацията.
3. Указване на типа на данни, който бива върнат от функциите. В основата си PHP е динамично типизиран език , което често води до различни грешки, които е трудно да бъдат хванати. [1]

**Composer** е инструмент за управление на зависимостите при изпълнението на големи PHP проекти. Пакетите, сходно на NPM се съхраняват в основно хранилище и програмистите лесно могат да ги достъпят и последствие добавят към собствените си проекти. Последствие, управлението на тези проекти става чрез изпълнение на различни команди. Чрез използване на Composer, имаме "централизирано" място където се държат всички зависимости и управлението им е сравнително лесно, като същевременно можем да се възползваме от механизма на PHP за автоматично зареждане на файлове.

**Apache** е един от най-често използваните уеб сървъри в днешни дни. Уеб сървърите представляват "хранилище" за файлове, като технически те приемат, обработват и връщат отговор на HTTP (т.к. Apache e HTTP/HTTPS уеб сървър) заявки направени от клиенти. В световен мащаб, над 70% от уеб сайтовете използват Apache като уеб сървър.

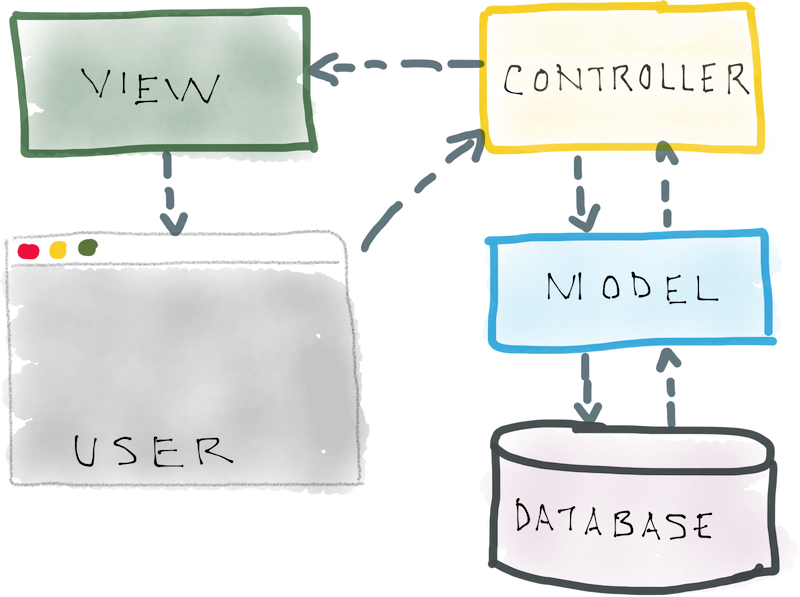
Приложението стартира на много операционни системи, включително Unix, GNU, FreeBSD, Linux, Solaris, Mac OS X, Microsoft Windows, OS/2, Novell NetWare и други платформи.

Apache се разработва от отворено общество от разработчици – Apache Software Foundation. Обществото поддържа и разработва множество други проекти най-важните от които са Apache Ant, Apache SpamAssassin, Apache Tomcat и огромен брой проекти основани на XML езика, но Апачи е най-известния и широко разпространен продукт. [2]

**Laravel Framework** - Laravel е уеб фреймуърк (2) с безплатен отворен код на PHP, създаден от Тейлър Отуел и предназначен за разработка на уеб приложения, следващи схемата model–view–controller (MVC). Някои функции на Laravel са: модулна система за пакети, които могат да се използват напълно независимо, различни начини за достъп до релационни бази данни.

Към март 2015 г. Laravel се смята за един от най-популярните PHP фреймуъркове, заедно със Symfony, Nette, Codeigniter, Yii2 и други.

Изходния код на базата на Laravel се съхранява в GitHub и се лицензира съгласно условията на MIT лиценз. [3]



**Фигура 1.** Принцип на работа на MVC шаблона.

**Blade** - въпреки, че PHP сам по себе си е език, който може да се използва в темплейти, Laravel предлага собствен "template engine", който е вдъхновен от .NET Razor engine-a. Той разполага с гъвкав синтаксис, възможност за наследяване, както и множество мощни функции, които спомагат създаването на изгледи в Laravel.

*Примерен код:*

<**html**>

<**head**>

<**title**>App Name - **@yield('title')**</**title**>

</**head**>

<**body**>

**@section('sidebar')**

This is the master sidebar.

**@show**

<**div class="container"**>

**@foreach (**$users **as** $user**)**

**@if (**$loop->**first)**

This is the first iteration.

**@endif**

**@if (**$loop->**last)**

This is the last iteration.

**@endif**

<**p**>This is user {{ $user->**id** }}</**p**>

**@endforeach**

// Дали да се добави template-a зависи от стойността на $boolean променливата.

**@includeWhen(**$boolean, **'view.name'**, [**'some'** => **'data'**]**)**

**@if (***count*($users) === 1**)**

I have one record!

**@elseif (***count*($users) > 1**)**

I have multiple records!

**@else**

I don't have any records!

**@endif**

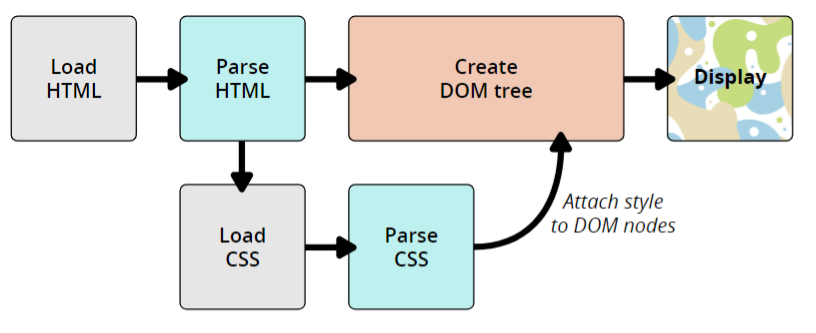
</**div**>

</**body**>

</**html**>

**HTML** e съкращение от “**H**yper**T**ext **M**arkup **L**anguage” е език, който описва структурата на уеб страниците. Често бива бъркан с език за програмиране, но всъщност, както и самото име подсказва е "описващ" език. HTML представлява стандарт в интернет, като правилата биват разписвани и одобряване от международния консорциум W3C. В HTML, описанието на елементите става посредством т.нар. "тагове", които оформят различни елементи от нашата уеб страница. HTML документите са обикновени текстови файлове и могат да бъдат създадени, или редактирани чрез текстов редактор, но за улеснение на програмистите е желателно да се използва IDE, т.е. интегрирана среда за разработка. Последната версия на HTML стандарта е HTML 5.0, или иначе казано HTML5. [4]

**CSS** е съкръщение от Cascading Style Sheets и се използва за визуално оформление на HTML страниците. В основата си представлява съвкупност от правила и начина им на прилагане, които казват на браузъра как точно трябва да изглеждат елементите.



**Фигура 2.** Принцип на работа на CSS в браузъра.

**SASS** е съкращение от "Sassy CSS" представлява надмножество (т.нар. "super set") на синтаксиса на CSS3, което на практика ознаава, че всеки валиден CSS3 код е също така валиден SASS код. SASS върви заедно с компилатор, който "превежда" кода към чист CSS, който е разбираем за браузърите. Част от възможностите, които ни дава SASS са: използване на променливи в кода, използване на т.нар. "миксини" (позволяващи преизползването на код), вграждането (чрез което се избягва повторението на вложените селектори един в друг), селекторно наследяване и много други.

*Примерен код:*

***@mixin square****(****$size****,* ***$color****) {*

***width****:* ***$size****;*

***height****:* ***$size****;*

***background-color****:* ***$color****;*

*}*

*.****small-blue-square*** *{*

***@include square****(20****px****,* ***rgb****(0,0,255));*

*}*

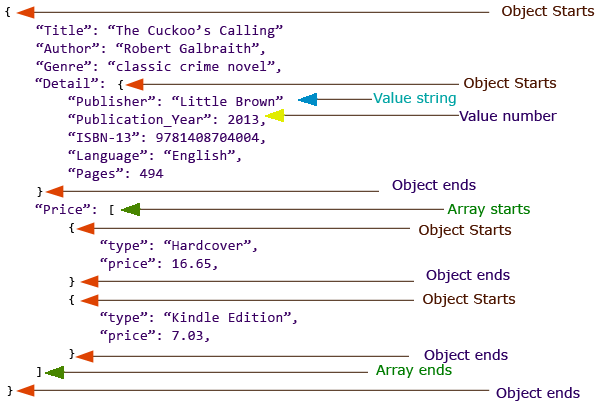
*.****big-red-square*** *{*

***@include square****(300****px****,* ***rgb****(255,0,0));*

*}*

**JavaScript** - е един от най-важните езици за програмиране в днешно време, т.к. това е езикът на уеб браузърът, но това не изключва и възможността да се изпълнява и на сървърната част. JavaScript е т.нар. "first class", защото всяка една функция в действителност има тип "Object". В основата си JavaScript е "weak typed" език, т.е. компилаторът не засича грешки свързани с използването на несъответстващи по тип данни. Една от най-силните страни на езика е т.нар. “prototypal inheritance”, което дава възможност на обекти от един тип да наследяват директно свойства от други обекти, което се различава до някъде от основните принципи на обектно-ориентиdраното програмиране. Езикът лежи върху стандарти, които са съгласувани и реализирани от всички браузъри, като този стандарт непрекъснато се усъвършенства за да е в крак, най-масовата версия на използвана в днешни дни е ECMAScript 2016. Въпреки приликите си със Java, като идеология езикът е по-близо до езиците Lisp и Scala. [5]

**JSON** - JavaScript Object Notation е текстови базиран стандарт създаден за пренос на данни. Изграден е на базата на обектите в JavaScript и цели лесно представяне на данни. Въпреки връзката си със JavaScript, JSON обектите се използват масово във почти всички езици за програмиране, които разполагат със библиотеки, които се грижат за обработването на подобни структури от данни.



**Фигура 3.** Примерен JSON обект.

**NPM** - това е “package manager” за JavaScript, чиято основна идея е да даде възможност на програмистите сравнително лесно да управляват различни зависимости в техните проекти, достъпвайки кода до различни пакети от основно “хранилище”. В модерните приложения писани върху Angular, React, Vue.js това е безценен помощник, защото в тях се използва солиден набор от пакети и управлението им без подобен инструмент би било невъзможно.

**jQuery** е "cross-browser" (т.е. работеща на всички основни браузъри) библиотека за JavaScript, която е проектирана за да улесни и опрости писането на JavaScript код, като успоредно с това е оптимизирана достатъчно добре с цел бързодействие. В практиката, с няколко реда код написан на jQuery бихме могли да си спестим множество редове написани на чист JavaScript.

**Vue.js** е "progressive" (т.е. може лесно да бъде вграждан в съществуващи приложения) фреймуърк за създаване на потребителски интерфейси. Основният му фокус е да улесни създаването на интерфейса, като размерът на библиотеката е малък и интеграцията със вече изградени страници е изключително лесна. Vue има солиден набор от инструменти, които му позволяват също така да бъде удачен избор за създаването на SPA ("Single Page Applications") в уеб пространството.

**Bootsrap** e безплатен и същевременно най-популярният в днешни дни "full-stack" фреймуърк за разработка на адаптивни (т.нар. "responsive") уеб сайтове, който лежи върху HTML5 и CSS3. Той дава възможност на програмистите да използват солиден набор от UI (User Interface) елементи, като същевременно прави надграждането им лесна и приятна задача. Напълно съвместим е със всички основни браузъри, които се използват в днешни дни, също така е лесен за учене.

**Адаптивен уеб дизайн -** това е подход, който предполага, че проектирането и разработването трябва да отговорят на поведението на потребителя, въз основа на размера на екрана, платформата и ориентацията. Отзивчивият уеб дизайн ни гарантира, че сайтът ни ще изглежда добре на всички възможни устройства през които потребителят може да го достъпи. [6]

#### 

#### 3.2 Структура на приложението.

##### 3.2.1 Обща архитектура

В основната архитектура на приложението седи MVC (Model - View - Controller) шаблонът за дизайн, който основно се състои от три части:

**Модел** (англ. Model) - ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които се моделира и над който се работи - въвеждане, промяна, показване и т.н. Трябва да се прави разлика между реалния обкръжаващ свят и въображаемият абстрактен моделен свят, който е продукт на разума, който се възприема като твърдения, формули, математическа символика, схеми и други помощни средства. Например в банково приложение това са класовете, описващи клиентите, техните сметки, транзакциите, които са осъществили и т.н., както и класовете за извършване на операции над тези обекти (engines) - например клас Transfer с методи като createInterBankTransfer(), createInnerBankTransfer(), getCash() и т.н.

**Изглед** (англ. View) - тази част от изходния код на приложението, отговорна за показването на данните от модела. Например изгледът може да се състои от PHP шаблонни класове, JSP страници, ASP страници, JFrame наследници в Swing приложение. Зависи от това какъв графичен интерфейс се прави и каква платформа се използва;

**Контролер** (англ. Controller) - тази част от сорс кода (клас или библиотека), която взима данните от модела или извиква допълнителни методи върху модела, предварително обработва данните, и чак след това ги дава на изгледа. Например може да бъде създаден един малък обект, в който да бъдат сложени данните за транзакцията - като в контролера бъдат взети данните за транзакцията от модела, бъдат преведени датите от UNIX формат във четим от потребителя формат, бъде преобразувана валутата от долари в евро например, бъде закръглено до втория знак вместо да се виждат данните както са в модела (и в базата) до 10-тия. Също така когато се прави уеб графичен интерфейс това би довело до много лесна модификация на HTML кода дори от човек, който не е програмист - той ще гледа на шаблона просто като на обикновена HTML страница. [7]

**Какви са предимствата на MVC архитектурата?**

* Моделът е бива изолиран от контролера и изгледа, т.е. бихме могли да създадем повече от един изгледи, които да са свързани с даден модел.
* Всеки един компонент от тази архитектура може да се разглежда като независим, т.е. не са пряко свързани помежду си.
* Бърза и лесна разработка, тъй като повечето разработчици са запознати с концепцията.

**Примерен сценарий на работа на MVC в приложението**

Ако потребителят иска да види списък от новини в дадена категория, тогава той ще изпрати заявка към контролера, който отговаря за показването на новини в дадена категория. Методът в контролера ще комуникира с модела за да зареди списък с новините от избраната категория, и последствие ще сервира съдържанието като зареди изгледът, който отговаря за показванет она новини в дадена категория. Това с няколко думи показва затворения цикъл и работа на едно MVC приложение.

##### 3.2.2 База данни

Базата данни е място където се съхраняват данни от дадена предметна област, като същите са структурирани по определен начин. Поддържането на базата данни става чрез използването на СУБД (Система за управление на база данни). СУБД е жизнено важна част от всяко едно приложение и правилния подбор на технологичния стек, както и конфигурацията й е ключът към създаване на скалируемо приложение, което може да понесе голямо натоварване и бързодействие.

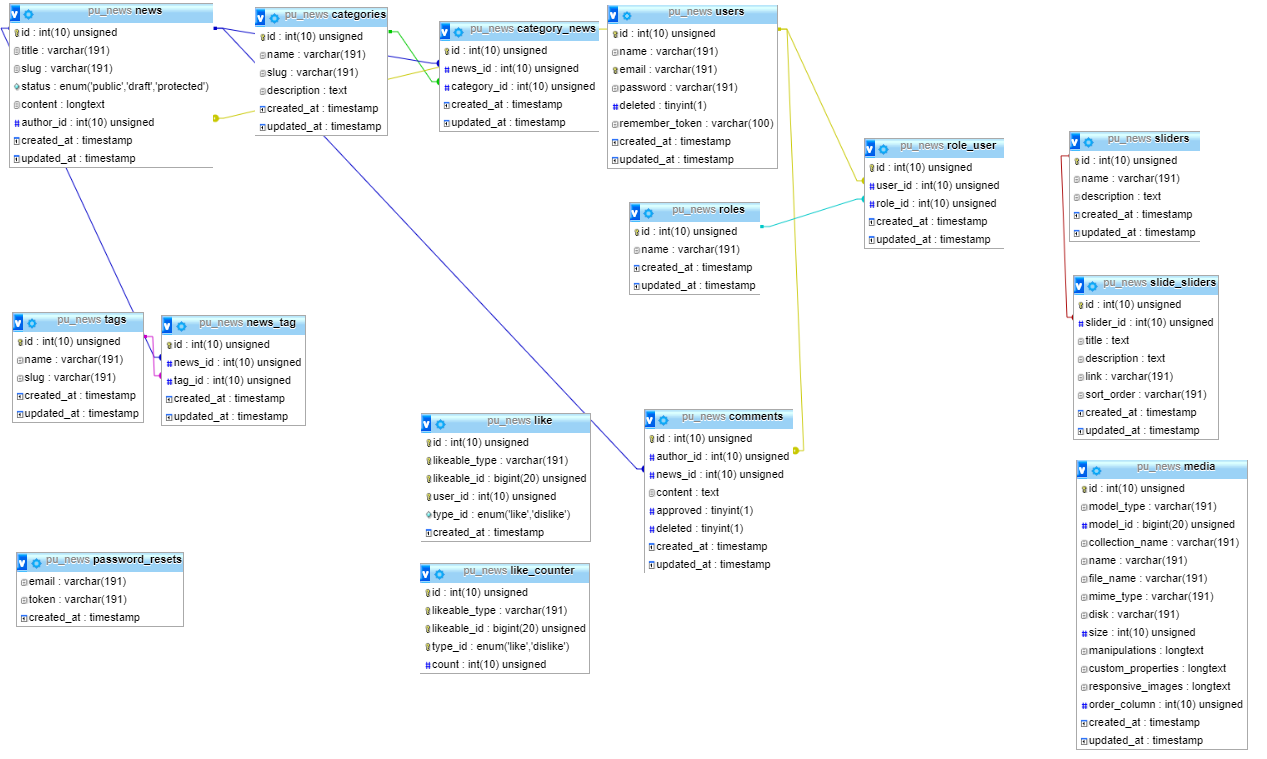
Всяка една СУБД има различни механизми за съхраняване на данните. Двете основни системи в MySQL са MyISAM и InnoDB, за целите на текущата дипломна работа съм избрал InnoDB, защото има някои основни предимства пред InnoDB, а именно [8]:

1. InnoDB поддържа “row-level” lock.
2. InnoDB съхранява индексите по добре оптимизиран начин.
3. InnoDB поддържа външни ключове.
4. InnoDB поддържа транзакции, т.е. няколко заявки могат да бъдат изпращани в “група” и съответно да бъдат възстановявани в случай на грешка.
5. InnoDB е по-бърза при обработването на гоямо количество информация.

Основните характеристики на MyISAM са следните:

1. MyISAM поддържа "table-level" locking.
2. MyISAM няма поддръжка на външни ключове, затова можем да кажем, че MyISAM не работи върху релационния модел.
3. MyISAM не поддържа транзакции.
4. MyISAM поддържа пълнотекстово търсене, след версия 5.6 на MySQL същото е налично и в InnoDB.

Таблиците и връзките между тях могат да се видят в следната диаграма:



**Фигура 4.** Схема на базата данни използвана в текущата дипломна работа.

От диаграмата се вижда, че основните таблици, които използваме са:

Таблица: **categories**

id - *уникален идентификатор (p.k)*

name - *заглавие на категорията*

slug - *user-friendly URL, пр. “/categories/****biznes****”*

description - *описание на категорията*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **category\_news**

id - *уникален идентификатор (p.k)*

news\_id - *ID на дадена новина (f.k. към p.k. в “news”)*

category\_id **-** *ID на дадена категория (f.k. към p.k. в “categories”)*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **comments**

id - *уникален идентификатор (p.k)*

author\_id - *ID на даден автор (f.k. към p.k. в “users”)*

news\_id - *ID на дадена новина (f.k. към p.k. в “news”)*

content **-** *съдържание на коментар*

approved **-** *флаг дали коментарът е одобрен*

deleted **-** *флаг дали коментарът е изтрит*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **like**

id - *уникален идентификатор (p.k)*

likeable\_type - *моделът (пр. новина, коментар) към който се отнася действието*

likeable\_id- *ID спрямо горната колона към което се отнася действието*

user\_id - *ID на потребитея, който е извършил действието*

type\_id - *извършеното действие (пр. “like” или “dislike”)*

created\_at - *дата на създаване*

Таблица: **like\_counter**

id - *уникален идентификатор (p.k)*

likeable\_type - *моделът (пр. новина, коментар) към който се отнася действието*

likeable\_id - *ID спрямо горната колона към което се отнася действието*

count - *тоталния брой на извършени действия (пр. “like” или “dislike”)*

Таблица: **media**

id - *уникален идентификатор (p.k)*

media\_type - *моделът (пр. новина) към който принадлежи прикаченият файл*

model\_id - *ID спрямо горната колона към което принадлежи прикаченият файл*

collection\_name - *име на колекцията от файлове*

name - *изрично упоменато име на файла*

file\_name - *име на физическия файл*

mime\_type - *тип на файла*

disk - *място където физически се съхранява файла*

size - *размер на файла*

manipulations - *масив в който се съхраняват производни файлове на основния (пр. “thumbnails”)*

custom\_properties - *масив в който се съхранява допълнителна информация към файла (пр. “автор”, “локация” и т.н.)*

responsive\_images - *допълнителна възможност за съхранение на responsive images, които биха могли да се използват за progressive loading и т.н.*

order\_column - *подредба на дадения файл в колекцията*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **news**

id - *уникален идентификатор (p.k)*

title - *заглавие на новината*

slug - *user-friendly URL, пр. “/news/moyata-pyrva-novina”*

status - *статус на новината (пр. “публична” или “чернова”)*

content - *съдържание на новината*

author\_id - *ID на даден автор (f.k. към p.k. в “users”)*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **news\_tag**

id - *уникален идентификатор (p.k)*

news\_id - *ID на дадена новина (f.k. към p.k. в “news”)*

tag\_id - *ID на даден таг (f.k. към p.k. в “tags”)*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **password\_resets**

id - *уникален идентификатор (p.k)*

email - *email на който да се прати линк за възстановяване*

token - *уникален идентификатор за възстановяване*

created\_at - *дата на създаване*

Таблица: **roles**

id - *уникален идентификатор (p.k)*

name - *наименование на ролята*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **role\_user**

id - *уникален идентификатор (p.k)*

user\_id - *ID на даден потребител (f.k. към p.k. в “users”)*

role\_id - *ID на дадена роля (f.k. към p.k. в “roles”)*

created\_at - *дата на създаване*

Tаблица: **sliders**

id - *уникален идентификатор (p.k)*

name - *име на слайдъра*

description - *описание на слайдъра*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **slide\_sliders**

id - *уникален идентификатор (p.k)*

slider\_id - *ID на даден слайдер (f.k. към p.k. в “sliders”)*

title - *заглавие на слайда*

description - *описание на слайда*

link - *линк на слайда*

order\_column - *подреба на слайда*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **tags**

id - *уникален идентификатор (p.k)*

name - *име на тага*

slug - *user-friendly URL за тага (пр. “tags/my-tag”)*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **users**

id - *уникален идентификатор (p.k)*

name - *име на потребителя*

password - *парола (****б.а.:*** *хеширана, не се пази в “чист” вид)*

deleted - *флаг дали потребителя е изтрит*

remember\_token - *token за функциалността “remember me”*

created\_at - *дата на създаване*

updated\_at - *дата на последно редактиране*

Таблица: **options**

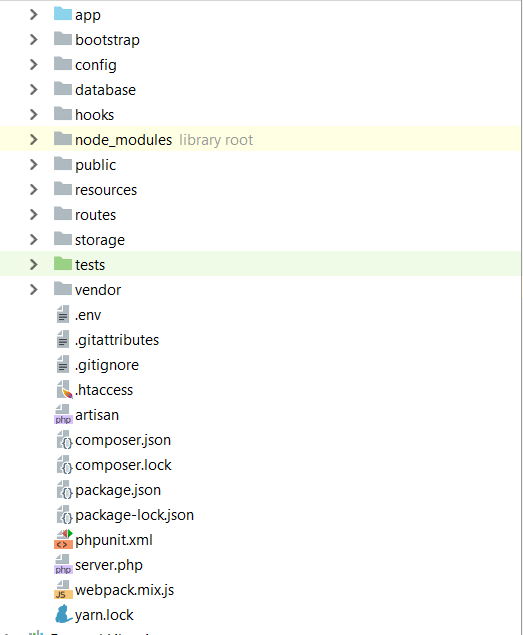
id - *уникален идентификатор (p.k)*

option\_name - *име на настройката*

option\_value - *стойност за съответната настройка*

##### 3.2.3 Структура на файловете

Laravel използва сравнително проста структура за организиране на файловете [9]:



**Фигура 5.** Структура на файловете и директориите в Laravel.

**app -** в тази директория се съхраняват основните части на нашето приложение. Тук се намират моделите и контролерите.

**bootstrap** - в тази директория се намира файлът, който “стартира” нашето приложение. Laravel използва “front-controller” шаблонът за дизайн при който имаме “централна” точка, която обработва всички входящи заявки и ги предава за обработка към вътрешните механизми. Също така тук Laravel пази вътрешен кеш за някои файлове, например кеширане на файловете свързани с рутирането.

**config -** тази директория съдържа различни конфигурационни файлове свързани с приложението.

**database -** в тази директория се съхраняват файлове свързани с базата данни. Laravel използва т.нар. “migrations” (представляват примитивен механизъм за контролиране на версиите свързан с промените, които се правят по БД), “seeds” (представляват парчета код, които се грижат за попълване на инициализиращи данни) и “factories” (което ни позволява сравнително лесно да създадем гоям набор от записи, пр. може да бъдат използвани за пълнене на тестови данни).

**node\_modules -** в тази директория се пазят всички пакети инсталирани през npm install <имепакет>. Добра практика е, ако се използва Git (или друга система за контрол на версиите) тази директория да бъде игнорирана, т.к. расте лавинообразно.

**public -** в тази директория се намират asset-ите, които използваме в нашето приложение (пр. JS файлове, CSS файлове, изображения и т.н.)

Тук също така се намира и “index.php”, който е файлът, който изпраща всички входящи заявки към логиката за стартиране на приложението, която споменахме по-горе.

**resources -** в тази директория основно се съхраняват нашите “изгледи”, както и в случай, че използваме някакъв вид “pre-processor” за CSS (например SASS, както в нашия случай), тук се намират и въпросните SASS файлове, които биват компилирани и последствие запазени в “/pubilc” директорията.

**routes -** в тази директория описваме правилата за “routing” (пътищата в URL-a) в нашето приложение, смислено те са разделени на няколко части - за публичната част, за административната част и др.

**storage**  - в тази директория се съхраняват компилираните файлове от “template engine”-a, който се използва от Laravel, различни видове логове генерирани по време на работа. Тук също така биха могли да се съхраняват файлове генерирани от потребителите (пр. качени файлове).

**tests**  - тук се пазят автоматизирани тестове, в случай, че програмистите са написали такива за различните функционалности, които приложението предлага.

**vendor -** тук се съхраняват всички зависимости управлявани от Composer.

#### 3.3. Описание на програмната реализация

**Контролерите** са разделени в няколко файла и съответно организирани в директории, защото част от тях отговарят за обработка на заявки в администраторската част, докато другата част отговаря за обработка на заявки в публичната част на портала. В случая използването на един и същ контролер не беше уместно, защото изпълняват различни задачи.

Показаният код по-долу отговаря за съхранение на новините в базата данни, както и запазването на принадлежащите към тях атрибути за класифициране, например “категория” или “таг”. Зависимост от заявката, която потребителят е изправил, кодът ще се погрижи да се извършват коректните операции. Валидирането на входящите данни е част от работата на контролера, като в случай, че валидацията е неуспешна, потребителите ще бъдат уведомени чрез грешка в потребителския интерфейс.

**Файл**: *app/Http/Controllers/Admin/NewsController.php*

**<?php**

**namespace** App\Http\Controllers\Admin;

**use** App\Models\Category;

**use** App\Models\News;

**use** App\Models\Tag;

**use** App\Models\User;

**use** Illuminate\Http\Request;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Support\Facades\Auth;

**class** NewsController **extends** Controller

{

*/\*\**

*\** ***@return*** *\Illuminate\Contracts\View\Factory|\Illuminate\View\View*

*\*/*

**public function** index()

{

$news = News::*withCount*(**'comments'**)

->where(**'status'**, **'='**, **'public'**)

->latest()

->paginate(25);

**return** view(**'backend.news.index'**, *compact*(**'news'**));

}

*/\*\**

*\** ***@return*** *\Illuminate\Contracts\View\Factory|\Illuminate\View\View*

*\*/*

**public function** create()

{

*// Когато потребителят се опита да създаде новина, ние автоматично създаваме такава със статус “чернова”*

*// Това е необходимо за да позволим на потребителите директно да закачат файлове към въпросната новина.*

*// Също така, по този начин можем да реализираме функционалност, която автоматично да съхранява работата на потребителите.*

$news = News::*create*([**'author\_id'** => Auth::*user*()->**id**, **'status'** => **'draft'**]);

$categories = Category::*all*()->pluck(**'name'**, **'id'**);

$tags = Tag::*all*()->pluck(**'name'**, **'id'**);

*// Автори могат да бъдат всички потребители с роля != “нормалната”, т.е. “администратор” или “редактор”*

$users = User::*authors*()->pluck(**'name'**, **'id'**);

*// Всички файлове асоциирани с нашата новина.*

$media = News::*find*($news->**id**)->getMedia();

**return** view(**'backend.news.create'**,

*compact*(**'news'**, **'users'**, **'categories'**, **'tags'**, **'media'**));

}

*/\*\**

*\** ***@param*** *$id*

*\** ***@return*** *\Illuminate\Contracts\View\Factory|\Illuminate\View\View*

*\*/*

**public function** edit($id)

{

$news = News::*find*($id);

**if** (*is\_null*($news))

{

**return** redirect()->route(**'admin.news.index'**);

}

$categories = Category::*all*()->pluck(**'name'**, **'id'**);

$tags = Tag::*all*()->pluck(**'name'**, **'id'**);

*// Автори могат да бъдат всички потребители с роля != “нормалната”, т.е. “администратор” или “редактор”*

$users = User::*authors*()->pluck(**'name'**, **'id'**);

$media = News::*find*($news->**id**)->getMedia();

**return** view(**'backend.news.edit'**,

*compact*(**'news'**, **'users'**, **'categories'**, **'tags'**, **'media'**));

}

*/\*\**

*\** ***@param*** *Request $request*

*\** ***@param*** *$id*

*\** ***@return*** *\Illuminate\Http\RedirectResponse*

*\*/*

**public function** update(Request $request, $id)

{

$this->validate($request, **array**(

**'title'** => **'required|max:255'**,

**'categories.\*'** => **'required|integer'**,

**'tags.\*'** => **'required|integer'**,

**'content'** => **'required'**

));

$news = News::*find*($id);

**if** (*is\_null*($news))

{

**return** redirect()->route(**'admin.news.index'**);

}

*// Запис в БД на новината след добавянето й*

$news->update($request->all() + [**'status'** => **'public'**]);

*// Запис на категориите към новината*

$categories = (array)$request->get(**'categories'**);

**if** (*count*($categories) > 0)

{

$news->categories()->sync($categories);

}

*// Запис на таговете към новината*

$tags = (array)$request->get(**'tags'**);

**if** (*count*($tags) > 0)

{

$news->tags()->sync($tags);

}

**return** redirect()

->route(**'admin.news.edit'**, $news->**id**)

->with(**'success'**, \_\_(**'You have edited news item successfully.'**));

}

*/\*\**

*\** ***@param*** *News $news*

*\** ***@return*** *\Illuminate\Http\RedirectResponse*

*\** ***@throws*** *\Exception*

*\*/*

**public function** destroy(News $news)

{

$deleted = $news->delete();

**if** (!$deleted)

{

**return** back()->withError(\_\_(**'We were not able to delete the item. Please try again later.'**));

}

**return** back();

}

}

**Файл**: *app/Http/Controllers/Admin/NewsController.php*

**<?php**

**namespace** App\Http\Controllers;

**use** App\Models\Slider;

**use** App\Models\Option;

**use** App\Models\News;

**use** Illuminate\Http\Request;

**class** NewsController **extends** Controller

{

**public function** index()

{

*// Вземаме слайдерът, който искаме да се покаже от таблицата с настройки*

$slider = Slider::*find*(Option::*find(***'option\_name', 'homepage\_slider\_idx'***)*);

*// Вземаме списък с последните добавени новини, т.к. ги показваме в потребителския интерфейс*

$latestNews = News::*orderBy*(**'id'**, **'desc'**)->take(5)->get();

**return** view(**'frontend.news.index'**, *compact*(**'slider'**, **'latestNews'**));

}

**public function** show($slug)

{

$news = News::*with*(**'comments'**, **'comments.author'**, **'comments.likesCounter'**, **'comments.dislikesCounter'**)

->where(**'slug'**, $slug)

->first();

**if** (*is\_null*($news))

{

*// Ако новината не е намерена, нека препратим потребителя в началната страница.*

**return** redirect()->route(**'home'**);

}

**return** view(**'frontend.news.single'**, *compact*(**'news'**, **'comments'**));

}

}

**Моделите** се съхраняват на едно място, т.к. се преизползват едни и същи файлове, както за административната, така и за публичната част.

В моделите използваме ORM (4), което представлява инструмент за съпоставяне на модел на таблица в БД със даден ООП клас в нашия PHP код. По този начин имаме възможност да програмираме чрез императивен подход начина по който нашето приложение си говори с базата данни. Чрез ORM-а получаваме едно допълнително ниво на абстракция и не е нужно да пишем чист SQL код, тъй като това се случва автоматично от кода свързан с управлението на този инструмент.

**Файл:** *app/Models/News.php*

**<?php**

**namespace** App\Models;

**use** Cviebrock\EloquentSluggable\Sluggable;

**use** Illuminate\Database\Eloquent\Model;

**use** Illuminate\Database\Eloquent\Relations\BelongsToMany;

**use** Illuminate\Database\Eloquent\Relations\BelongsTo;

**use** Illuminate\Database\Eloquent\Relations\HasMany;

**use** Spatie\MediaLibrary\HasMedia\HasMedia;

**use** Spatie\MediaLibrary\HasMedia\HasMediaTrait;

**use** Spatie\Image\Manipulations;

**use** Spatie\MediaLibrary\Models\Media;

**class** News **extends** Model **implements** HasMedia

{

**use** Sluggable;

**use** HasMediaTrait;

**protected $fillable** = [**'author\_id'**, **'slug'**, **'status'**, **'title'**, **'content'**];

**public function** registerMediaConversions(Media $media = **NULL**)

{

// За да изглеждат добре потребитеските изображения, създаваме няколко размера, които показваме на

// различни места в потребителския интерфейс.

$this

->addMediaConversion(**'square\_100'**)

->crop(Manipulations::***CROP\_CENTER***, 100, 100);

$this

->addMediaConversion(**'rect\_750\_350'**)

->crop(Manipulations::***CROP\_CENTER***, 750, 350);

$this

->addMediaConversion(**'rect\_350\_200'**)

->crop(Manipulations::***CROP\_CENTER***, 350, 200);

$this

->addMediaConversion(**'thumbnail'**)

->width(150)

->height(150);

$this

->addMediaConversion(**'medium'**)

->width(350)

->height(350);

$this

->addMediaConversion(**'large'**)

->width(1024)

->height(1024);

}

**public function** sluggable()

{

**return** [

**'slug'** => [

**'source'** => **'title'**

]

];

}

**public function** getRouteKeyName()

{

**return 'slug'**;

}

**public function** comments(): HasMany

{

**return** $this->hasMany(Comment::**class**);

}

**public function** categories(): BelongsToMany

{

**return** $this->belongsToMany(Category::**class**);

}

**public function** author(): BelongsTo

{

**return** $this->belongsTo(User::**class**, **'author\_id'**);

}

**public function** tags(): BelongsToMany

{

**return** $this->belongsToMany(Tag::**class**);

}

}

**Изгледите** са файлове, които представляват начина по който изглеждат части от нашето приложение. Там също е важна йерархичната подредба и възможността за преизползване на парчета код за да можем да следваме добрите практики в програмирането. Аналогично на контролерите са разделени спрямо административната и публичната част.

В примерните по-долу се вижда как имаме два template-a съответно за добавяне и редактиране на новина, като и двата преизползват един общ, който визуализира формата, която е една и съща и за двата с тази разлика, че при изгледа за редакция трябва да бъде предварително попълнена със съответните стойности спрямо редактираната новина.

**Файл**: *resources/views/backend/news/index.blade.php*

**@extends('backend.layouts.master')**

**@section('title'**, **'| '** . \_\_(**List news'**)**)**

**@section('content')**

<**div class="card"**>

<**div class="card-header bg-light"**>

{{ \_\_(**'List news'**) }}

<**div class="pull-right"**>

<**a href="**{{ route(**'admin.news.create'**) }}**" class="btn btn-primary"**>

{{\_\_(**'Create'**)}}

</**a**>

</**div**>

</**div**>

<**div class="card-body"**>

**@if (**$message = Session::get(**'message'**)**)**

<**div class="alert alert-primary"**>

<**p**>{{ $message }}</**p**>

</**div**>

**@endif**

<**div class="table-responsive"**>

<**table class="table"**>

<**tr**>

<**th**>{{\_\_(**'ID'**)}}</**th**>

<**th**>{{\_\_(**'Name'**)}}</**th**>

<**th**>{{\_\_(**'Created at'**)}}</**th**>

<**th**>{{\_\_(**'Updated at'**)}}</**th**>

<**th width="100px"**>{{\_\_(**'Actions'**)}}</**th**>

</**tr**>

**@foreach (**$news **as** $item**)**

<**tr**>

<**td**>{{$item->**id**}}</**td**>

<**td**>{{$item->**title**}}</**td**>

<**td**>{{date(**'M j, Y H:m'**, strtotime($item->**created\_at**))}}</**td**>

<**td**>{{date(**'M j, Y H:m'**, strtotime($item->**updated\_at**))}}</**td**>

<**td width="100px"**>

<**a href="**{{ route(**'admin.news.edit'**, [**'id'** => $item->**id**]) }}**" class="mr-1"**>

<**i class="fa fa-edit"**></**i**>

</**a**>

<**a href="#" title="**{{\_\_(**'admin.news.\_index.delete'**)}} **"**>

{!! Form::open(**array**(**'method'** => **'DELETE'**, **'route'** => [**'admin.news.destroy'**, $item], **'class'** => **'d-inline'**)) !!}

{!! Form::button(**'<i class="fa fa-minus-circle" aria-hidden="true"></i>'**, **array**(**'type'** => **'submit'**, **'class'** => **'remove-icon'**)) !!}

{!! Form::close() !!}

</**a**>

</**td**>

</**tr**>

**@endforeach**

</**table**>

</**div**>

<**div class="d-flex justify-content-center"**>

{{ $news->links() }} <!-- Странициране -->

</**div**>

</**div**>

</**div**>

**@endsection**

**Файл:** *resources/views/backend/news/create.blade.php*

**@extends('backend.layouts.master')**

**@section('title'**, \_\_(**'Create news'**)**)**

**@section('content')**

<**div class="container-fluid"**>

<**div class="row bg-light text-dark p-2"**>

<**div class="col-md-8"**>

{!! Form::*model*($news, [**'route'** => [**'admin.news.update'**, $news->**id**], **'method'** => **'PUT'**]) !!}

<!-- тук преизползваме общата част за формата -->

**@include('backend.news.\_form'**, [**'title'** => \_\_(**'Create news')**]**)**

{!! Form::*close*() !!}

</**div**>

<**div class="col-md-4"**>

<!-- преизползваме общата част за управление на файлове -->

**@include('backend.news.\_media')**

</**div**>

</**div**>

</**div**>

**@endsection**

**Файл:***resources/views/backend/news/edit.blade.php*

**@extends('backend.layouts.master')**

**@section('title'**, \_\_(**'Edit news'))**

**@section('content')**

<**div class="container-fluid"**>

<**div class="row bg-light text-dark p-2"**>

<**div class="col-md-8"**>

{!! Form::*model*($news, [**'route'** => [**'admin.news.update'**, $news->**id**], **'method'** => **'PUT'**]) !!}

<!-- тук преизползваме общата част за формата -->

**@include('backend.news.\_form'**, [**'title'** => \_\_(**'Edit news')**]**)**

{!! Form::*close*() !!}

</**div**>

<**div class="col-md-4"**>

**@include('backend.news.\_media')**

</**div**>

</**div**>

</**div**>

**@endsection**

**Файл**: *resources/views/backend/news/\_form.blade.php*

**@section('styles')**

<**script src="//cdn.tinymce.com/4/tinymce.min.js"**></**script**>

<**script**>

**tinymce**.init({

**selector**: **'textarea'**,

**plugins**: **'link code'**,

**menubar**: **false**

});

</**script**>

**@endsection**

<**div class="card"**>

<**div class="card-header bg-light"**>

{{ $title }}

</**div**>

<**div class="card-body"**>

**@include('backend.partials.\_form\_errors')**

**@include('backend.partials.\_form\_success')**

<**div class="form-group"**>

{{ Form::*label*(**'title'**, \_\_(**'Title'**), [**'class'** => **'form-control-label'**]) }}

{{ Form::*text*(**'title'**, **null**, [**'class'** => **'form-control input-lg'**]) }}

</**div**>

<**div class="form-group"**>

{{ Form::*label*(**'category\_id'**, \_\_(**'Category'**), [**'class'** => **'form-spacing-top'**]) }}

{{ Form::*select*(**'categories[]'**, $categories, $news->**categories**, [**'class'** => **'form-control'**]) }}

</**div**>

<**div class="form-group"**>

{{ Form::*label*(**'author\_id'**, \_\_(**'Author'**), [**'class'** => **'form-spacing-top'**]) }}

{{ Form::*select*(**'author\_id'**, $users, **null**, [**'class'** => **'form-control'**]) }}

</**div**>

{{ Form::*label*(**'tags'**, \_\_(**'Tags'**), [**'class'** => **'form-spacing-top'**]) }}

{{ Form::*select*(**'tags[]'**, $tags, $news->**tags**, [**'class'** => **'form-control select2-multi'**, **'multiple'** => **'multiple'**]) }}

{{ Form::*label*(**'content'**, \_\_(**'Content'**), [**'class'** => **'form-spacing-top'**]) }}

{{ Form::*textarea*(**'content'**, **null**, [**'class'** => **'form-control'**]) }}

{{ Form::*submit*(\_\_(**'Save'**), **array**(**'class'** => **'btn btn-success btn-lg btn-block'**, **'style'** => **'margin-top: 20px;'**)) }}

</**div**>

</**div**>

**Файл:** *resources/views/backend/partials/\_form\_errors.blade.php*

**@if(***count*($errors) > 0**)**

**@foreach (**$errors->get(**'title'**) **as** $error**)**

<**div class="alert alert-danger"**>

{{$error}}

</**div**>

**@endforeach**

**@endif**

В изгледите може да се види следния синтаксис:

{{ \_\_(‘Entity’) }}

Това е вградения синтаксис в Laravel за превод на текстове, които визуализираме. Преводите се пазят в JSON формат за всеки език, като името на “ключът” в обектите трябва да отговар на това, което сме използвали в темплейта.

**Middleware**

Middleware представлява механизъм, който служи за филтриране на входящите HTTP заявки. Например, Laravel има вграден механизъм, който се грижи за това да провери дали потребителя е вписан в системата и в случай, че не е да го пренасочи към страницата за вход в системата. Ако потребителят е авторизиран ще може да извърши действието, което се опитва.

Понеже Laravel няма вграден механизъм за потребителски роли сме създали такъв, който да решава дали потребителя може да продължи в зависимост от ролята.

**<?php**

**namespace** App\Http\Middleware;

**use** Closure;

**use** Illuminate\Http\Request;

**class** RoleMiddleware

{

**public function** handle(Request $request, Closure $next, string $role)

{

**if** (!$request->user()->hasRole($role))

{

**return** redirect()->route(**'home'**)->withErrors(\_\_(**'auth.not\_authorized'**));

}

**return** $next($request);

}

}

Този механизъм е особено полезен, тъй като може да бъде описан във routing-a и да се превърне в гъвкав начин за централизиран контрол на достъпа в нашето приложение.

Сайта е защитен от основен тип атаки (SQL инжекции, CSRF (5), XSS (6) атаки и т.н.). Чувствителната информация, като пароли на потребители не се съхранява в чист вид, а в хеширан вид. По този начин, ако злонамерен човек се сдобие с достъп до базата данни, няма да може да види паролите на потребителите.

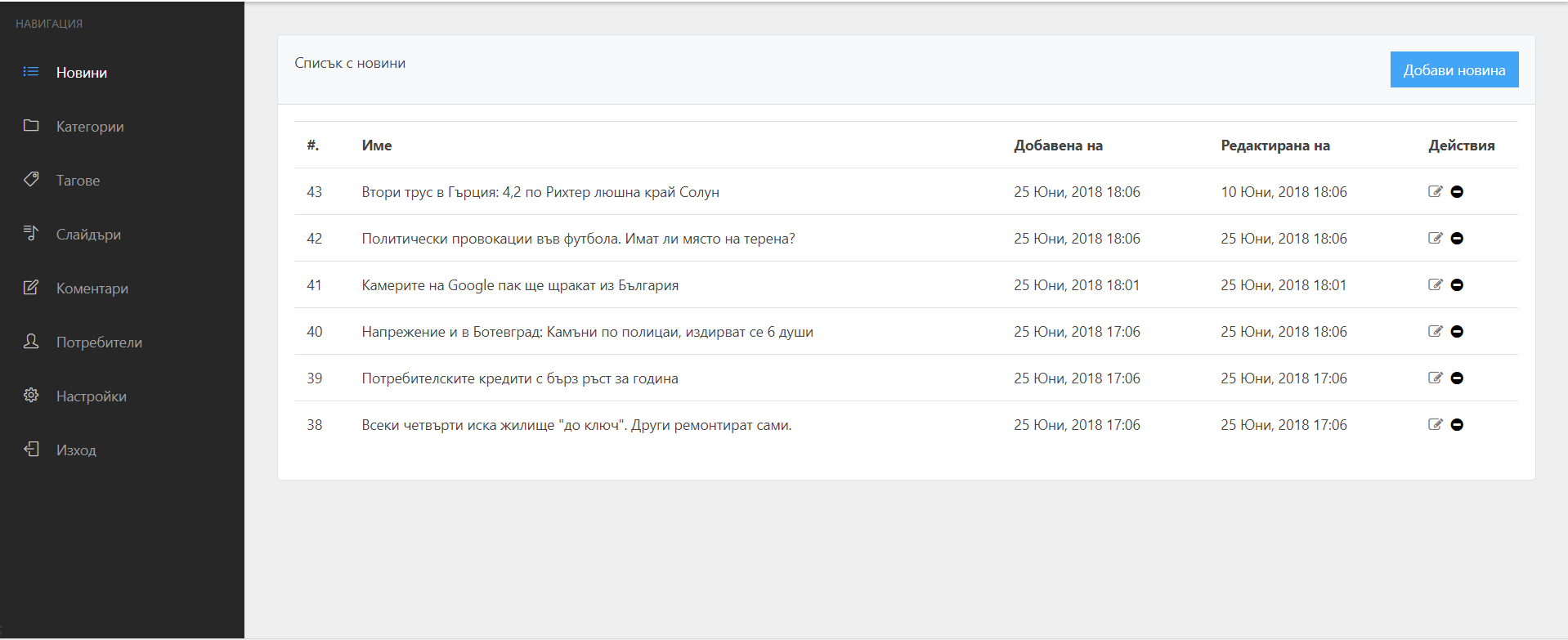
## 

## 4. Ръководство за потребителя

#### 4.1. Административна част

Потребителите, които имат необходимата роля (“Администратор”) ще имат достъп до административния панел; Там те ще могат да управляват различен вид съдържание, което ще бъде генерирано от нащия уеб сайт, като:

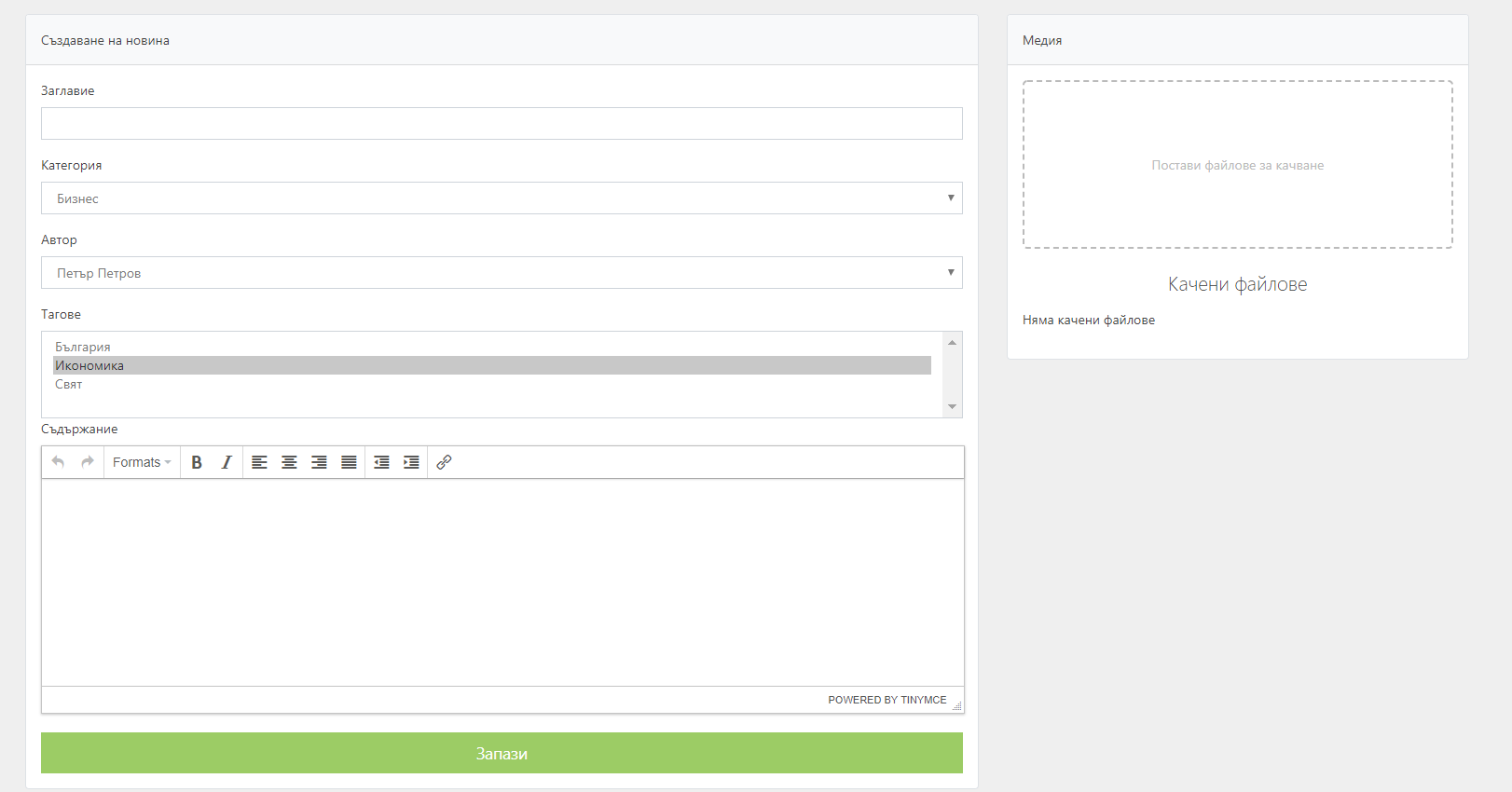
* Новини
* Категории
* Тагове
* Слайдъри
* Коментари
* Потребители



**Фигура 6.** Списък с новини. Начален екран на административния панел.

Това също така се явява и списъкът с новини, т.к. това е основния тип съдържание, който ще генерира нашата платформа. Вляво се вижда навигационно меню, което позволява бърз достъп до управление на основни еденици от данни в системата. Добра UX (2) практика е да има консистентност между изгледите, затова таблиците за показване на данни изглеждат по един и същ начин в нашия интерфейс - изброяваме няколко важни колони, които носят ценна информация и последната колона е предвидена за извършване на действия върху дадения ред. За удобство на потребителя в менюто има индикация показваща коя е текущата отворена страница.

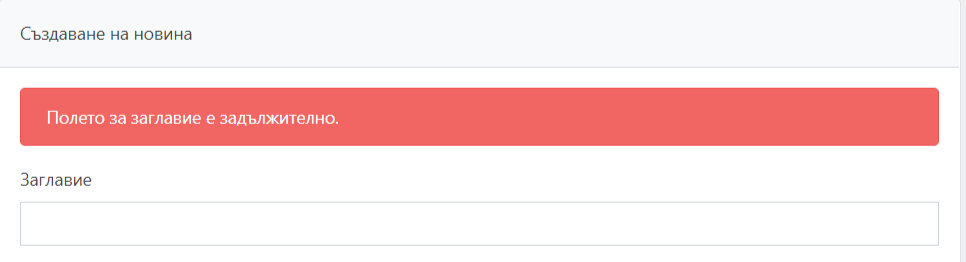
**Добавяне на новина**



**Фигура 7.** Добавяне на новина.

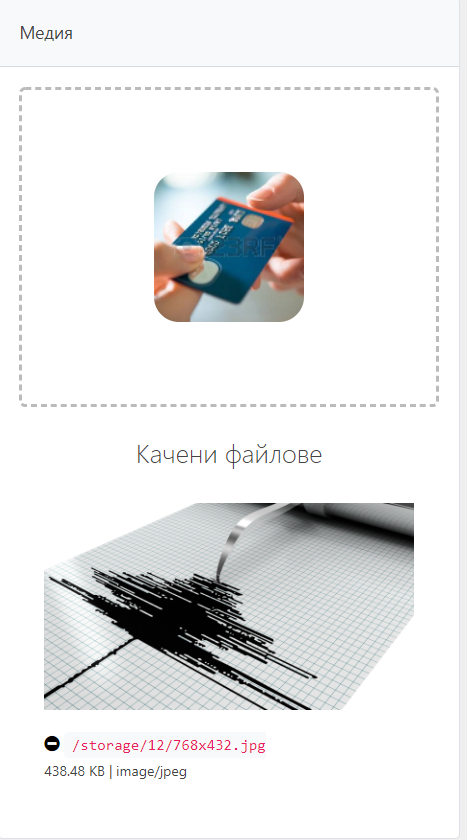
От този екран хората, които създават новина могат да изберат различни класификатори за ная (например категория или таг), също така да се упомене кой е автора на новината и да се прикачи медия към нея. Първото добаваено изображение ще излиза в потребителския интерфейс като “основно”, т.е. ще се показва във всички изгледи в които новината присъства като елемент. В случай, че се добавят множество изображения, те ще се покажат под формата на “слайдер” в изгледа, който отговаря за четене на еденична новина.

Входните данни се валидират от PHP частта, за да сме сигурни, че винаги ще съхраняваме само “чисти” и валидни данни. В случай, че потребителя се опита да въведе невалидни данни, ще види съобщение за грешка:

****

**Фигура 8.** Съобщение за грешка при въвеждане на невалидни данни.

**Добавяне на медия към новина**

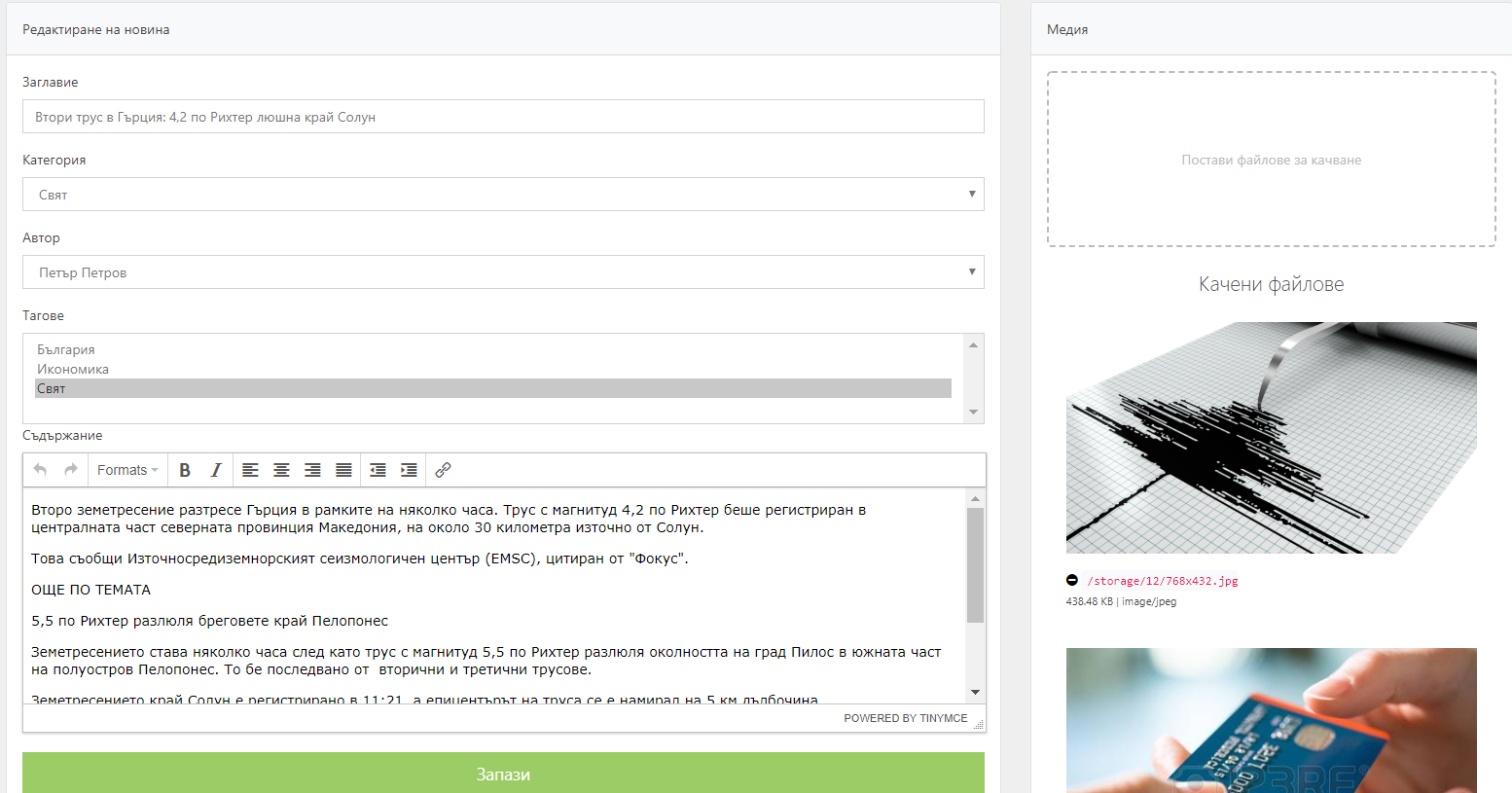
****

**Фигура 9.** Добавяне на медия към новина. Списък с файлове към новина.

При избор на файл, потребителя ще види индикатор относно прогреса на качването и допълнителен индикатор след като качването приключи. В случай на грешка ще види причината поради която файлът не е качен успешно. Файловете към дадена новина могат да бъдат лесно изтрити чрез използване на бутона за “изтриване”.

Важно е да споменем, че по време на качването, кодът създава множество различни по размери версии на изображението. Това е необходимо за да сме сигурни, че изображението ще изглежда максимално добре в различни случаи на използването му. Например, ако в изгледът с новини в дадена категория искаме да покажем снимки с размер 100 на 100 пиксела, то трябва да сме сигурни, че показаното изображение няма да изглежда непропорционално. В случаите в които размерите на изображението не позволяват преоразмеряването без загуба на част от картината, то ще бъде изрязано в краищата, като по този начин акцента ще остане върху съдържанието на снимката в нейния център.

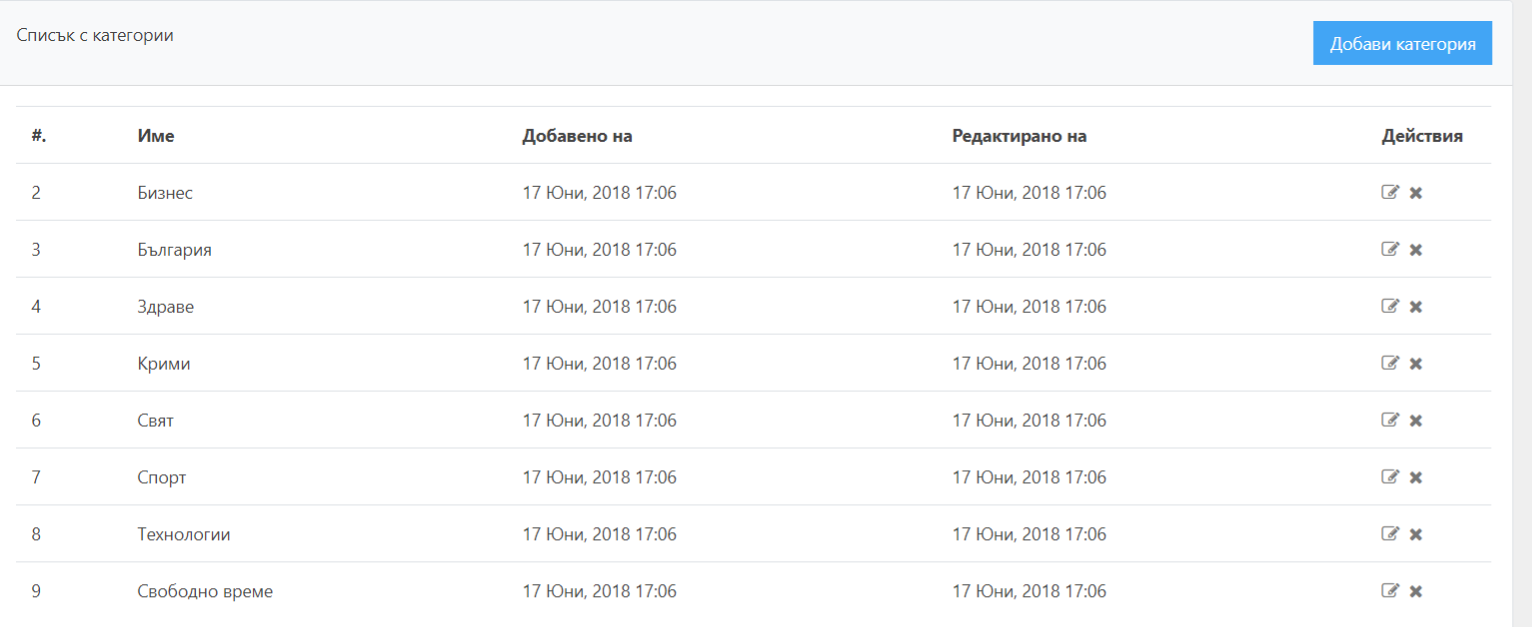
**Редактиране на новина**



**Фигура 10.** Редактиране на новина.

Изгледът за редактиране на новини е сходен с този за създаването, с тази разлика, че формата бива предварително попълнена зависимост от избраната новина. След редактиране, потребителят ще види съобщение, което му казва, че новината е редактирана успешно - или грешка, ако нещо се обърка.

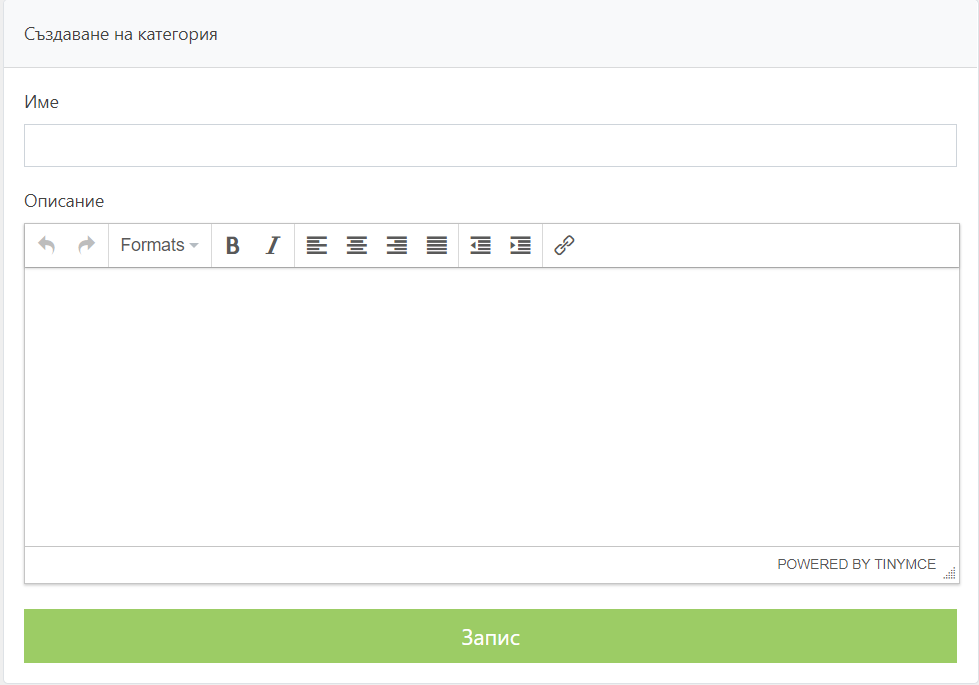
**Списък с категории**

****

**Фигура 11.** Списък с категории.

Аналогично на списъка с новини, потребителите имат възможност да видят списък с всички добавени категории. Чрез използване на бутоните вдясно всяка една категория може да бъде редактирана или изтрита.

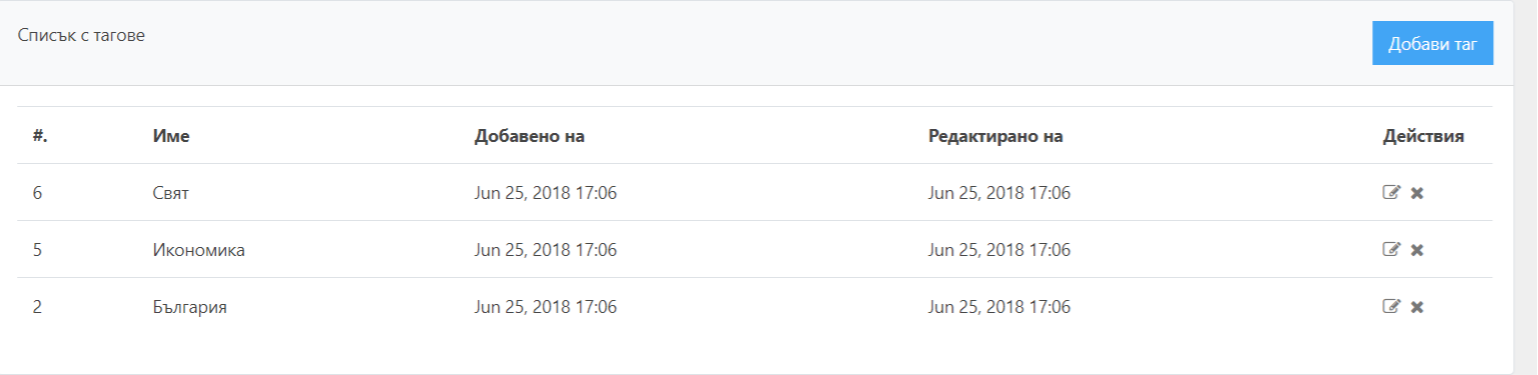
**Добавяне на категория**



**Фигура 12.** Добавяне на категория.

При добавяне на новина трябва да се попълнят име и описание на категорията. Тук също има валидация на входните данни и в случай, че потребителя въведе невалидни данни ще получи съобщение за грешка, която да отрази проблема

**Списък с тагове**

****

**Фигура 13.** Списък с тагове.

Таговете ни предоставят сравнително бърз начин за класифициране на информация, който дава бърз достъп на потребителите до нея. За разлика от категориите, които са по-генерализирани, идеята на таговете е техния брой да расте зависимост от съдържанието, което се добавя. Една новина може да бъде свързана с повече от един таг.

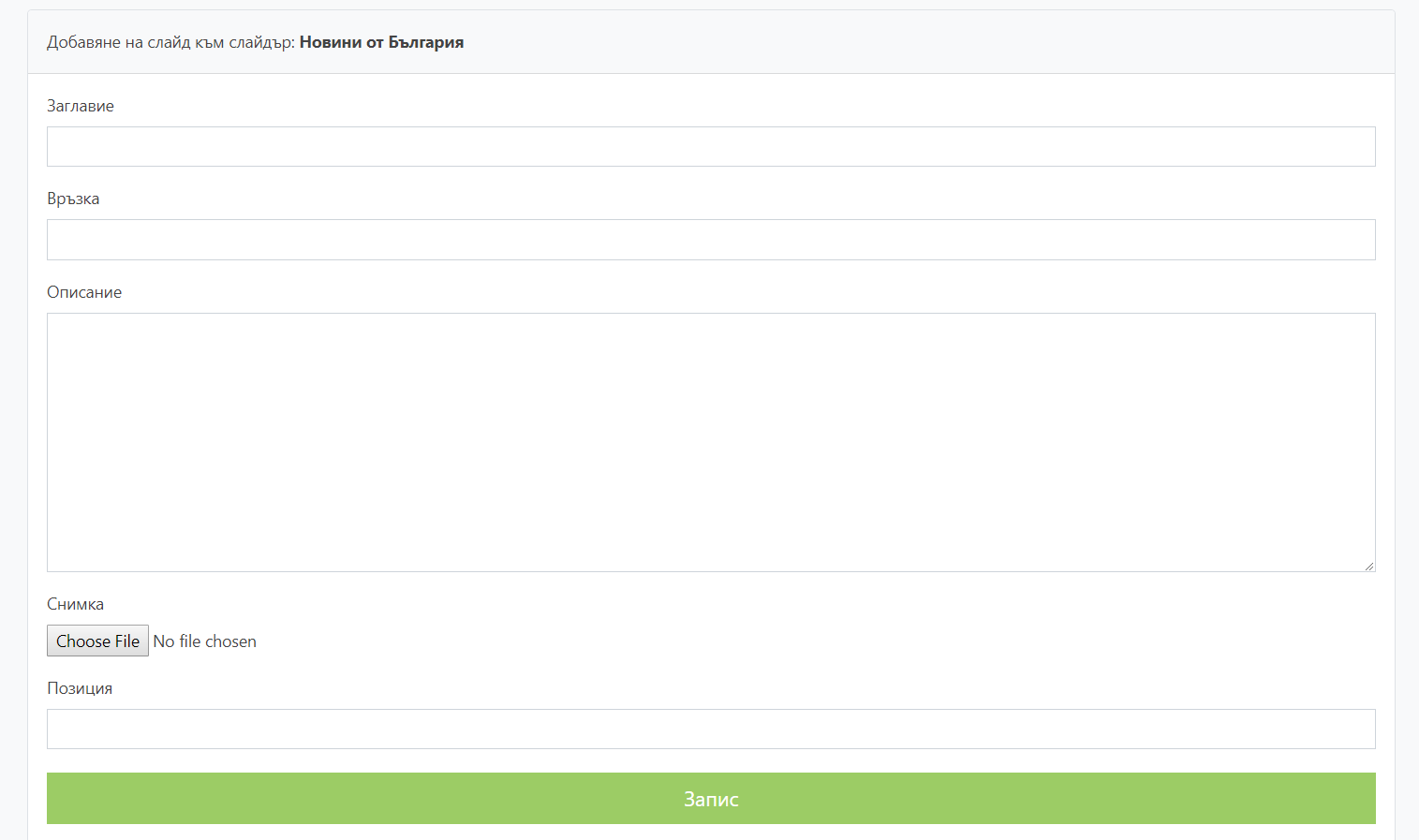
**Списък със слайдери**

****

**Фигура 14.** Списък със слайдери.

Слайдерите представляват “галерии” от изображения, като промяната им в потребителския интерфейс става чрез анимация за да привлече вниманието на потребителите върху важни акценти.

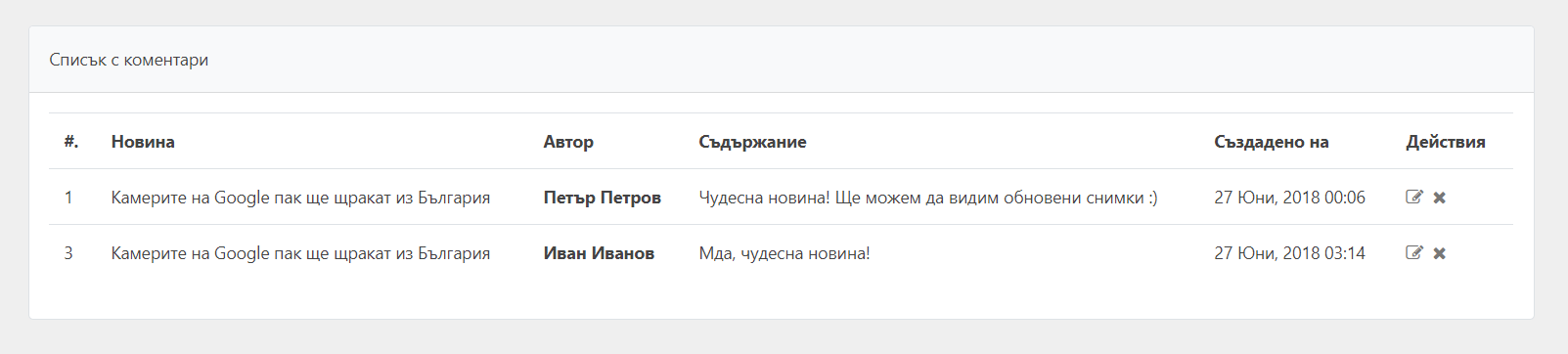
**Добавяне на слайд към слайдер**

****

**Фигура 15.** Добавяне на слайд към съществуващ слайдер.

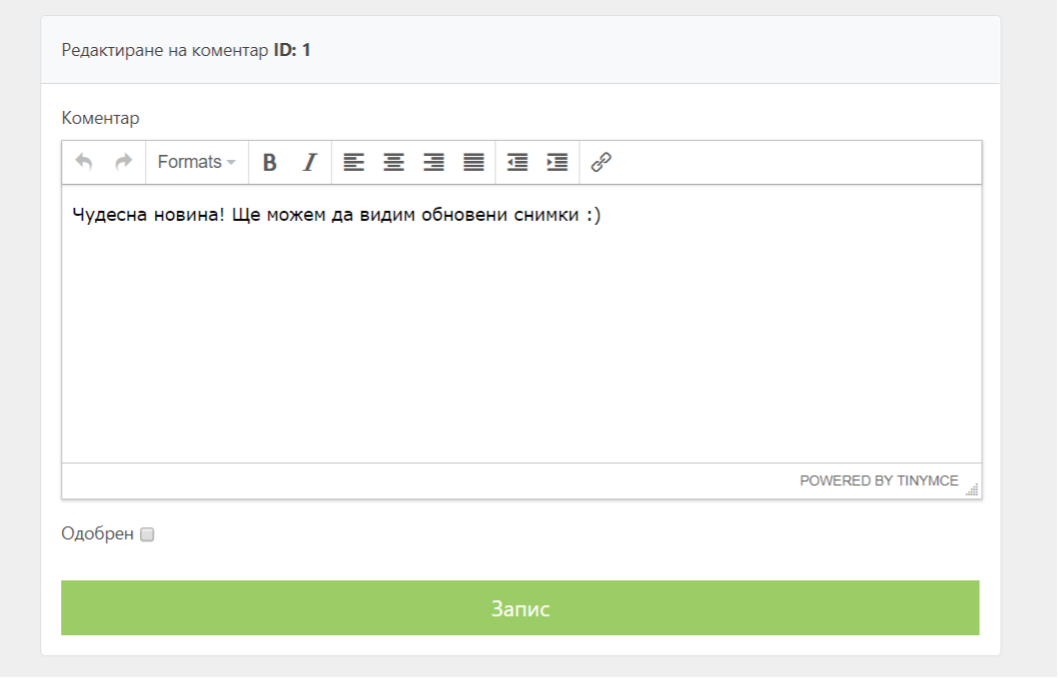
Всеки един слайд е отделно изображение в “родителският” слайдер. Описанието на слайда, както и изображението към него трябва да са достатъчно добре подбрани за да могат да привлекат вниманието на потребителите.

**Списък с коментари**

****

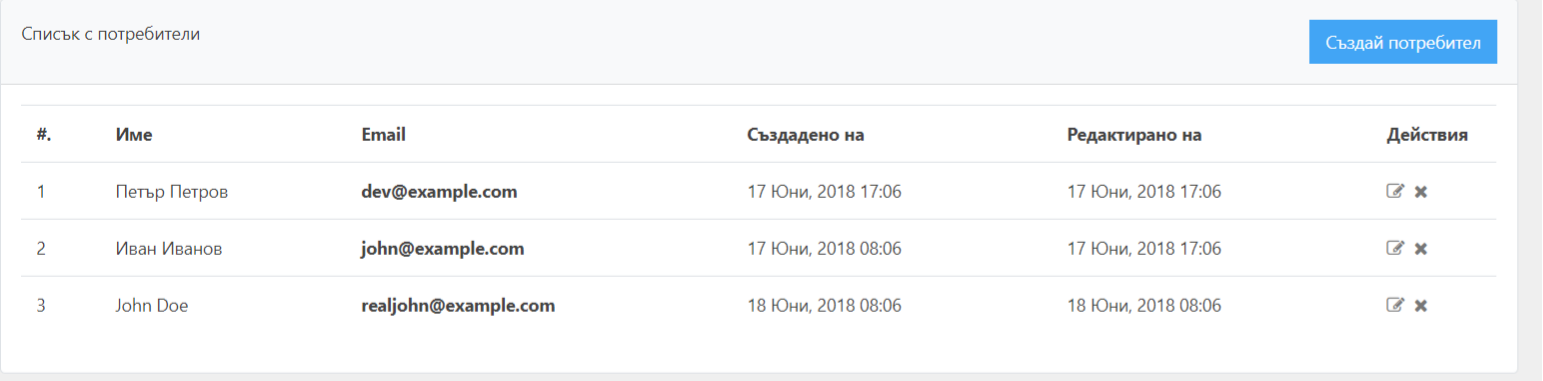
**Фигура 16.** Списък с коментари публикувани в различните новини.

**Редактиране на коментар**

****

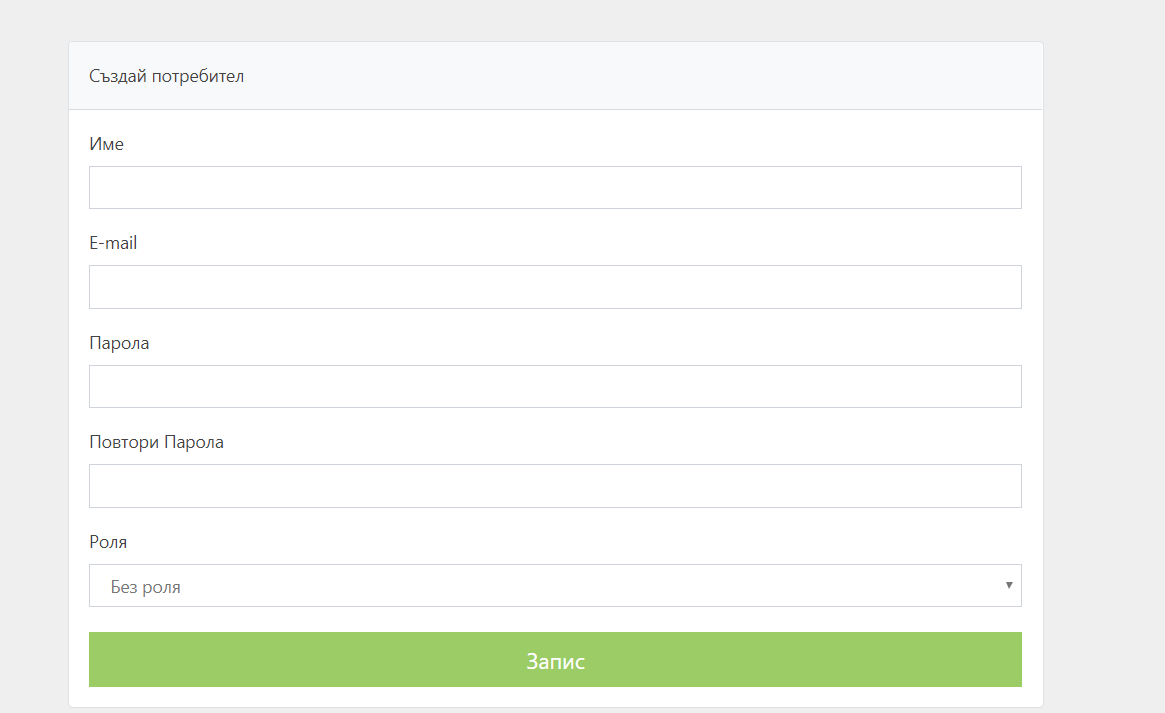
**Фигура 17.** Редактиране на потребителски коментар.

**Списък с потребители**

****

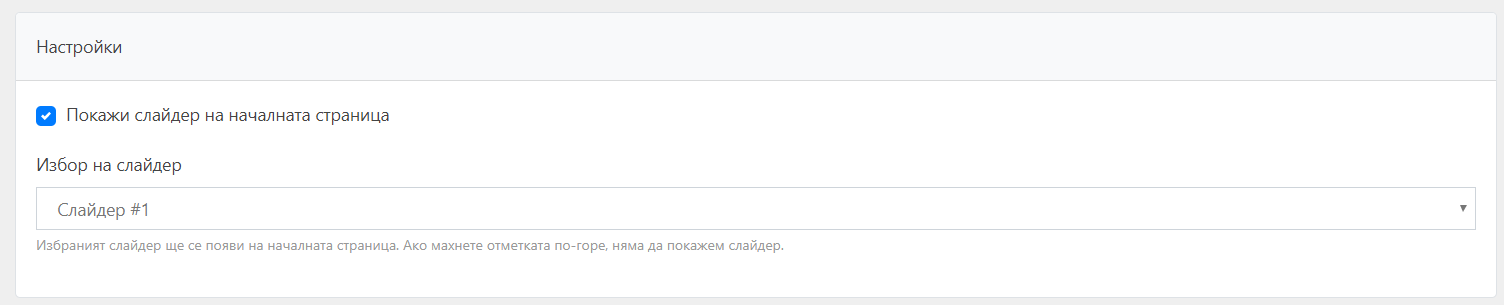
**Фигура 18.** Спиък с потребители.

**Добавяне на потребител**

****

**Фигура 19.** Добавяне на потребител.

**Настройки**

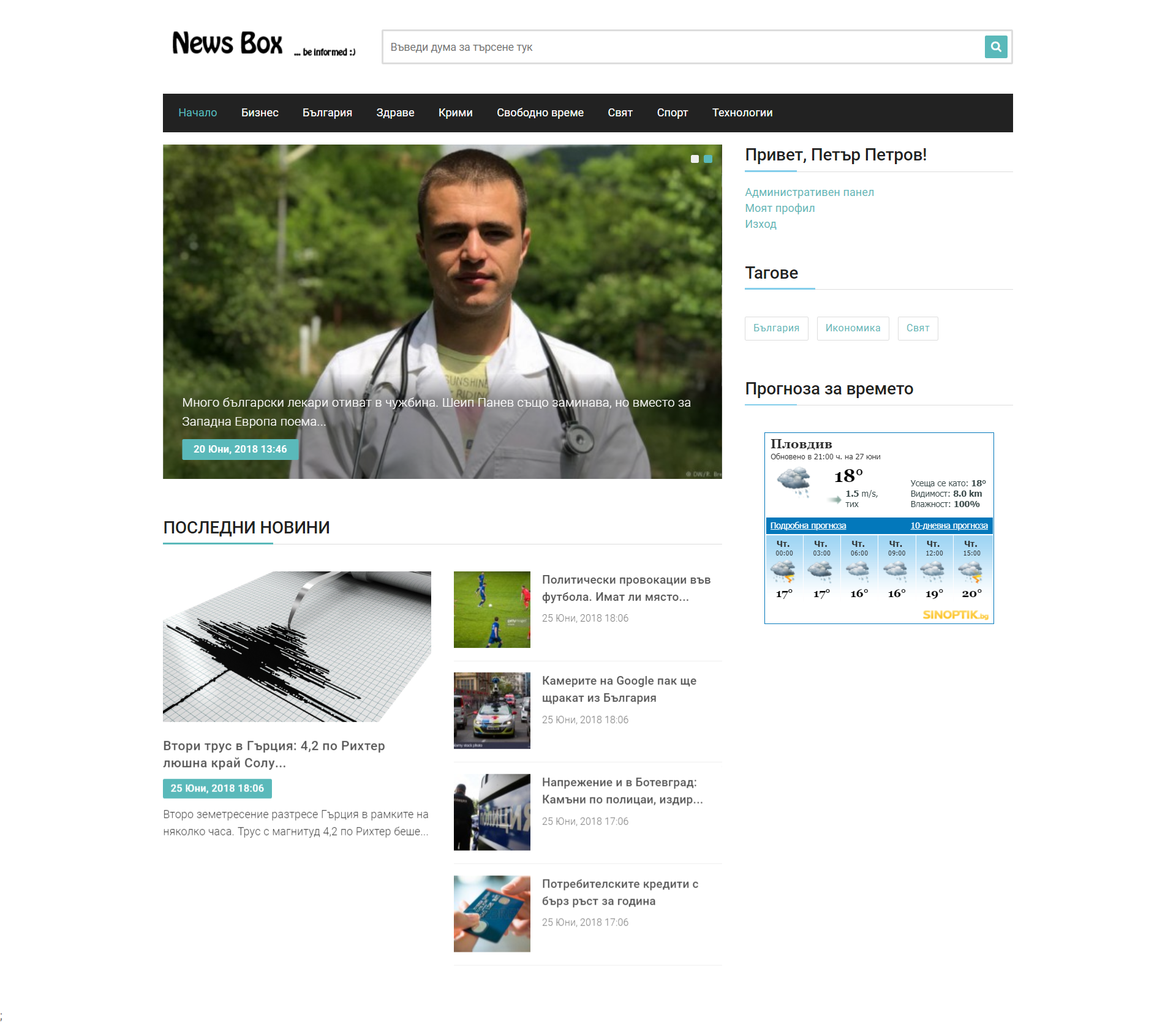
****

**Фигура 20.** Настройки на сайта.

В момента, екранът за настройки позволява само избор на слайдер, който да се покаже на началната страница. В случай, че не искаме да визуализираме такъв, трябва да махнем отметката.

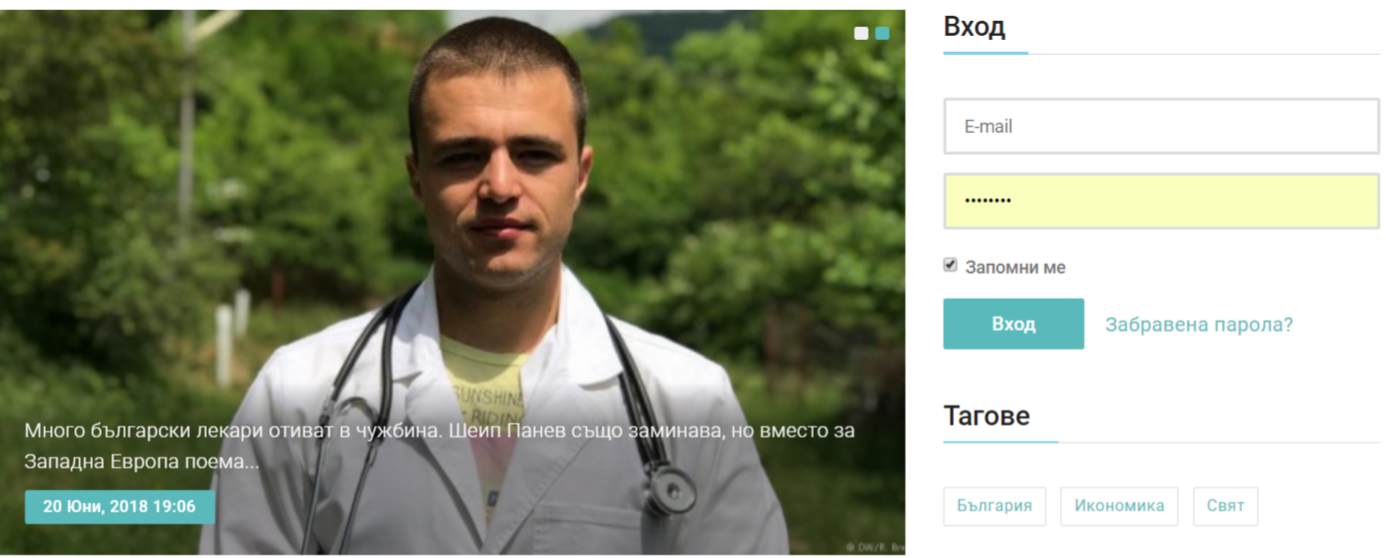
#### 4.2. Публична част

**Начална страница**



**Фигура 21.** Начална страница (за регистрирани потребители).

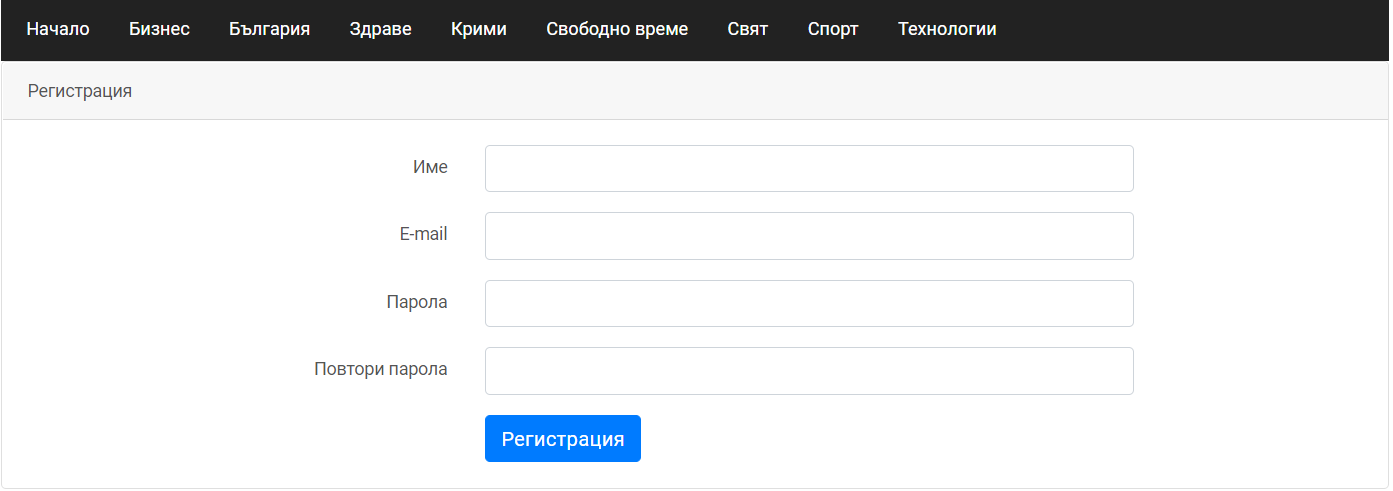
В основното меню са изброени категориите добавени във сайта. С цел привличане вниманието на потребителя, основния акцент е върху слайдъра и последните новини. Чрез полето за търсене и таговтете в дясната колона, намирането на нужната информация ще е бързо.

****

**Фигура 22.** Начална страница (за нерегистрирани потребители).

Потребителите, които не са се вписали в системата ще виждат форма за автентикация, която се намира точно над списъка с тагове.

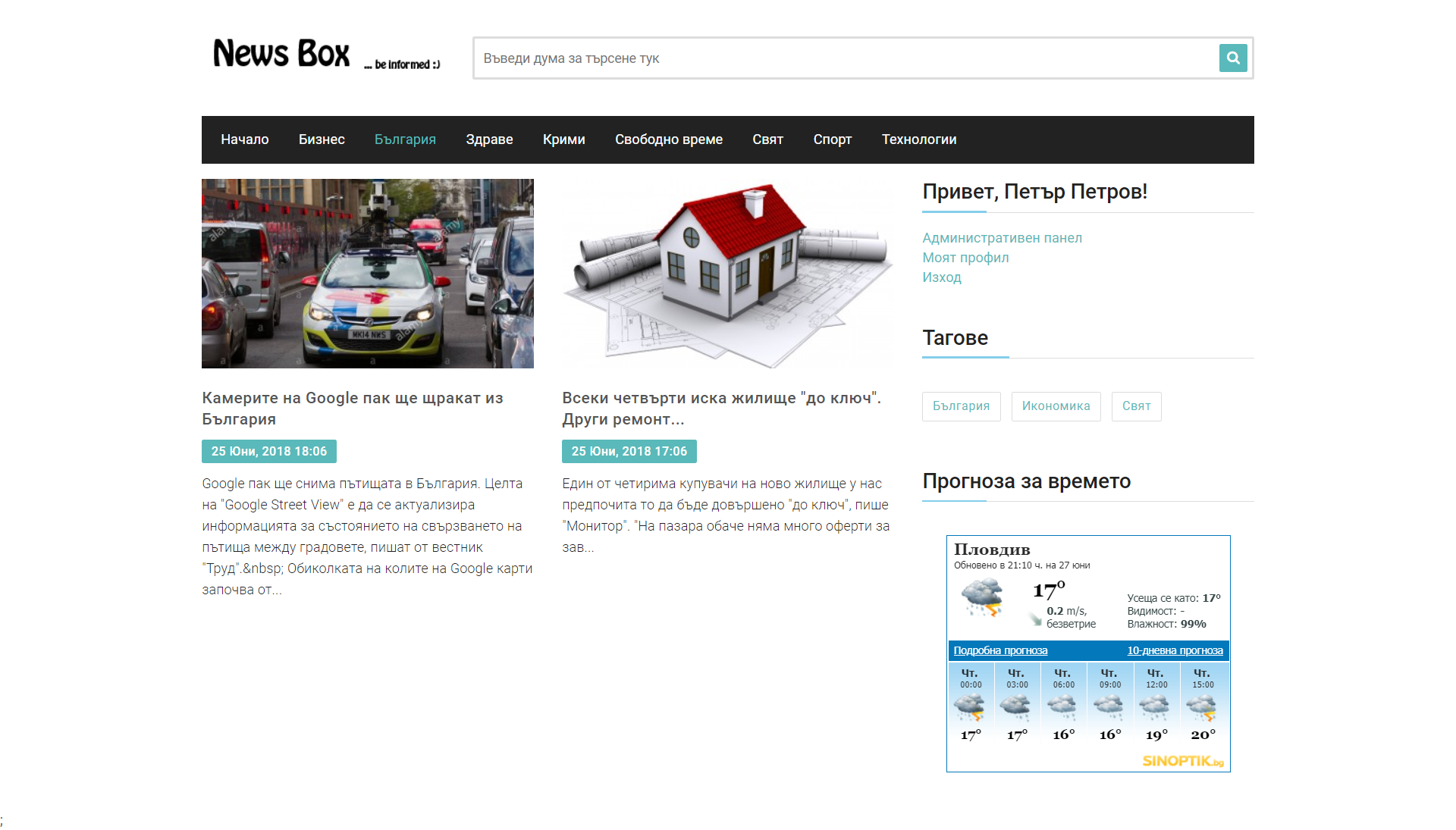
**Регистрация на потребители**



**Фигура 23.** Регистрация на потребител.

Процесът на регистрация е бърз, след въвеждане на данните потребителите ще могат да се автентикират без да е нужно потвърдение на акаунта. В случай, че се забележи злонамерена активност от даден потребителски акаунт, администраторите ще могат да му ограничат достъпа.

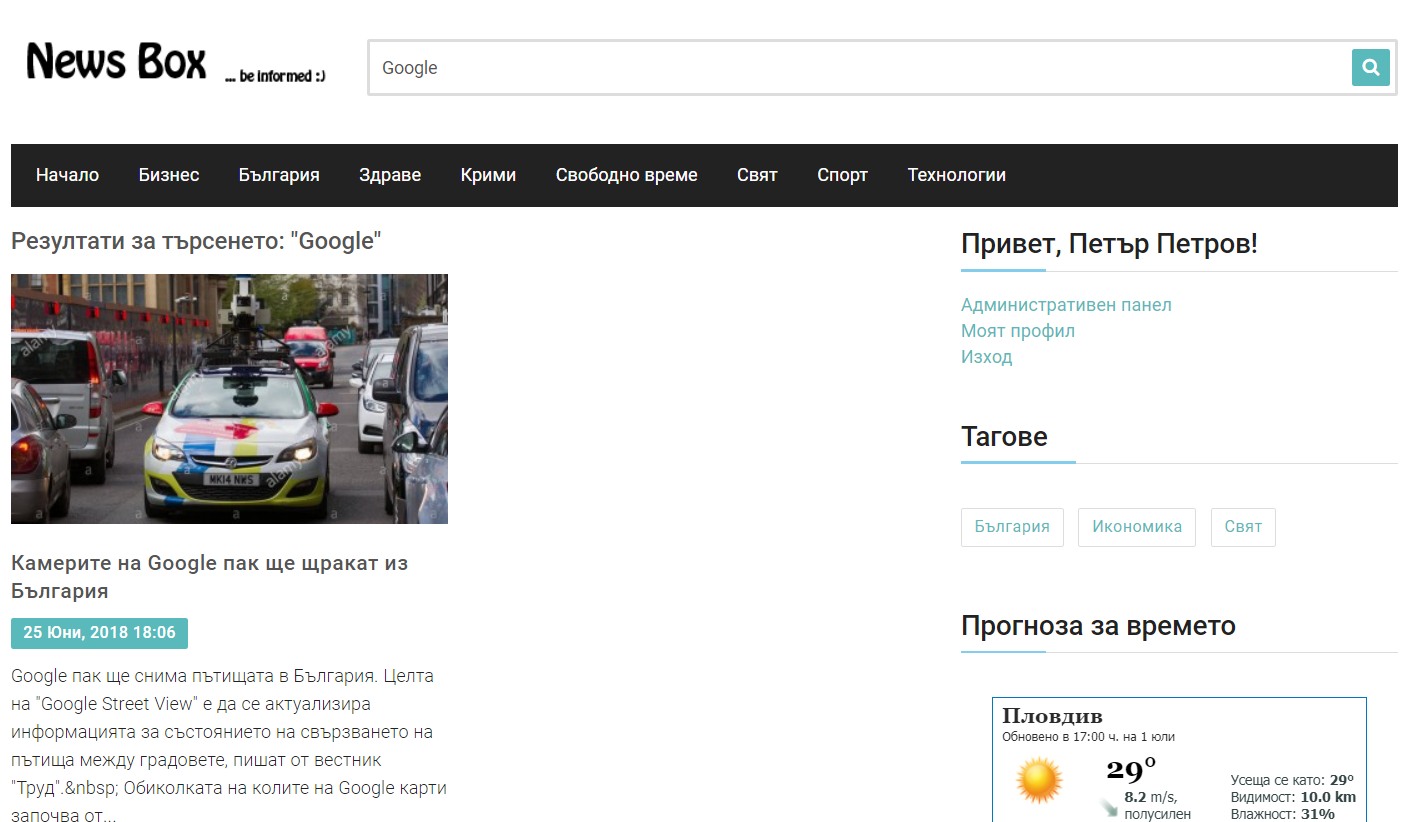
**Списък с новини в категории**



**Фигура 24.** Списък с новини в дадена категория.

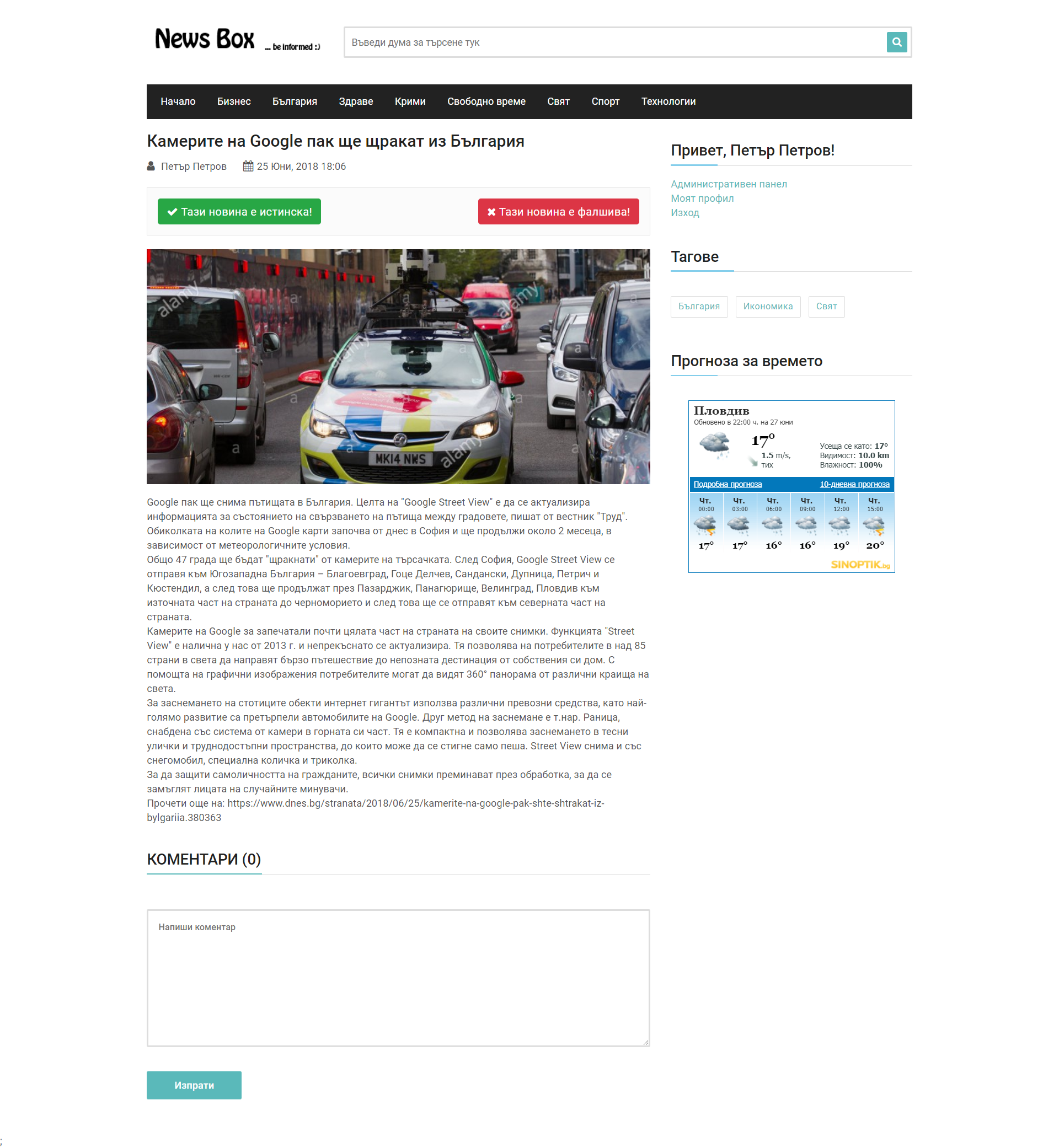
При избор на категория от менюто, същата се маркира в интерфейса за ориентация къде точно се намира потребителя в момента. Всички новини от категорията ще се покажат с кратка “презентация”, като заради възможния голям обем информация е предвидено и използването на странициране, като ще се показват по 15 новини на страница.

**Търсене на новина чрез търсачката**

****

**Фигура 25.** Търсене на новина.

**Преглед на новина**



**Фигура 26.** Преглед на новина.

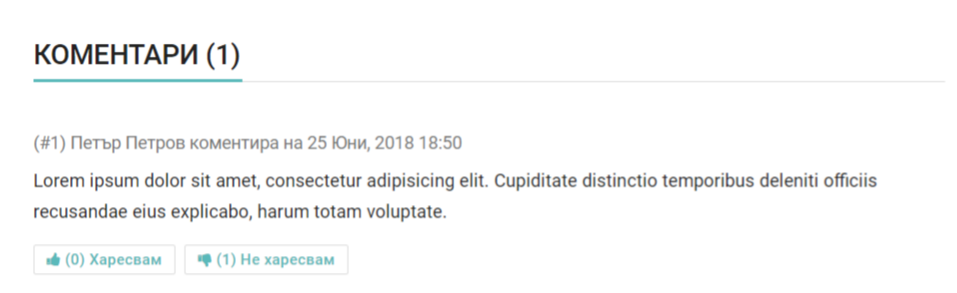
При отварне на еденична новина, в горната част можем да видим бутоните чрез които регистрираните потребители могат да споделят личното си мнение дали новината е фалшива или не. След гласуване, потребителите ще имат възможност да видят как са гласували и останалите посетители на сайта.



**Фигура 27.** Статистическа информация дали дадена новина е фалшива.

Освен логиката за “фалшиви новини”, показваме всички медийни файлове прикачени към новината, като в долната част има възможност за коментиране.

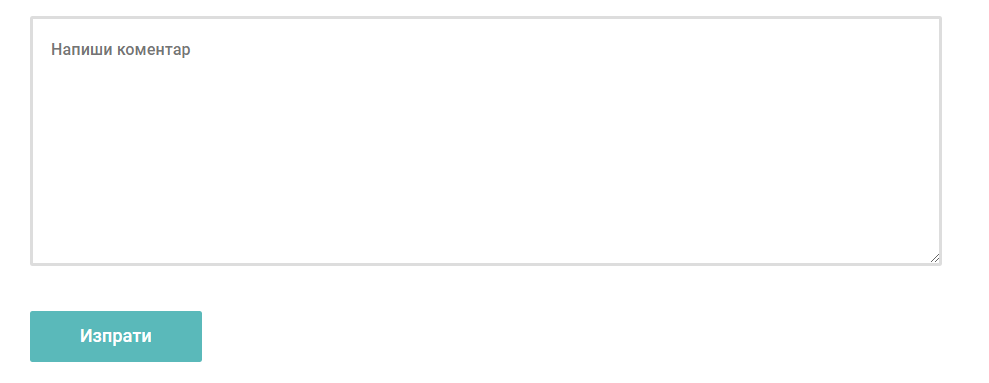
**Коментари към новина**



**Фигура 28.** Списък с коментари към новина.

На всеки един коментар може да се реагира с бутон “харесвам” или “не харесвам”. Тази информация ще е публична и всеки един посетител ще може да види как потребителите са оценили коментарите към новината. Само регистрирани потребители могат да използват тази функционалност.

**Писане на коментар към новина:**



**Фигура 29.** Писане на коментар към новина.

Коментирането на дадена новина е възможно само за регистрирани потребители. След като изпрати коментара, страницата ще бъде презаредена и той ще се появи в списъка с коментари.

## 

## 5. Заключение

#### 5.1. Обобщение на постигнатите резултати

В настоящата дипломна работа е реализиран информационен портал за новини от България и света. Проекта представлява цялостна и завършена система за публикуване на новини с акцент върху сериозния проблем, който се среща в медиите в днешни дни - фалшивите новини.

Платформата е изградена чрез използване на последните технологии в уеб програмирането, както и чрез използване на добри практики от UX дизайна, чиято идея е сайтът да бъде максимално лесен за използване от потенциалните потребители. Това ще гарантира бързо навлизане в проекта на други програмисти, които биха могли да помагат в доразвитието му.

Фалшивите новини са сериозен проблем на нашето общество в днешно време. За съжаление, все още няма разработен достатъчно добър метод за справянето с тях. Именно това е коренът в идеята чрез която се борим с фалшивите новини, колаборативния подход благодарение на който всеки един от нас може да даде своя принос.

“Ако трудът на едно човешко същество не води до благо за обществото, живота на този човек е напразен” (Джигоро Кано, създател на бойното изкуство джудо)

След успешната разработка на системата, може да се заключи, че целите на дипломната работа са постигнати.

#### 5.2. Възможности за бъдещето развитие и усъвършенстване

В зависимост от изследването на потребителското поведение и активност, бихме могли да реализираме редица функционалности, които да спомогнат за подобряването на цялостното поведение на уеб сайта:

1. Подобрена оптимизация за търсещите машини (SEO - Search Engine Optimization).
2. Реализиране на ML (Machine Learning) алгоритми, които да се самообучават и да определят дали дадена новина е фалшива или не.
3. С напредването на криптовалутите и “blockchain” технологията, бихме могли да създадем криптовалута на базата на някоя доказана, като идеята е да награждаваме потребителите за генериране на качествено съдържание.
4. След достигането на определен брой импресии бихме могли да реализираме система за управление на рекламни банери, която може да носи допълнителни приходи за сайта.
5. Разширяване на функционалностите зад потребителските профили - получаване на нотификации на база преференции.
6. Използване на WebSocket технологията с цел обновяване на различни парчета информация в реално време, без да е нужно презареждане на страницата.
7. Възможност за споделяне на новините в различни социални мрежи (Facebook, Twitter и т.н.)

## 6. Използвана литература

1. **PHP Manual.** Migrating from PHP 5.6.x to PHP 5.7.x [Онлайн] [Цитирано: 10.06.2018г.]

<http://php.net/manual/en/migration70.php>

2. **Wikipedia.** Apache. Wikipedia. [Онлайн] [Цитирано 11.06.2018г.]

<https://bg.wikipedia.org/wiki/Apache>

3. **Wikipedia.** Laravel. Wikipedia. [Онлайн] [Цитирано 11.06.2018г.]

<https://bg.wikipedia.org/wiki/Laravel>

4. **Wikipedia.** HTML. Wikipedia. [Онлайн] [Цитирано: 13.06.2018г.]

<https://bg.wikipedia.org/wiki/HTML>

5. **Douglas Crockford.** JavaScript: The Good Parts. 2008. стр. 3. ISBN: 0596517742

6. **Wikipedia.** Адаптивен дизайн. Wikipedia. [Онлайн] [Цитирано: 14.06.2018г.]

<https://bg.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0%D0%BF%D1%82%D0%B8%D0%B2%D0%B5%D0%BD_%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD>

7. **Wikipedia.** Model-View-Controller. Wikipedia [Онлайн] [Цитирано: 14.06.2018г.]

<https://bg.wikipedia.org/wiki/Model-View-Controller>

8. **Stackoverflow.** What is the difference between MyISAM and InnoDB. Stackoverflow. [Онлайн] [Цитирано: 14.06.2018г.]

<https://stackoverflow.com/questions/12614541/whats-the-difference-between-myisam-and-innodb>

8. **Laravel documentation.** Directory structure. Laravel. [Онлайн] [Цитирано: 17.06.2018г]  
<https://laravel.com/docs/5.6/structure>

## 7. Списък със съкращенията

1. CMS - Content Management System

2. UX - User Experience

3. TCP/IP - Transmission Control Protocol / Internet Protocol

4. ORM - Object-relational mapping

5. XSS - Cross-site scripting

6. CSRF - Cross-site request forgery

## 

## 8. Индекс на фигурите

Фигура 1. Принцип на работа на MVC шаблона.

Фигура 2. Принцип на работа на CSS в браузъра.

Фигура 3. Примерен JSON обект.

Фигура 4. Схема на базата данни използвана в текущата дипломна работа.

Фигура 5. Структура на файловете и директориите в Laravel.

Фигура 6. Списък с новини. Начален екран на административния панел.

Фигура 7. Добавяне на новина.

Фигура 8. Съобщение за грешка при въвеждане на невалидни данни.

Фигура 9. Добавяне на медия към новина. Списък с файлове към новина.

Фигура 10. Редактиране на новина.

Фигура 11. Списък с категории.

Фигура 12. Добавяне на категории.

Фигура 13. Списък с тагове.

Фигура 14. Списък със слайдери.

Фигура 15. Добавяне на слайд към съществуващ слайдер.

Фигура 16. Списък с коментари публикувани в различните новини.

Фигура 17. Редактиране на потребителски коментар.

Фигура 18. Списък с потребители.

Фигура 19. Добавяне на потребител.

Фигура 20. Настройки на сайта.

Фигура 21. Начална страница (регистрирани потребители).

Фигура 22. Начална страница (нерегистрирани потребители).

Фигура 23. Регистриране на потребител.

Фигура 24. Списък с новини в дадена категория.

Фигура 25. Търсене на новина.

Фигура 26. Преглед на новина.

Фигура 27. Статистическа информация дали дадена новина е фалшива.

Фигура 28. Списък с коментари към новина.

Фигура 29. Писане на коментар към новина.