

CodeIgniter 4

Техническа спецификация

изготвена от

Тодор Христо Николов - 1701681011

Емил Емилов Крумов - 1701681012

Петър Георгиев Белегански - 1701681014

Съдържание:

1. Въведение	3
1.1. Какво представлява CodeIgniter	3
1.2. Предимства	3
1.3. Недостатъци	3
1.4. Къде се използва	3
2. Инсталация и употреба	4
2.1. Инсталация	4
2.2. Разработване	4
2.3. Стартиране	5
3. Технологии и принципи	6
4. Компоненти на рамката	7
4.1. Контролери	7
4.2. Маршрутизиране	8
4.3. Филтри	9
4.4. Локализация	10
4.5. Email	11
4.6. Security	12
4.7. Honeypot	13
4.8. Session Library	13
4.9. Image Manipulation Class	14
4.10. Entity/Model	15
4.11. Pagination	17
5. Примерен проект	17
5.1. Идея на проекта	17
5.2. Технологичен стек	17
5.2.1. MySQL	18
5.2.2. CodeIgniter 4	18
5.2.3. Bootstrap 5	18
5.3. Екрани	18

1. Въведение

1.1. Какво представлява CodeIgniter

CodeIgniter е фреймуърк с отворен код(open-source), предназначен за бързо изграждане на динамични уеб приложения с програмния език PHP. Базиран е на известния MVC(Model-View-Controller) шаблон за програмиране.

Фреймуъркът е известен със своето бързодействие и с това, че е сравнително по-лек от другите алтернативни рамки за PHP.

Първата версия на CodeIgniter е публикувана през февруари 2006 г., като последната версия(CodeIgniter 4) е излезнала на 24-ти февруари 2020 г. Основните разлики спрямо предната версия са:

- CodeIgniter 4 е изграден на PHP 7 и не поддържа по-стари версии на езика
- Подновена файлова структура
- По-високо ниво на оптимизация и съответно подобро бързодействие
- Минимизиране на необходимото начално конфигуриране и настройки, за да се стартира проекта
- Вградена употреба на Entities – модели, подпомагащи изграждането на базата данни на приложението, като описват съществуващите колони в таблиците

1.2. Предимства

- Лек
- Лесен за инсталация
- Бърз
- Вградени функционалности за сигурност
- Лесен за разширяване

1.3. Недостатъци

- По-малко налични библиотеки спрямо други съвременни алтернативи

1.4. Къде се използва

- gshock.com
- propofs.com
- parchment.com
- много правителствени сайтове в САЩ

2. Инсталация и употреба

2.1. Инсталация

За да се използва CodeIgniter е необходимо да има съществуваща инсталация на PHP версия 7.2 или по-нова.

След като това условие е изпълнено, има два основни начина за инсталация и подготвяне на проект на CodeIgniter:

1. Ръчна инсталация:
 - a. Изтегля се zip-файл, който съдържа необходимите файлове за CodeIgniter проект.
 - b. Файлът се разархивира
 - c. (По избор) Създадената папка се преименува на името на проекта
2. Инсталация през Composer:
 - a. Трябва да се позволи свалянето на разширения, като се редактира конфигурационния файл `php.ini`, който се намира в инсталационната директория на PHP:
 - i. намират се редовете

```
extension=intl.dll;  
extension=mbstring.dll;
```

или

```
extension=php_intl.dll;  
extension=php_mbstring.dll;
```

и се премахва знакът ';' в края на реда

- b. Изпълнява се командата ``composer create-project codeigniter4/appstarter project-root`` като `'project-root'` трябва да бъде заменено с името на проекта

След като необходимите файлове са вече инсталирани, проектът е готов за разработване с инструментите на фреймуърка.

2.2. Разработване

Всички файлове, които се използват за разработване на приложение на рамката CodeIgniter, се намират в папката `/app`. В нея са поместени множество подпапки, предназначение да съдържат различните групи от файлове. Основната част от разработването е редактирането или добавянето на файлове в тези папки.

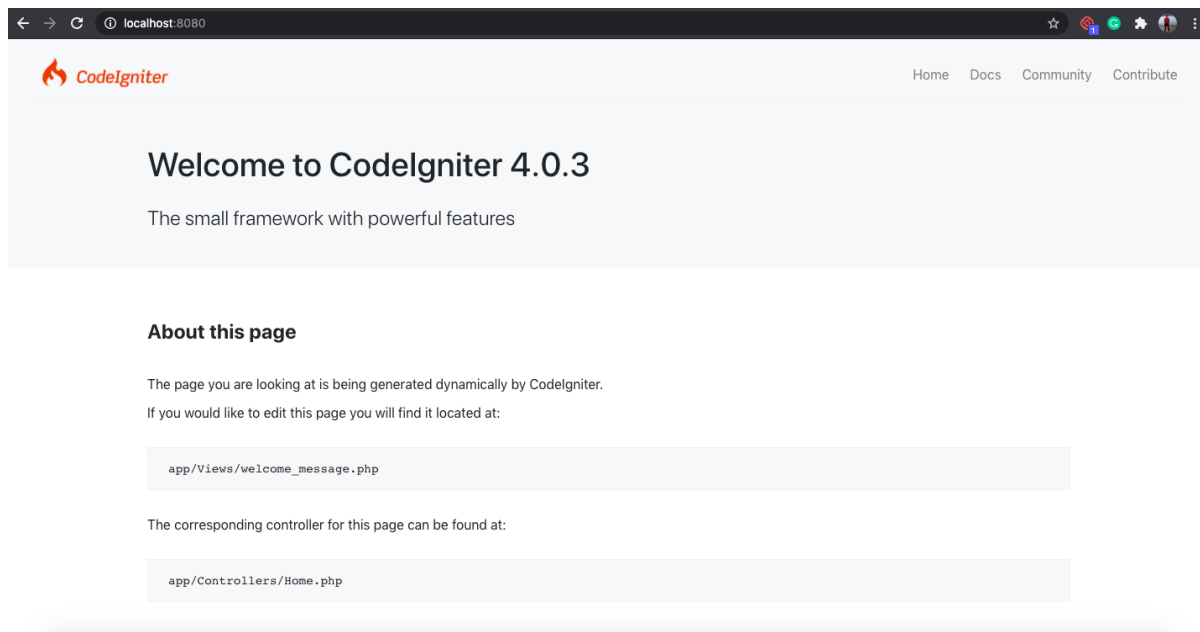
Най-важните директории и техните предназначения са:

Име на директория в <i>/app/</i>	Предназначение
Config	Съдържа файлове, отговарящи за глобалните настройки на приложението и на модулите, които то използва
Controllers	Съдържа контролерите
Database	Съдържа история на основни операции за изграждането и попълването на базата данни
Entities	Съдържа класове, които представляват дадени таблици от базата данни
Filters	Съдържа класове, които дефинират дадена логика, която се изпълнява преди и/или след извикването на метод в контролерите
Language	Съдържа файловете, отговарящи за локализацията(превеждането) на приложението
Models	Съдържа класове, които осъществяват връзката между Entity клас и дадена таблица в базата данни и предоставят възможните операции с нея
Views	Съдържа, файлове, представляващи визуалната част на приложението

Повече информация за изброените файлове и папки може да се намери в **Глава 4: Компоненти на рамката**.

2.3. Стартиране

За да бъде стартиран проектът трябва да се изпълни командата ``php spark serve`` след което да се отвори браузър и да се въведе адресът ``http://localhost:8080/``. Ако всичко е конфигурирано правилно, би трябвало да се визуализира началната страница по подразбиране:



3. Технологии и принципи

CodeIgniter е предназначен за изграждане на проекти на базата на шаблона MVC. MVC шаблонът се характеризира с разделянето на логиката на три слоя:

- **Model** – Централната част на шаблона. Представява структурата на динамичните данни в приложението. Тази част е независима от потребителския интерфейс. В този слой се намира бизнес логиката на приложението, където се извършват операциите върху данните.
- **View** – Визуалната част на приложението. В този слой се съдържа логиката и форматът на информацията, която ще бъде представена на крайния потребител.
- **Controller** – Отговаря за получаването на входни данни/заявки от потребителя, валидирането им и преобразуването им в команди към моделния или визуалния слой.

По-общо казано, когато потребителят отвори дадена страница или попълни някаква форма, се изпраща заявка към съответния контролер, който отговаря за тази страница (един контролер може да отговаря и за няколко страници), която може да съдържа входни данни. Контролерът от своя страна обработва заявката, валидира параметрите и ако всичко е наред, изготвя команда за моделът и я предава на моделния слой. Там се извършват необходимите операции спрямо командата - въвеждане, промяна, изтриване или извеждане на данни от дадена база данни и обработка на данните, ако е необходимо. След това моделът връща резултат на контролерът, на базата на който контролерът приготвя команда за визуализационния слой. Визуализационния слой приема командата от контролера, попълва данните, които е

получил, ако има такива, и приготвя крайния изглед в указан формат(най-често HTML). Обработеният изглед се връща на браузърът на потребителя, където бива визуализиран.

CodeIgniter също така може да бъде конфигуриран да използва HMVC(Hierarchical MVC), който се отличава от MVC с това, че вместо приложението да е разделено на три слоя, приложението е разделено на йерархия от модули, всеки от които съдържа трите MVC слоя.

4. Компоненти на рамката

4.1. Контролери

Контролерите като съставна част от MVC архитектурата обикновено стоят поместени в `/app/Controllers/` папката. Те се грижат за приемането на команди и входни данни от браузъра на крайния потребител и координирането на тези команди и данни с моделите и визуализацията. За лесно изграждане на такъв контролер, CodeIgniter предоставя класа *BaseController*.

Важно е да се спомене, че името на файла, съдържащ даден контролер винаги трябва да започва с главна буква и това да е единствената главна буква в цялото име.

Всеки метод в контролера е предназначен за отделен маршрут в приложението. Например, ако имаме метод `my()` в контролер *Notes.php*, то този метод ще бъде повикан при поискването на маршрут `/notes/my`. Метод, който идва наследен от *BaseController* е `index()` и той ще бъде повикан когато се поискат маршрутите `/notes` и `/notes/index`, т.е. това е метод по подразбиране.

Обекти, наследени от *BaseController* са:

- `request` - обект, съдържащ всички данни за получената заявка - метод(GET, POST, PUT, PATCH, DELETE и т.н.), параметри, хедери, данни за потребителската сесия и др.
- `response` - данни за обектът, който ще бъде върнат на браузърът като отговор на заявката - код за статус, тяло и др.
- `logger` - обект, използван за изграждане на логове
- `forceHTTPS()` - метод, който принуждава браузърът да праща само HTTPS заявки
- `validator` и `validate()` - за валидиране на данните е предоставена функцията `validate()`, с която може да се конфигурират кои полета какво изискват и каква грешка да бъде върната, ако изискванията не са спазени. Грешките се съхраняват в обекта *validator*. Пример:

```
public function updateUser(int $userID)
{
    if (! $this->validate([
        'email' => "required|is_unique[users.email,id,{ $userID}]",
        'name'  => 'required|alpha_numeric_spaces'
    ]))
    {
```

```

        return view('users/update', [
            'errors' => $this->validator->getErrors()
        ]);
    }

    // извърши някакви действия, ако входът е валиден
}

```

4.2. Маршрутизиране

Както бе описано в секцията с контролерите, маршрутизирането по подразбиране е настроено да следва шаблонът <име_на_контролер>/<име_на_метод>. Но доста често се налага да се конфигурира друг начин за дефиниране на маршрути, например, когато искаме да вземем някакъв продукт по идентификатор и трябва пътищата ни да изглеждат по следния начин:

/products/1

/products/2

/products/3

За дефинирането на маршрути в други формати, трябва да се въведат конфигурации в *app/Config/Routes.php*. За дефинирането на параметри в самите маршрути се използват заместители:

Заместител	Описание
(:any)	ще възприеме всичко оттук-нататък в маршрута като един параметър
(:segment)	ще възприеме всичко до следващата наклонена черта / като параметър
(:num)	ще възприеме цяло число като параметър
(:alpha)	ще възприеме символна поредица като параметър
(:alphanum)	ще възприеме поредица от цифри и символи като параметър
(:hash)	възприема хеширан идентификатор на даден модел като параметър

Така например дефиницията на маршрутът *products/<идентификатор>*, който трябва да извика методът *getById(id)* в контролерът *Products*, ще изглежда по следния начин:

```
$routes->add('products/(:num)', 'App\Products::getById');
```

4.3. Филтри

Филтрите са предназначени за извършване на дадени действия непосредствено преди или след изпълнението на даден метод в контролер. Примерни действия, които могат да бъдат извършени с филтри са:

- извършването на CSRF защита на пристигащите заявки
- ограничаване на зони от приложението спрямо потребителска роля
- ограничаване на броя заявки към даден маршрут
- показване на страница “Извършва се профилактика”

За създаването на филтър трябва да се създаде клас, който имплементира интерфейса *CodeIgniter\Filters\FilterInterface*. Така всеки филтър трябва да съдържа методите *before()* и *after()*, но е позволено някой от методите да остане празен, ако не е необходимо да се извършват действия преди или след дадения метод.

Примерен филтър, който проверява дали потребителят е логнат в приложението и ако не е, го препраща към маршрутът *login* изглежда така:

```
<?php

namespace App\Filters;

use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use CodeIgniter\Filters\FilterInterface;

class LoginFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $auth = service('auth');

        if (! $auth->isLoggedIn())
        {
            return redirect()->to(site_url('login'));
        }
    }

    public function after(RequestInterface $request,
```

```
ResponseInterface $response, $arguments = null) { }  
}
```

За дефиницията на филтрите за контролери и методи, трябва да се използва файлът *app/Config/Filters.php*. Добра практика е за всеки филтър да бъде дефиниран по-кратък псевдоним(alias), като може няколко филтъра да ползват един и същи псевдоним. Това се случва като се попълни полето *\$aliases*:

```
public $aliases = [  
    'login' => 'App\Filters\LoginFilter'::class,  
];
```

След това псевдонимите трябва да бъдат добавени към съответния масив, който дефинира при какви обстоятелства да се извика даден филтър. За да дефинираме филтър, който да се извиква глобално при всички маршрути например трябва да попълним масива, който се съдържа в полето *\$globals*:

```
public $globals = [  
    'before' => [  
        'login',  
    ],  
    'after' => [],  
];
```

Различните видове полета, в които могат да се добавят филтрите спрямо това кога да бъдат изпълнени са:

Поле	Описание
\$globals	филтрите се изпълняват глобално - за всеки метод във всеки контролер
\$methods	филтрите се изпълняват глобално спрямо даден HTTP метод(GET/POST/др.)
\$filters	филтрите се изпълняват спрямо дадени критерии на поискания маршрут

CodeIgniter предоставя и готовите филтри Honeypot и CSRF, служещи за защитаване от хакерски атаки, и DebugToolbar, който подпомага за откриването на грешки и дебъгване.

4.4. Локализация

Локализацията е процес, който се грижи за поддържането на различни езици в дадено приложение. За тази цел се използват файловете

/app/Language/<код_на_език>/app.php. Примерна файлова структура за приложение, поддържащо български и английски език е:

```
/app
--/Language
----/en
-----app.php
----/bg
-----app.php
```

Във файловете се дефинират различните възможни текстове във формат `ключ=>стойност`. Например грешка за невалидни данни би изглеждала по следния начин:

/en/app.php:

```
return [
    'errorInvalidData' => 'The data you provided is invalid!',
];
```

/bg/app.php:

```
return [
    'errorInvalidData' => 'Предоставените данни са невалидни!',
];
```

Езикът по подразбиране може да конфигурира във файлът `Config/App.php` по следния начин:

```
public $defaultLocale = 'en';
```

Има два начина да се дефинира как приложението да разбере на какъв език да върне съдържание:

- договаряне на локализация - дефинира се списък с допуснати езици за локализация; при всяка заявка се проверяват хедерите на заявката за поискан език и ако има съвпадение с допуснатите езици, превежда резултатът на съответния език; в противен случай използва езикът по подразбиране
- част от маршрута - може да се дефинира дадена част от маршрута чрез заместителя `{locale}`, която да служи за дефиниране на желан език

За да се изведе локализиран текст се използва методът `lang()`. Пример:

```
echo lang('errorInvalidData');
```

4.5. Email

Вграденият Email клас се грижи за лесно изпращане на имейли и поддържа следните функционалности:

- Множество протоколи: Mail, Sendmail и SMTP
- TLS и SSL криптиране за SMTP
- Множество получатели
- CC и BCCs
- HTML формат и чист текст
- Прикачени файлове
- Приоритети
- Инструменти за дебъгване на имейли

Изпращането на имейл е изключително лесно, като нужната конфигурация може да се декларира глобално в `/app/Config/Email.php` или в кода преди самото изпращане. Пример:

```
$email = \Config\Services::email();

$email->setFrom('your@example.com', 'Your Name');
$email->setTo('someone@example.com');
$email->setCC('another@another-example.com');
$email->setBCC('them@their-example.com');

$email->setSubject('Email Test');
$email->setMessage('Testing the email class.');
```

```
$email->send();
```

4.6. Security

Най-важната функционалност, която е предоставена от Security модула на CodeIgniter е CSRF филтърът. Той подsigурява заявките срещу Cross Site Request Forgery, т.е. заявки, които не пристигат от браузъра на потребителя, а от друг източник, което е потенциална хакерска атака. CSRF защитата става, като се генерира жетон за всяка форма при всяко зареждане и след изпращане на формата, жетонът се валидира. Тази валидация се изпълнява автоматично от CSRF филтъра, който трябва да се конфигурира глобално под псевдонима `'csrf'`:

```
public $globals = [
    'before' => [
        // 'honeypot',
        'csrf',
    ],
];
```

За да се вгради във формата, трябва да се добави следния HTML елемент:

```
<input type="hidden" name="<?= csrf_token() ?>" value="<?= csrf_hash() ?>" />
```

или да се добави автоматично със:

```
<?= csrf_field() ?>
```

4.7. Honeypot

Honeypot модулът е подобен на Security модулът, като целта му е с по-голяма насоченост към засичане на заявки от ботове. Защитата работи, като се добави скрито поле във формата, което не се вижда от потребителя, но се вижда от бота. Когато това поле бъде попълнено, може да се твърди, че заявката е изпратена от бот. Защитата става автоматично като се добави Honeypot филтъра под псевдонима 'honeypot':

```
public $globals = [  
    'before' => [  
        'honeypot',  
        '/'csrf',  
    ],  
    'after' => [  
        'honeypot',  
    ],  
];
```

Скритото поле се добавя от after() метода на филтъра.

4.8. Session Library

Сесията съхранява настоящите данни за потребителят, който използва системата. В PHP тя се представлява от глобалната променлива \$_SESSION. Сесията се пази, докато браузърът на съответния потребител не бъде затворен. CodeIgniter предоставя класа Session, който предлага улеснен достъп до \$_SESSION чрез няколко метода и полета.

За да се достъпи инстанция от класа Session трябва да се напише следния ред:

```
$session = \Config\Services::session($config);
```

където *\$config* не е задължителен параметър и представлява конфигурацията на приложението. Без да се подава конфигурация, може направо да се напише:

```
$session = session();
```

Елементите в сесията се съхраняват в двойки от ключ и стойност. За да вземем например стойността на ключа 'email', можем да направим следното нещо:

```
$session->get('email');
```

или да използваме “магическия” гетър:

```
$session->email;
```

или дори:

```
session('email');
```

За да добавим или променим данни в сесията, можем или да дефинираме цялото съдържание на сесията:

```
$newdata = [  
    'username' => 'johndoe',  
    'email'    => 'johndoe@some-site.com',  
    'logged_in' => TRUE  
];  
  
$session->set($newdata);
```

или да добавим стойностите ключ по ключ:

```
$session->set('some_name', 'some_value');
```

За да проверим дали даден ключ съществува използваме метода `has()`:

```
$session->has('some_name');
```

За да премахнем ключ и неговата стойност, използваме `remove()`:

```
$session->remove('some_name');
```

За да изчистим цялата сесия използваме `destroy()`:

```
$session->destroy();
```

4.9. Image Manipulation Class

CodeIgniter предоставя клас, с който удобно да се обработват изображения. Поддържат се следните операции:

- преуразмеряване
- създаване на миниатюри
- изрязване
- завъртане
- добавяне на воден знак

```
$image = \Config\Services::image();
```

Основните методи на класа са:

Име на метод	Описание
crop()	изрязване
convert()	промяна на типа на файла
fit()	побира изображението в даден размер
flatten()	заменя прозрачен фон с даден RGB цвят
flip()	обръща по хоризонталната или вертикалната ос
resize()	преоразмеряване
rotate()	завъртане
text()	добавяне на текст върху изображението

4.10. Entity + Model

Entity класовете в CodeIgniter представляват дадена таблица в базата данни. Полетата на един такъв клас, всъщност представляват полетата на един ред от съответната таблица. Методите на класа служат за изграждането на бизнес логиката свързана със съответния ред.

Нека предположим, че имаме таблица в базата данни, наречена *users*, която изглежда по следния начин:

Име на поле	Тип
id	integer
username	string
email	string
password	string
created_at	datetime

Entity класът за таблицата *users* би изглеждал така:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
```

```
// ...  
}
```

За да работим с Entity класа, трябва да направим и клас, който наследява класа Model. Той ще служи за настройването на връзката между Entity класа и таблицата в базата данни, както и за извършване на различни видове операции, свързани със съответната таблица. Примерен Model клас за User моделът би изглеждал така:

```
<?php  
  
namespace App\Models;  
  
use CodeIgniter\Model;  
  
class UserModel extends Model  
{  
    protected $table          = 'users';  
    protected $allowedFields = [  
        'username', 'email', 'password'  
    ];  
    protected $returnType     = 'App\Entities\User';  
    protected $useTimestamps = true;  
}
```

Основни методи за работа с данните на съответната таблица:

Име на метод	Описание
find()	намира ред по даден първичен ключ
findColumn()	върща всички стойности в дадена колона
findAll()	върща всички резултати
first()	върща първия резултат
insert()	въвежда един ред в таблицата
update()	обновява съществуващ ред в таблицата
save()	общ метод за <i>update()</i> и <i>insert()</i>
delete()	изтрива ред по даден първичен ключ

Важно: Методите *find()* и *findAll()* могат да бъдат използвани след метода *where()* с цел филтриране на данните. Пример:

```
$users = $userModel->where('active', 1)->findAll();
```


4.11. Pagination

Pagination модулът помага за лесно изграждане на пагинация - разбиването на резултатите на няколко страници спрямо дадена бройка резултати на страница. Модулът е автоматично зареден и може да се използва наготово.

Пагинацията се реализира чрез методът `paginate()`, наличен в `Model` класа. Данните за настоящото състояние за текущата страница се съдържа в полето `pager`, което също принадлежи на `Model` класа.

Ако например искаме да изведем всички записи от таблица `users`, в страници от по 10 реда на страница, бихме имали следното нещо:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class UserController extends Controller
{
    public function index()
    {
        $model = new \App\Models\UserModel();

        $data = [
            'users' => $model->paginate(10),
            'pager' => $model->pager,
        ];

        echo view('users/index', $data);
    }
}
```

За да се визуализира навигационно меню с опции за сменяне на настоящата страница, трябва да добавим следния код във визуализационната част:

```
<?= $pager->links() ?>
```

5. Примерен проект

5.1. Идея на проекта

Разработеният примерен проект представлява платформа за записване на бележки. Системата съдържа функционалност за създаване на акаунт, създаване, редактиране, изтриване и преглеждане на бележки.

5.2. Технологичен стек

За реализацията на приложението са използвани следните инструменти:

5.2.1. MySQL

За реализация на базата с данни е използва базата MySQL. Причината да се предпочете тази база е, че CodeIgniter има лесни инструменти за настройки с нея и заради нейната популярност и обширна документация.

5.2.2. CodeIgniter 4

Използван е фреймуъркът CodeIgniter 4 за изграждане на цялостната бизнес и визуализационна логика.

Най-използваните модули от рамката в приложението са:

- Model + Entity - за улеснен обектно-ориентиран достъп и извършване на операции с базата данни
- View - за визуализацията на страниците
- Controller - за координацията между потребителските действия, моделите и визуализацията
- Filters - за постигане на аутентикация(когато потребител, който не е влезнал в системата се опита да достъпи страница, която изисква вход, филтърът препраща потребителят към страницата за вход.
- Security + Honeypot - за защита от кибер-атаки(CSRF и ботове)
- Routing - за настройване на маршрутите в приложението

5.2.3. Bootstrap 5

За стилизация на страниците е използвана библиотеката Bootstrap 5. Тя предоставя множество класове за стилно визуализиране на хедери, линкове, бутони, текст и др.

5.3. Екрани

За да се използва системата, потребителят трябва да си създаде акаунт:

LoGo

Login Register

Sign Up

Name

Email address

Password

Confirm Password

Register



Ако потребителят вече има създаден акаунт може да използва формат за вход:

LoGo

Login Register

Sign In

Email address

t.nikolov@email.com

Password

Login

След като акаунтът е създаден и потребителят е влезнал в системата, може да види списък със своите бележки. Разполага с опциите да създаде нова бележка, да прегледа вече създадена бележка, или да излезе от профила си:

LoGo

Welcome, t.nikolov View all notes Add a note Logout

Notes

ID	Title	Content
3	Test	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i...

За да създаде нова бележка, трябва да въведе заглавие и описание на бележката и да изпрати формата. След успешно създаване на бележка, потребителят вижда детайли за

новосъздадения обект, както и опциите за редакция и изтриване:


LoGo

Welcome, t.nikolovView all notesAdd a noteLogout

Test

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Edit](#) [Delete](#)



За да редактира своя бележка, потребителят просто трябва да промени заглавието и/или описанието и да изпрати формата:

LoGo

Welcome, t.nikolovView all notesAdd a noteLogout

Edit note


Title

Test

Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Update



В списъка със всички бележки потребителят може да избере бележка, на която да види детайли:

LoGo

Welcome, Lnikolov [View all notes](#) [Add a note](#) [Logout](#)

Notes

ID	Title	Content
3	Test	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i...



Излизането от профила, ще препрати потребителят обратно към страницата за вход.