**Icon

Description automatically generatedПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА – гр. Пловдив**

**професия код 481030 „Приложен програмист“**

**специалност код 4810301 „Приложно програмиране“**

**ДИПЛОМЕН ПРОЕКТ**

**за придобиване трета степен професионална квалификация**

**Тема: Система за управление на библиотека**.

**Дипломант**: Йордан Ангелов Иванов

Клас: 12а

e-mail: dani-angel@abv.bg

**Ръководител-консултант**: Катя Семкова

Пловдив

2022

# **Съдържание**

[**Увод** 4](#_Toc92144025)

[**Основна част**](#_Toc92144026) 6

**[Глава 1](#_Toc92144027) : Цели на проекта и Използвани технологии при разработката**  [6](#_Toc92144027)

**1.1 Цели на проекта** [6](#_Toc92144028)

**1.2 Използвани технологии и програмни езици** [8](#_Toc92144028)

[**Гла****ва 2**](#_Toc92144028) **: Структура на приложението**  [15](#_Toc92144028)

**2.1 Интерфейс** [15](#_Toc92144028)

**2.2 Структура на файловете** [16](#_Toc92144028)

**2.3 Маршрутизатори** [16](#_Toc92144028)

**2.4 миграции** [17](#_Toc92144028)

**2.5 база данни** [17](#_Toc92144028)

[**Глава 3**](#_Toc92144029) **: Програмна реализация на приложението** [18](#_Toc92144029)

**3.0 Код на приложението** [19](#_Toc92144028)

**3.1 роли** [25](#_Toc92144028)

**3.2 Администратори** [25](#_Toc92144028)

**3.3 читатели** [26](#_Toc92144028)

**3.4 връзки между таблици в приложението** [27](#_Toc92144028)

[**Глава 4**](#_Toc92144030) **: Ръководство за потребителя**  [29](#_Toc92144030)

**4.0 информация за потребителя** [29](#_Toc92144028)

**4.1 табло на админ** [29](#_Toc92144028)

**4.2 редактиране на профил от тип админ** [30](#_Toc92144028)

**4.3 преглеждане на профил от тип админ** [30](#_Toc92144028)

**4.4 създаване на автор** [31](#_Toc92144028)

**4.5 преглеждане на всички автори** [32](#_Toc92144028)

**4.6 преглеждане на информация за книга** [32](#_Toc92144028)

**4.7 преглеждане на всички потребители от тип админ** [33](#_Toc92144028)

**4.8 задаване на книга към потребител** [33](#_Toc92144028)

**4.9 табло на потребител** [34](#_Toc92144028)

[**Заключение** 35](#_Toc92144031)

[**Източници** 36](#_Toc92144032)

[**Приложения**](#_Toc92144033) 37

# **Увод**

Системата за управление на библиотеката е проект, който има за цел разработването на компютъризирана система за поддържане на цялата ежедневна работа на библиотеката. Library Management System помага както на читателите, така и на библиотечния служител да следят правилно всички налични книги в библиотеката на колежа или училищната библиотека. Тя позволява както на библиотекаря, така и на ученика да търсят желаната книга. Основната характеристика на Library Management System е, че всички налични книги в библиотеката могат да бъдат показани в списък, така че читателите и администраторите да не се налага да обикалят из цялата библиотека, за да намерят книга. Освен това приложението ефективно поддържа данните за потребителите/читателите, на които са издадени книги, също така записва датата на издаване и връщане.

Системата за управление на библиотеката е софтуер, който е предназначен да управлява всички нейни функции. Той помага на администратора да поддържа базата данни с нови книги, да пази информация за заетите от читатели книги, както и сроковете за връщане.

Приложението дава възможност на всеки администратор да изпълнява задълженията си. Автоматизира следните дейности:

* Добавяне на книги;
* Записване на информация за заети книги;
* Регистриране на читатели;
* Премахване на потребители.

В проекта има два типа потребители, които имат различни роли - администратори и посетители. Всеки един администратор ще има възможност да добавя книга в системата заедно с нейните детайли. Администраторите ще имат възможност както да регистрират посетители, така и да регистрират и нови администратори.

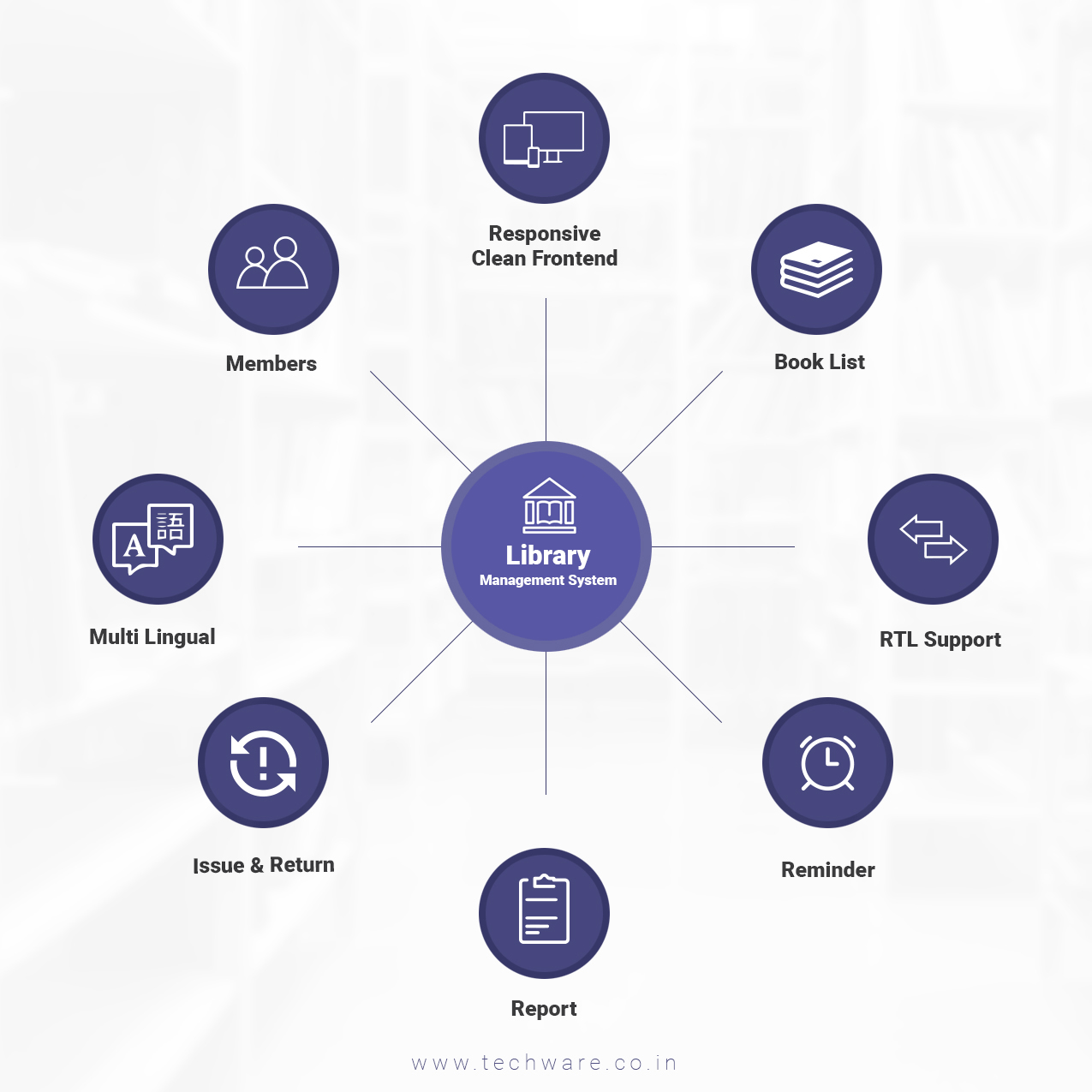
Системата поддържа CRUD (създаване, четене, промяна и изтриване на данни от база данни) операции, които дават възможност на всеки един администратор да контролира функционалността на приложението чрез натискане на бутони и въвеждане на данни в съответните таблици.

Тя е предназначена за потребители, искащи да контролират цялото съдържание и всичко, което се случва в приложението. Също така са интегрирани готови функционалности, които осигуряват удобството на всеки един потребител при използването му.

**Основна част**

## **Глава 1. Цели на проекта и Използвани технологии при разработката.**

* 1. **Цели на проекта**

**1.1.1. Ефективност на софтуера за управление на библиотека** 

**Фигура 1. Елементи на библиотечна система.**

Системата за управление на библиотеката повишава ефективността както на библиотекарите, така и на потребителите на библиотеката. Освен това дава възможност на библиотекарите лесно да каталогизират книгите и да водят правилни записи за издадените, преиздадените и на върнатите книги. От друга страна, за потребител на библиотека, те могат лесно да намерят наличността на всяка книга, от която се нуждаят.

**1.1.2. Намаляване на разходите за управление на библиотеката**

Със система за управление на библиотеката може да се намали броя на служителите или персонала и да се съхраняват различни файлове, без да се използват документи, като по този начин се намаляват и разходите. Може също така да се съхранява голям обем данни в една електронна система, така че да се сведе до минимум „ръчната“ обработка на записи.

**1.1.3. Повишаване производителността на библиотечните работници**

Автоматизацията на библиотеката ще увеличи производителността. С нарастващата лекота на работа служителите могат да се занимават с дейностите си в библиотеката, без библиотеката да поема допълнителни разходи.

**1.1.4. Подобряване на отчитането и наблюдението**

Самоактуализиращите се записи с автоматизирана система за управление на библиотеката пораждат възможности за динамично отчитане и надзор. Това ще поддържа ефективни резервации, циркулация на материали и проследяване на потребителите. Софтуерът на системата за управление на библиотеката за управление на библиотеката позволява управление на материали в системата, така че да можете лесно да знаете какво точно е налично и какво е заето.

* 1. **Използвани технологии и програмни езици при разработката.**

**1.2.1. Същност – програмни езици, библиотеки и технологии.**

Използва се PHP технология за изграждане на този Library Management System проект, както и LARAVEL Framework на PHP. Това е една от най-популярните рамки сред използваните в момента.

За реализация на операции с база данни се използва Mysql.

Технология от страна на сървъра: PHP (Laravel).

За извличане на записи от базата данни е функцията Eloquent и Query builder на Laravel.

Също така е използвана Middleware за влизане в системата.

**1.2.2.Какво е уеб разработка?**

Уеб разработката, известна още като разработка на уебсайтове, се отнася до задачите, свързани със създаването, изграждането и поддържането на уебсайтове и уеб приложения, които се изпълняват онлайн в браузър. Той обаче може да включва също уеб дизайн, уеб програмиране и управление на база данни.



**Фигура 2 Уеб разработка**

**1.2.3. Какво е PHP и Laravel?**

**PHP** е скриптов език върху сървърната (обслужваща) страна. Той е език с отворен код, който е проектиран за уеб програмиране и е широко използван за създаване на сървърни приложения и динамично уеб съдържание.

[[1]](#footnote-1)

**Laravel** е уеб фреймуърк 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

**1.2.4. Защо PHP?**



**Фигура 3 . Лого на език PHP**

Лесен е за научаване и използване. Една от основните причини PHP да стане толкова популярен е, че е сравнително лесен за започване. Дори без обширни познания или опит в уеб разработката, повечето хора биха могли да създадат уеб страница с един PHP файл за сравнително кратък период от време. Синтаксисът е прост и командните функции са лесни за научаване, което означава, че бариерите за изучаване на PHP са по-ниски, отколкото при много други езици.

Той е с отворен код (и следователно безплатен). Това също помага на разработчиците да започнат с PHP. Може да се инсталира бързо и на нулева цена. Има и отворен достъп до широк набор от PHP рамки, като Laravel и Symfony. Тази функция също е привлекателна за компаниите, тъй като помага да се контролират разходите за уеб разработка.

Той е добре свързан с бази данни. PHP улеснява сигурната връзка с почти всякакъв вид база данни.

**1.2.5. Защо Laravel?**



**Фигура 4 Лого на Laravel**

Фреймуърк Laravel се предлага с архитектурен модел на модел-view-controller (MVC). Той е лесен за използване и по този начин предлага изключително удобен начин за изграждане на големи или малки бизнес приложения. С него уеб програмистите могат да организират големи проекти с повече от пет файла за подобрена поддръжка.

Когато трябва да работите по-голям проект, трябва да се справите с много неструктуриран код. Използването на MVC може да опрости вашата кодираща структура и да ви улесни работата.

Поддръжката на MVC също улеснява намирането на файлове в техните логически директории при работа по-големи проекти. С него можете да контролирате процеса на това как вашето приложение се появява в реалния свят, така че можете да подобрите приложението за целите на проекта.

Задачите за програмиране често включват използването на механизъм за шаблони, който работи като инструмент за обработка на много текстови данни в уеб приложения. Накратко, механизмът за шаблони свързва модела на данни, обработва кода, даден в шаблоните на източника и насочва изхода към определен текстов файл или поток.

Laravel се предлага с шаблонен двигател Blade. Това е мощен, лек и предварително инсталиран механизъм за шаблони, който помага на уеб разработчиците да направят процеса на разработка гладък и лесен със своите изключителни оформления.

Модулът на шаблона Blade позволява показване на данни и разширяване на оформленията, без да се засяга производителността и скоростта на приложението. Той ви помага да създавате иновативни и невероятни оформления, като използвате функцията за засяване на съдържание.

В допълнение към това, той предоставя собствена структура за създаване и преглед на файл, който включва условни изрази и цикли. Можете също да използвате главния шаблон, за да създадете просто оформление, което може да бъде разширено с други файлове.

**1.2.6.Зaщо MYSQL?**

Уникалната архитектура на механизма за съхранение позволява на професионалистите в базата данни да конфигурират MySQL сървъра на база данни специално за конкретни приложения, като крайният резултат е невероятна производителност. Независимо дали предвиденото приложение е система за високоскоростна обработка на транзакции или уеб сайт с голям обем, който обслужва милиард заявки на ден, MySQL може да отговори на най-взискателните очаквания за производителност на всяка система.

С високоскоростни помощни програми за зареждане, отличителни кешове на паметта, пълнотекстови индекси и други механизми за повишаване на производителността, MySQL предлага всички подходящи технологии за днешните критични бизнес системи. разширени функции към всяка HTML таблица.

**[[2]](#footnote-2)**

**1.2.7.** **Регистрация и вход**

Всеки един потребител има възможност, независимо администратор или читател, да си създаде акаунт чрез регистрация. Регистрацията има следните полета:

* име на потребител,
* парола на потребител,
* потвърждаване на парола чрез въвеждане на флаг който проверява дали е чакащ или одобрен,
* тип на потребител, администратор или нормален потребител.

Регистрацията е оптимизирана и дава възможност да се създават различни типове акаунти и да се тества функционалността както от гледна точка на администратори, така и за читателите. След въвеждане на данни, следва изпращане на заявката за регистрация. Тя препраща директно в панела спрямо това дали е на администратор или потребител. Създадената валидация не позволява при грешно въведени или непопълнени данни да се завърши процеса на регистрация.

Предоставена е възможност при излизане от дадения акаунт, да се впишем отново чрез LOGIN формата.

**1.2.8. Валидация**

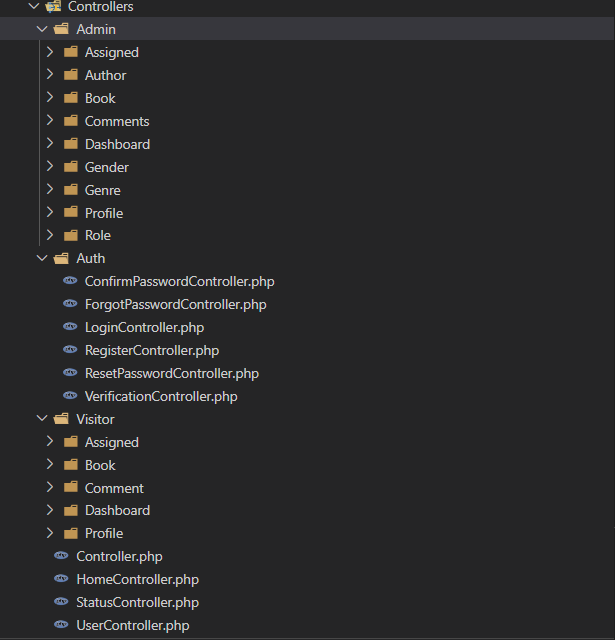
В системата за управление на библиотека всяка една форма, която създава нужните данни в приложението, има валидация. Така при въвеждане на некоректни данни потребителят няма възможност да ги записва в таблиците.

Създадени са допълнителни класове от тип APP/HTTP/REQUEST които позволяват да се валидират данните директно във Request класа вместо в контролера, което позволява да поддържаме по-чист код. Всяко едно поле има зададена валидация със специфични правила спрямо въвеждането на данни от потребител, за да може да се предпази запазването на невалидни и нефункционални записи в системата.

**1.2.9. Защита против SQL INJECTION.**

Използван е модула csrf token във всяка една форма, която ни предпазва от въвеждането на данни, които може да достъпат директно пътища, файлове и колони в базата данни и в самото приложение. Csrf token също така ни дава сигурност спрямо извличане на данни от зловреден потребител.

**1.2.10.Съхранение на функционалността.**



**Фигура 5 Съхранение на функционалността**

Всяка една част, която е свързана с функционалността на приложението, е съхранена в контролери, които управляват логиката на приложението. Всеки един контролер има за цел да предостави нужната функционалност, до така наречените “views’, които имат за цел да предоставят нужния интерфейс до крайния потребител, чрез който те ще могат да управляват функционалността чрез различни бутони и данни които, ще се визуализират във специализирани таблици “Data tables“, които визуализират нужните данни за всяка една таблица от база данни.

## **Глава 2. Структура на приложението.**

**2.1.** **Интерфейс**

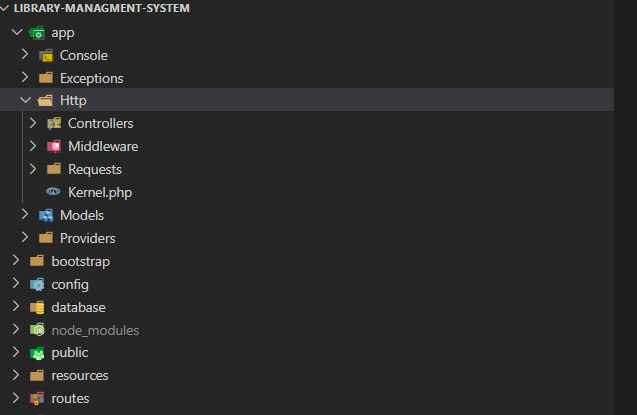
Интерфейсът на приложението е изграден с “Bootstrap 4” и “Admin LTE 3” темата, която ни предоставя готови стилизирани елементи, които може да използваме в нашето приложение, както и на библиотеката за готови функционалности на таблици „Data table“. Той е базиран на “Javascript” библиотеката за стилизиране и оптимизиране на процесите във УЕБ “Bootstrap”. “Bootstrap” има огромна колекция от класове CSS, които съответстват на обичайните ситуации при уебсайтовете. В допълнение към обичайните HTML компоненти, Bootstrap има много други компоненти, написани на CSS и JavaScript, проектирани така, че да улеснят разработчиците да създадат висококачествен потребителски интерфейс.

Компонентите включват:

* бутони;
* изображения;
* прогрес бар;
* съобщения.

DataTables е плъгин за библиотеката на jQuery Javascript. Това е изключително гъвкав инструмент, изграден върху основите на прогресивното подобрение, който добавя всички тези функционалности, които можем да използваме за оптимизиране на таблиците, както като дизайн, така и като функции за обработка на данни.

**2.2. Структура на файловете**



**Фигура 6. Файлова структура на приложението**

Приложението е базирано на фундаменталните правила за структуриране, управление и комуникиране между файлове наречено “Model View Controller”. Чрез този похват ние имаме възможността да разделим приложението си на модели- това са данните които ще има във базата за всяка една таблица, като име, години, идентификационен номер, дата на раждане и тн, изгледи- това са всички файлове във които се съдържа структурата на HTML, CSS и представянето на данни като интерфейс и контролери- това са всички файлове във които се съхраняват функционалността, записите към базата данни, функционалността за операциите, като редактиране, създаване, изтриване, и четене на данни, като четенето на данни става през контролерите и изгледите.

**2.3. Маршрутизатори**

Маршрутизацията е една от основните концепции в Laravel. Маршрутизирането в Laravel ни позволява да насочваме всичките си заявки за приложения към съответния контролер. По този начин маршрутизаторът ни позволява да правим връзка между функционалните файлове и изгледите които трябва да чрез или да изпратим заявка или да покажем данните на потребителя които е избрал или съответно които е изпратил като заявка, за да може да се запаметят във базата данни.

**2.4. Миграции**

Laravel миграциите е основна функция в Laravel, която ни позволява да създаваме таблица в нашата база данни. Позволява ни да променяме и споделяме схемата на базата данни на приложението. Може да променим таблицата, като добавим нова колона или изтрием съществуваща колона.

**2.5 База данни**

**2.5.1.Таблици**

В базата данни информацията се съхранява в следните таблици:

* Assign\_books – таблица показваща всички книги към даден потребител
* Assign\_comments – всички коментари направени от потребител към дадена книга
* Authors - всички автори
* Books – всички книги
* Books\_flags – книгите и техните флагове спрямо избора на читателя
* Genders - полове
* Genre\_authors – жанрове на автори
* Genre\_books – жанрове на книги
* Roles – жанрове на автори
* User\_flags – флагове на потребители спрямо избора на администратора и избора на флаг при регистрация
* Users – всички потребители

**2.5.2.Връзки между таблици**

Връзките обединяват отделните таблици, което повишава ефективността от използването им. Таблиците могат да се използват всяка сама за себе си, но възможностите нарастват значително, когато се използват заедно. Връзките са ключ към поддържане коректността на информацията и целостта на базата данни. Връзка между две таблици се установява като във всяка от тях се дефинират полета, които съдържат информация, обща за двете таблици. Обикновено първичният ключ в едната таблица се свързва към съответното поле в другата таблица

**2.5.3. Видове връзки**

**Съществуват три вида връзки:**

* 1:1(едно към едно)
* 1:много (едно към много)
* Много: много (много към много)

## **Глава 3. Програмна реализация на приложението**

**3.0. Код на приложението**

**3.0.1.Създаване на потребител**

Методът „store“ ни дава възможност да създадем тип потребител в системата. „Store“ методът взима дадените данни от „Post request“ във формата, която ни дава възможност да въведем нужните данни за новия потребител. Чрез тази функция ние валидираме данните, изпращаме информацията до дадения „end-point“ и ако данните са правилни, те се запазват в базата.

public function store(Request $request)

{

$user = new User();

$user->name = $request->name;

$user->email = $request->email;

$user->role\_id = $request->role\_id;

$user->flag\_id = $request->flag\_id;

$user->gender\_id = $request->gender\_id;

$user->dob = $request->dob;

if ($request->file('image')) {

$file = $request->file('image');

$filename = date('YmdHi') .

$file- >getClientOriginalName();

$file->move(public\_path('upload/images'), $filename);

$user['image'] = $filename;

}

$user->password = bcrypt($request->password);

$user->save();

return redirect()->route('user.index')->with('message', 'Successfully created data!');

}

**3.0.2.Редактиране на потребител**

Взимане на данните на текущия потребител. В този метод „edit“, ние взимаме конкретните данни чрез „Get Request“ за дадения потребител и по този начин ни препраща на страницата, която ни дава възможност да редактираме данните на потребителя. Тук взимаме всички роли, пол на потребителя и флагове, за да може да изберем правилната опция от полетата.

public function edit(User $user)

{

$genders = Gender::all();

$roles = Role::all();

$flags = UserFlags::all();

return view('admin.crud.users.edit',compact('user'),[

'genders'=>$genders,

'roles'=>$roles,

'flags'=>$flags

]);

}

*Може да откриете останалата част от кода във приложениетo:*

*Файл 1 на стр. 37*

**3.0.3.Актуализиране на потребител**

Методът „Update“, ни дава възможността да актуализираме данните за потребител като първо намерим неговото „id“, чрез query builder заявка и валидиране на данните от формата. Също така сме добавили метод който актуализира парола и след това я хешира във базата данни така, че да бъдеза защитена от зловредни намеси.

public function update(Request $request, $id)

{

$user = User::where('id', $id)->first();

$user->name = $request->name;

$user->email = $request->email;

$user->role\_id = $request->role\_id;

$user->flag\_id = $request->flag\_id;

$user->gender\_id = $request->gender\_id;

$user->dob = $request->dob;

if ($request->file('image')) {

$file = $request->file('image');

$filename = date('YmdHi') . $file->getClientOriginalName();

$file->move(public\_path('upload/images'), $filename);

$user['image'] = $filename;

}

$user->password = bcrypt($request->password);

$user->save();

return redirect()->route('user.index')->with('message', 'Successfully updated data!')

**3.0.4.Пренасочване на потребител**

Този метод проверява дадена роля на потребителя и спрямо нея пренасочва потребителите към дадената страница след регистрация или вход от страна на потребителя.

public function index()

{

if (Auth::user()['roles']['name'] == 'admin') {

return redirect()->route('dashboard.index');

}else if (Auth::user()['roles']['name'] == 'visitor') {

return redirect()->route('visitor.view.dashboard');

}

}

**3.0.5. Регистрация на потребител**

Методът „createUser“ създава потребител в базата данни след валидиране на формата и правилно въведени данни на потребителя при регистрация. Методът запазва данните, взети от „Post request“, при изпращане на формуляра от потребителя. Достъпваме User клас, който извиква статичен метод create, създаващ записите в базата.

protected function createUser(array $data){

return User::create([

'name' => $data['name'],

'email' => $data['email'],

// 'dob' => $data['dob'],

'gender\_id' => $data['gender\_id'],

'flag\_id' => 2,

'role\_id' => 1,

'password' => Hash::make($data['password']),

]);

}

**3.0.6.Регистрация на администратор**

Методът „createAdmin“ създава потребител от тип админ във базата данни след валидиране на формата и правилно въведени данни на потребителя при регистрация. Методът запазва данните във базата взети от „Post request“, при изпращане на формуляра от потребителя. Достъпваме User клас, който извиква статичен метод create, създаващ записите в базата.

protected function createAdmin(array $data){

return User::create([

'name' => $data['name'],

'email' => $data['email'],

'dob' => $data['dob'],

'gender\_id' => $data['gender\_id'],

'flag\_id' => 3,

'role\_id' => 2,

'password' => Hash::make($data['password']),

]);

}

**3.0.7. Логин на потребител**

Този метод проверява дадения флаг на потребителя и в зависимост какъв е флагът на потребителя, той го препраща към дадената страница. Използваме статични методи, които ни дават възможност бърза и сигурна автентикация

protected function authenticated()

{

if (Auth::user()->role\_id == '2' && Auth::user()->flag\_id == '3') {

return redirect()->route('dashboard.index');

} else if (Auth::user()->role\_id == '1' && Auth::user()->flag\_id == '2' || Auth::user()->flag\_id == '1' ) {

return redirect()->route('visitor.view.dashboard');

}

}

**3.1. Роли**

Във всяко едно приложение, базирано на основните принципи за разработка на уеб приложения, има няколко типа потребители, които имат различни функции спрямо това дали са администратори, потребители или нещо друго. В нашата система за управление на библиотека имаме два вида потребители и те са, администратори, които управляват по-голямата част от функциите на приложението и посетители (читатели), които имат по-ограничени възможности спрямо тези на администраторите, тъй като това са този тип потребители, които имат достъп само до тези части на приложението, които администраторът им е добавил в техния профил, в нашия случай книги.

Двата типа потребители имат също така споделена функционалност, която дава възможност да достъпват едни и същи ресурси, като например преглеждане на детайли за всяка една книга.

Приложението има добавени ограничения спрямо това дали потребителят въобще има роля или просто е без регистрация. Ако дадения потребител е без регистрация, той няма право да достъпва никакви ресурси, тъй като за всяка една роля има различни ограничения, както казахме. Всеки един нерегистриран потребител има право единствено да отвори страницата за логин или регистрация.

**3.2. Администратори**

Администраторите са най-важната част от приложението, тъй като това са тези потребители, които трябва да работят с най-много данни. От тях зависи дали даден посетител ще има възможност да преглежда своите книги и съответно да работи с тях спрямо това, което има като възможности. Всеки един администратор ще има следните възможности за редактиране, добавяне и изтриване на данни в приложението:

1. **Потребители**
2. **Автори**
3. **Полове**
4. **Жанрове**
5. **Роли на потребители**
6. **Коментари на потребители**

Всеки един администратор освен тези операции ще може да преглежда всички книги, които е добавил към един читател, всички коментари на книгите от потребители, всички коментари на потребители към книгите, одобрени потребители, чакащи потребители, всички администратори в системата, всички посетители, независимо одобрени или не, дали даден потребител е на линия или не, преди колко време е бил на линия и колко книги има в неговия профил, дати на създаване и актуализиране, независимо към коя таблица от базата правим записи. Администраторът ще може да променя парола на всеки един потребител и естествено да решава дали е одобрен или не. Системата предлага допълнителни функционалности на администратора, като преглеждане на автори и потребители, данни за тях, преглеждане на автори спрямо към коя книга са присъединени, жанрове на автори, преглеждане всички автори спрямо жанр в който пишат. Данните за отделна книга – от какъв жанр е тя, кой е авторът и. Той също така може да редактира собствения си профил.

**3.3.** **Читатели**

Читателите са с по-малко функционалности, тъй като те зависят от това дали администраторът на системата е одобрил тяхната регистрация и дали ще има добавени книги в техния профил. Но все пак читателите имат разнообразни функции като добавяне и редактиране на коментари към дадена книга, промяна на статуса на книга спрямо това, дали даденият читател иска неговите книги да бъдат в таблиците, одобрени, предстоящи, архивирани, или прочетени. Ако потребителят не е одобрен няма да има нито една функционалност, единствено ще може да стои статично в профила си със съобщение, че неговата заявка се очаква да бъде одобрена от администратора. При одобрение, ще има възможността да използва системата, както тези функции, които изброихме, също така да променя данните на профила си, профилната снимка, да вижда броя на коментарите, на книгите, броя на книгите които са в отдел прочетени, предстоящи, архивирани или любими. Той няма да има удобството да изтрива книгите от профила си, а просто да мести в отделни таблици, докато администраторът не го направи.

**3.4.Връзки между таблици в приложението:**

* Всяка една **книга** може да има автор(author \_id), жанр(genre \_id), потребител(user \_id)

**name-> varchar (255)**

**content-> LONGTEXT**

**year-> DATE**

**book\_no-> int**

**author \_id-> Table(authors)->id(int);**

**genre \_id-> Table(genre \_books)->id(int);**

**user \_id-> Table(users)->id(int);**

* Всеки един **автор** може да има жанр на литература(genre \_id), пол(gender \_id)

**genre \_id-> Table(genre \_authors)->id(int);**

**gender \_id-> Table(genders)->id(int);**

* Всеки един **потребител** може да има роля(role \_id), пол(gender \_id)

**role \_id -> Table (roles)->id(int)**

**gender \_id-> Table (genders) ->id(int)**

* Всяка една **роля** може да има (id), име(name)

**name->varchar(255)**

* Всеки един **пол** има (id), име(name)

**name->varchar(255)**

* Всеки един **жанр на автор** може да има (id), име(name)

**name->varchar(255)**

* Всеки един **жанр на книга** може да има (id), име(name)

**name->varchar(255)**

* Всяка една **възложена книга** може да има (user\_id), (book\_id), (date\_of\_receipt), (date\_of\_return)

**user\_id-> Table(users)->id(int)**

**book\_id-> Table(books)->id(int)**

**date\_of\_receipt-> DATE**

**date\_of\_return-> DATE**

* Всяка един **коментар** може да има (user\_id), (book\_id), (name)

**user\_id-> Table(users)->id(int)**

**book\_id-> Table(books)->id(int)**

**name -> varchar(45)**

* Всяка един **флаг на потребител** може да има (user\_id), (name)

**user\_id-> Table(users)->id(int)**

**name -> varchar(45)**

**Глава 4. Ръководство за потребителя**

**4.0. Информация за потребителя**

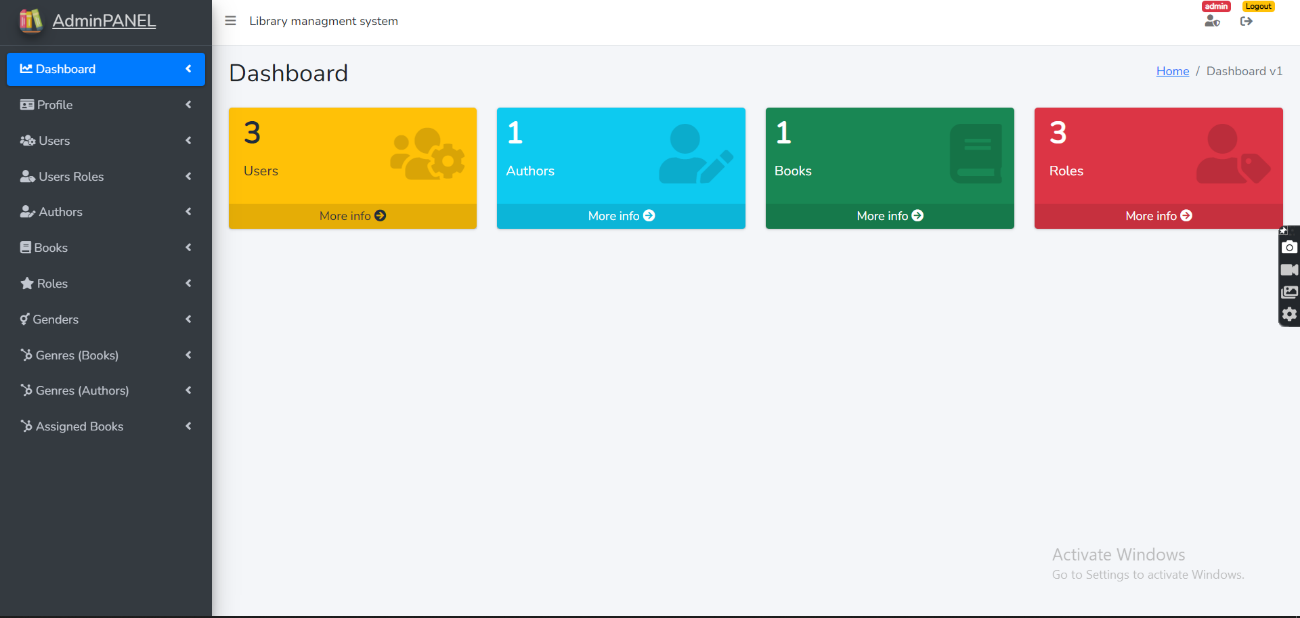
В системата има две роли, администратор и потребител. В зависимост каква роля има потребителя, има различни функционалности. Ролята определя типа на потребителя, а типа на потребителя определя възможностите, които има даден потребител. Администраторът има пълен достъп до системата като има възможност да работи със данните както на потребителите, така и с всяка една функционалност отнасяща се до системата, като добавяне на книги, автори, жанрове и т.н.

Администраторът има възможност да сменя паролите на потребители, да изтрива книги към даден потребител, които са едни от най-важните функционалности, отнасящи се до администраторската част.

Обикновеният потребител има по-ограничени възможности, тъй като той не трябва да работи с цялата функционалност на приложението, а само с тази част, която дава възможност да избира дадената книга какъв тип да бъде, да редактира профила си и да преглежда книгите, които има.

**4.1. Табло на админ**

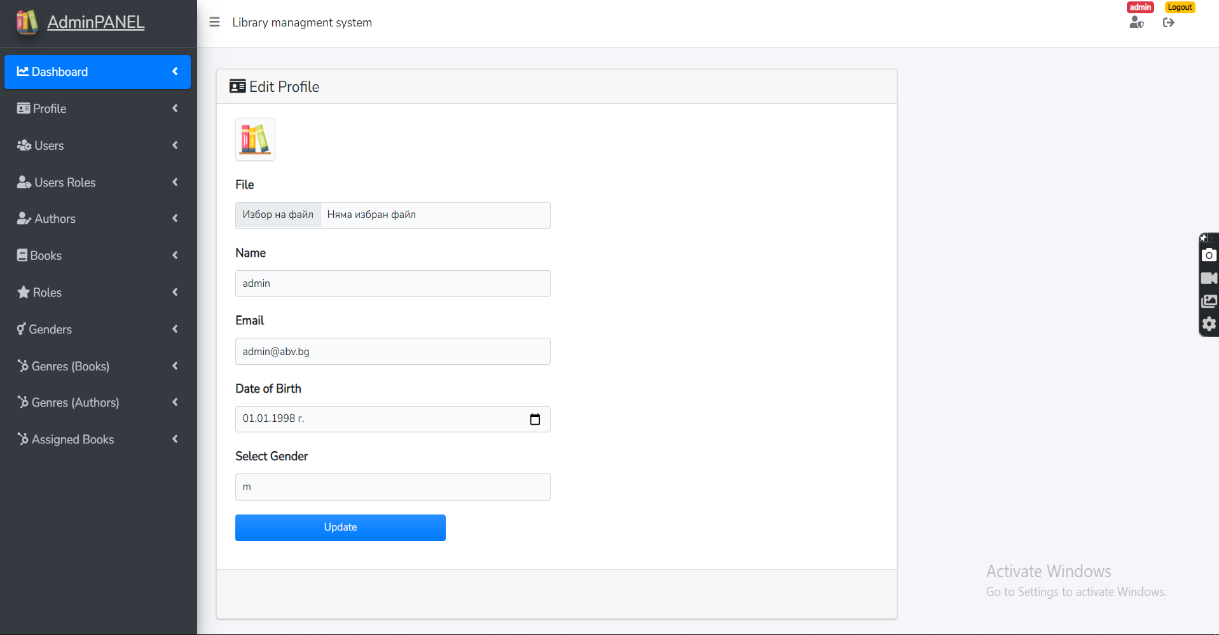
* Потребители
* Автори
* Книги
* Роли



**Фигура 7. Табло на админ**

**4.2. Редактиране на профил от тип админ.**

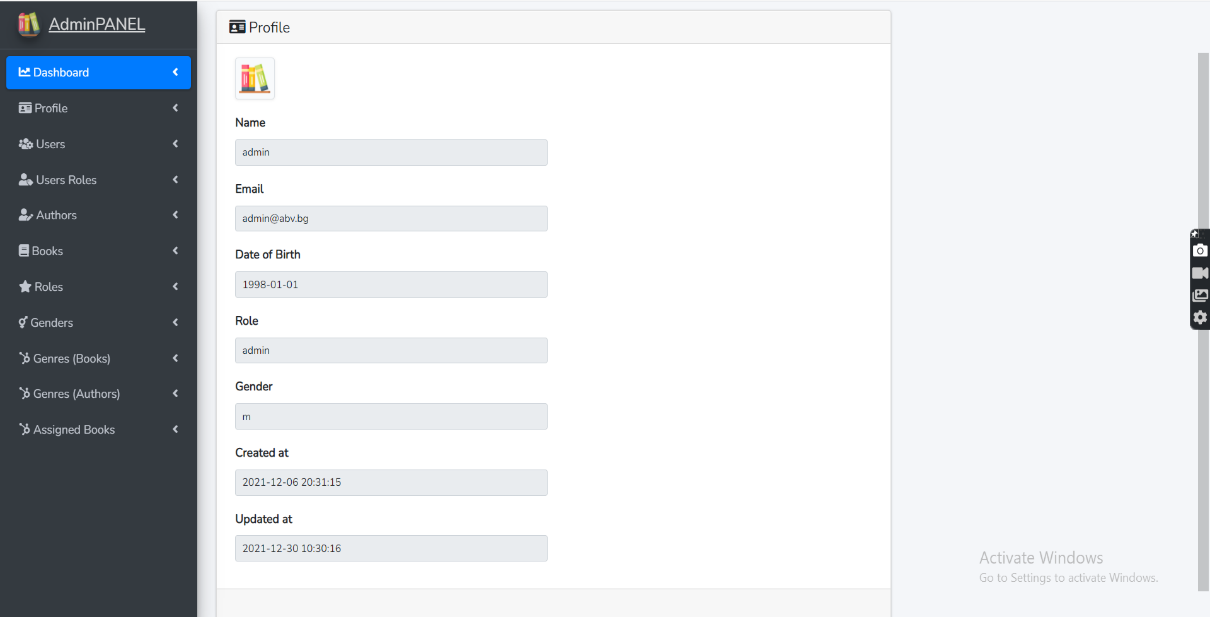
* Редактиране на снимка
* Редактиране на име
* Редактиране на имейл
* Редактиране на рождена дата
* Редактиране на пол



**Фигура 8. Редактиране на профил от тип админ.**

**4.3. Преглеждане на профил от тип админ:**

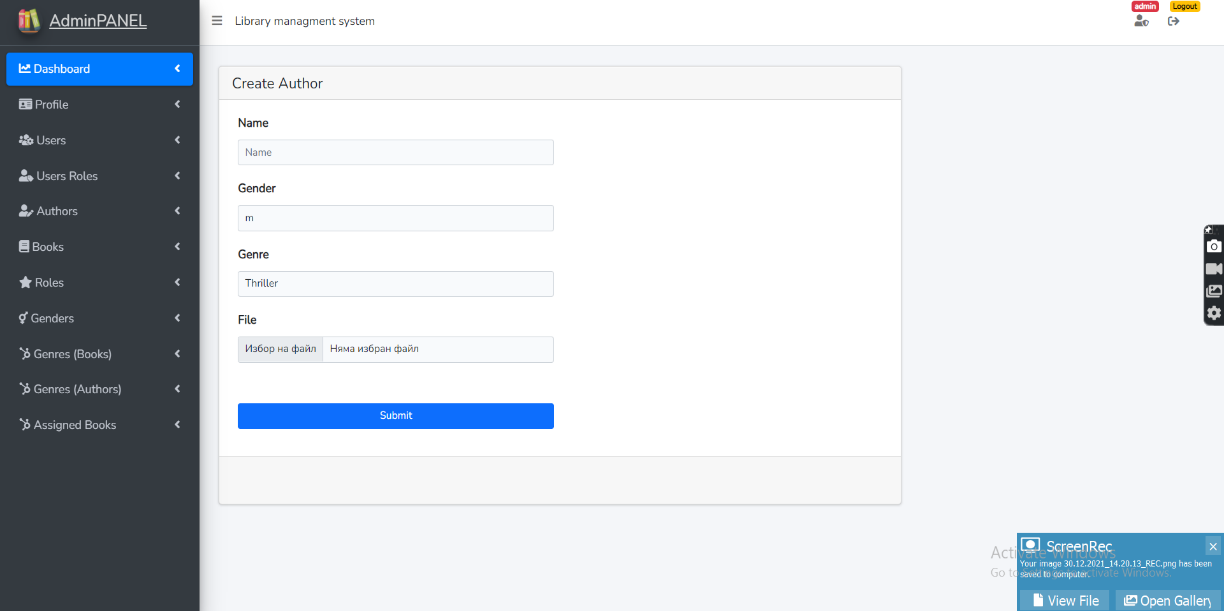
* Име
* Имейл
* Дата на раждане
* Роля
* Пол
* Дата на създаване на потребителя
* Дата на редактиране на потребителя



**Фигура 9. Преглеждане на профил от тип админ.**

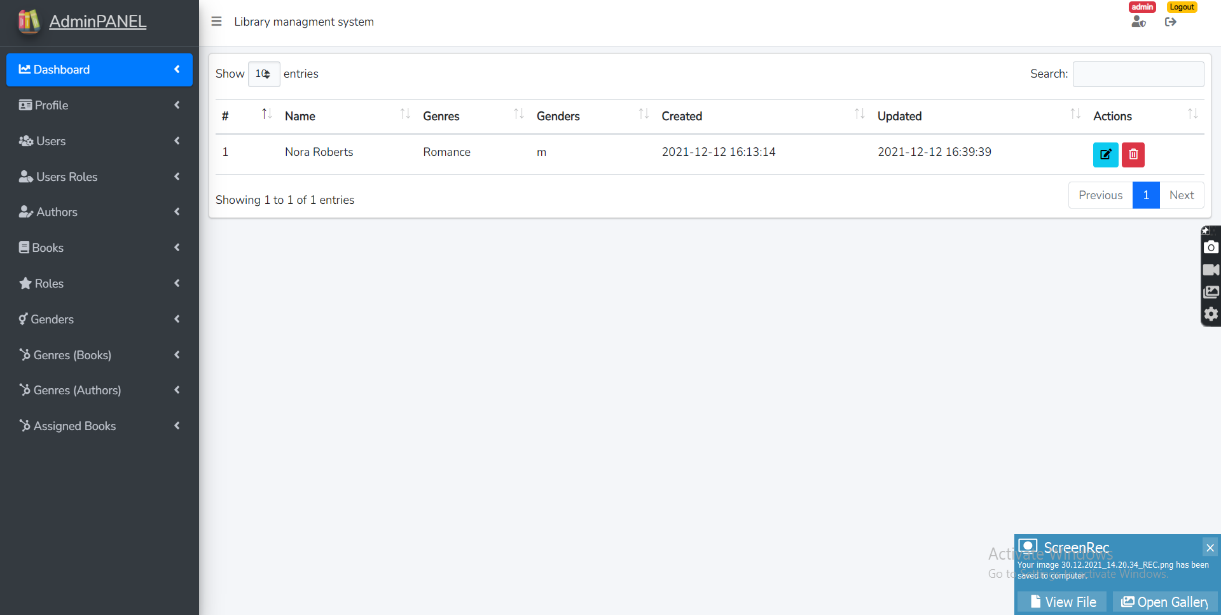
**4.4. Създаване на автор:**

* Име
* Пол
* Жанр на автор



**Фигура 10. Създаване на автор.**

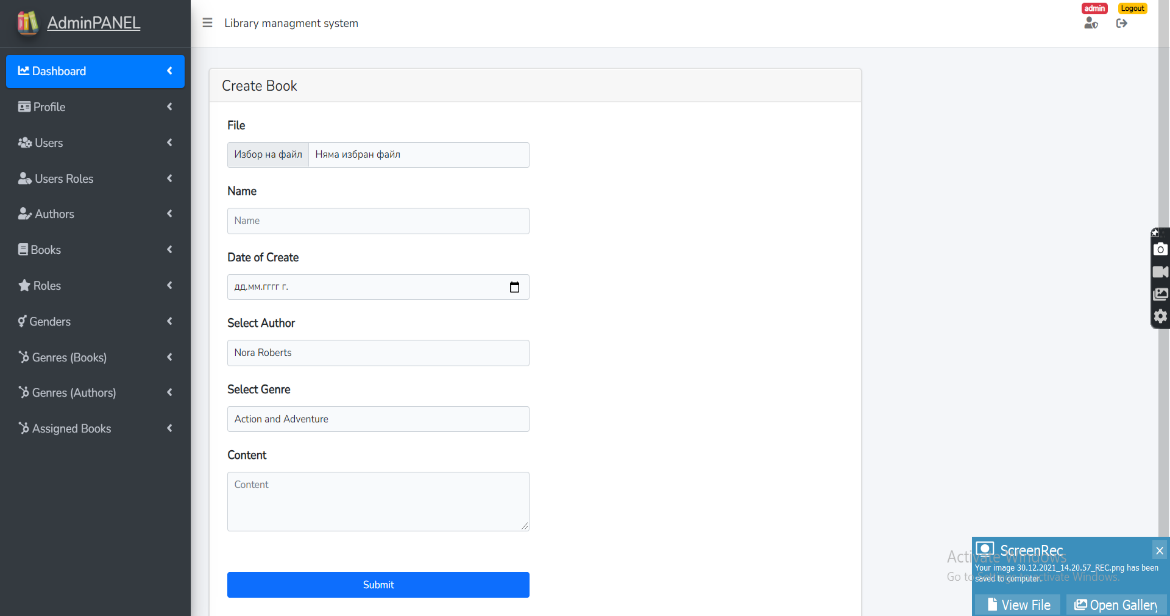
**4.5. Преглеждане на всички автори**



**Фигура 11. Преглеждане на всички автори.**

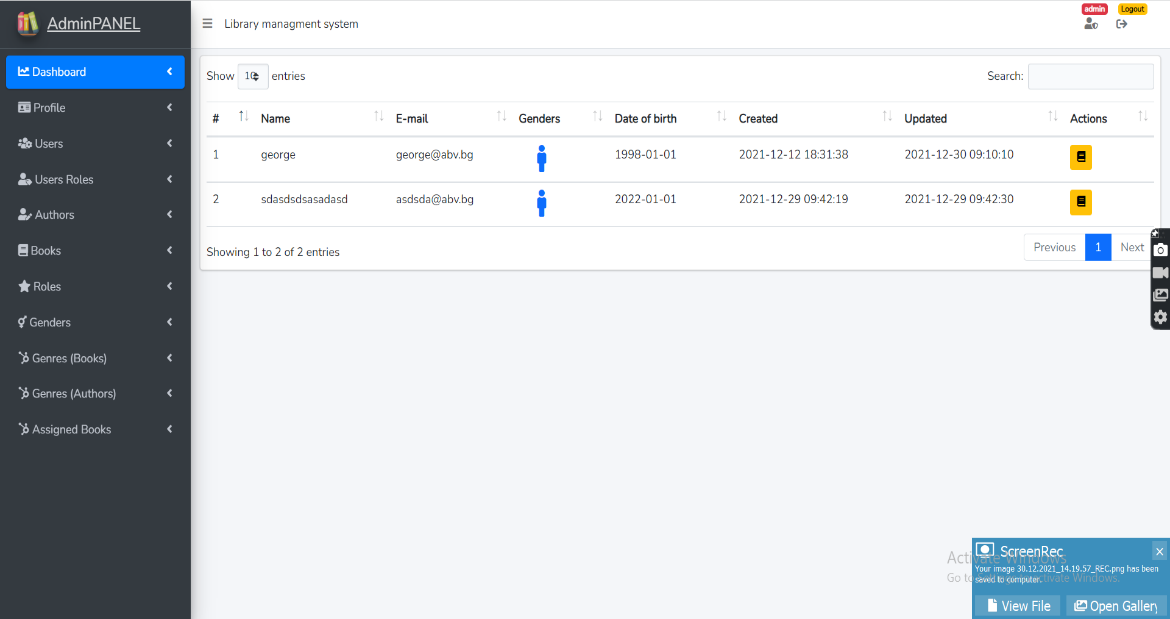
**4.6. Преглеждане на информация за книга**

* Редактиране
* Изтриване
* Всички потребители имащи тази книга



**Фигура 12. Преглеждане на книга.**

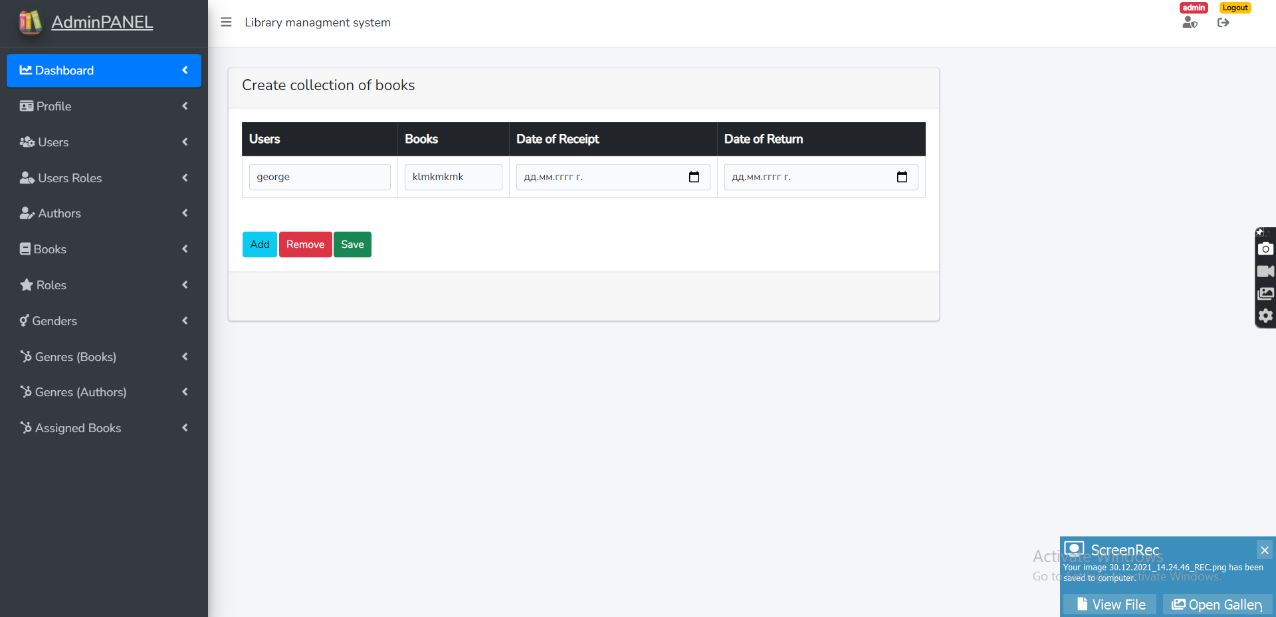
**4.7. Преглеждане на всички потребители от тип посетител и админ**



**Фигура 13. Преглеждане на книга.**

**4.8. Задаване на книга към потребител**

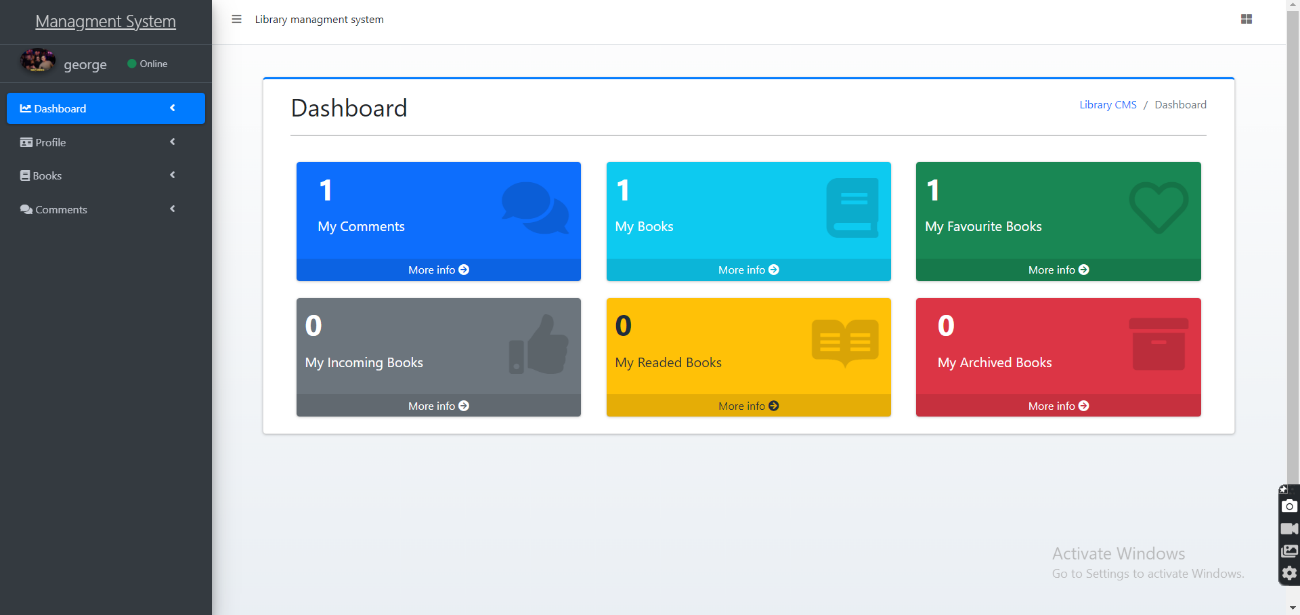
* Потребител
* Книга
* Дата на вземане
* Дата на връщане



**Фигура 14. Задаване на книга към потребител.**

**4.9. Табло на потребител**

* Преглеждане на неговите коментари
* Общо наброй книги
* Общо на брой книги със статус любими
* Общо на брой книги със статус прочетени
* Общо на брой книги със статус архивиране
* Общо на брой книги със статус предстоящи



**Фигура 15. Табло на потребител**

# 

# **Заключение**

Библиотеката е съвкупност от източници на информация. Ресурсите в нея се използват от дефинирана общност, включваща читатели, студенти и работещи хора, които да споделят или да заемат книгите по-удобно. Софтуерът на системата за управление на библиотеката се използва за лесно намиране на книги и достъп до списания. Системата за автоматизация на библиотеките автоматизира типичните процедури и намалява натоварването на библиотечния персонал. Когато хората ценят информацията все повече и повече, информационната индустрия се развива и технологията променя очакванията на посетителите на библиотеките. Това дава както възможност, така и предизвикателство за библиотеките. Интегрираната библиотечна система се използва за управление на по-сложните дейности и дава възможност на библиотекаря да управлява ресурсите по по-ефективен начин, за да спести време и усилия.

# **Източници**

**1. W3schools- сайт за свободно развитие във софтуерната индустрия**

W3Schools е безплатен образователен уебсайт за обучение на кодиране онлайн. Разработен през 1998 г., той получава името си от World Wide Web, но не е свързан с W3C. W3Schools предлага курсове, обхващащи всички аспекти на уеб разработката .

Уебсайт: *„https://www.w3schools.com“*

**2. Mozila Developer Center - сайт за със материали за уеб разработка**

Mozilla Developer Center, е хранилище за документация и учебен ресурс за уеб разработчици, използвани от Mozilla, Microsoft, Google и Samsung.

Уебсайт: *„https://developer.mozilla.org/en-US/“*

**3.Bootstrap – Софтуерна рамка, даваща допълнителни инструменти за разработване на потребителски интерфейс**

Официалния сайт на Bootstrap ни предоставя възможността да използваме библиотеката за стилизиране на уеб компоненти, начин на инсталиране, внедряване на библиотека във нашето приложение, използването на готови стилизиране и добре оптимизиране функционалности и шаблони, за по-добър интерфейс и оптимизация от страната на клиента.

Уебсайт: *„https://getbootstrap.com/“*

**4. DataТабле - Библиотека коята предоставя готови таблици със функционалност за разработване на уеб приложения**

Официалния сайт на Data table ни предоставя възможността да използваме библиотеката за стилизиране на уеб таблици, начин на инсталиране, внедряване на библиотека във нашето приложение, използването на готови стилизиране и добре оптимизиране функционалности и шаблони, за по-добър интерфейс и оптимизация на нашите страници от страната на клиента.

Уебсайт: *„https://datatables.net/“*

# **ПРИЛОЖЕНИЕ**

**А/ Файл 1: UserController.php**

<?php

namespace App\Http\Controllers;

use App\Models\AssignBook;

use App\Models\Book;

use App\Models\Gender;

use App\Models\Role;

use App\Models\User;

use Illuminate\Support\Facades\DB;

use App\Models\UserFlags;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class UserController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

$data['allData'] = User::whereNotNull('last\_seen')

->orderBy('last\_seen', 'asc')

->get();

return view('admin.crud.users.index', $data);

}

/\*\*

\* Show the form for creating a new resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function create()

{

$data['roles'] = Role::all();

$data['flags'] = UserFlags::all();

$data['genders'] = Gender::all();

return view('admin.crud.users.create', $data);

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

$user = new User();

$user->name = $request->name;

$user->email = $request->email;

$user->flag\_id = $request->flag\_id;

$user->role\_id = $request->role\_id;

$user->gender\_id = $request->gender\_id;

$user->dob = $request->dob;

if ($request->file('image')) {

$file = $request->file('image');

$filename = date('YmdHi') . $file->getClientOriginalName();

$file->move(public\_path('upload/images'), $filename);

$user['image'] = $filename;

}

$user->password = bcrypt($request->password);

$user->save();

return redirect()->route('user.index')->with('message', 'Successfully created data!');

}

// $validated =$request->validated();

// User::create($validated);

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

//

}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit(User $user)

{

$genders = Gender::all();

$roles = Role::all();

$flags = UserFlags::all();

return view('admin.crud.users.edit',compact('user'),[

'genders'=>$genders,

'roles'=>$roles,

'flags'=>$flags,

]);

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

$user = User::where('id', $id)->first();

$user->name = $request->name;

$user->email = $request->email;

$user->role\_id = $request->role\_id;

$user->gender\_id = $request->gender\_id;

$user->flag\_id = $request->flag\_id;

$user->dob = $request->dob;

if ($request->file('image')) {

$file = $request->file('image');

$filename = date('YmdHi') . $file->getClientOriginalName();

$file->move(public\_path('upload/images'), $filename);

$user['image'] = $filename;

}

$user->password = bcrypt($request->password);

$user->save();

return redirect()->route('user.index')->with('message', 'Successfully updated data!');

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function delete($id)

{

$user = User::find($id);

$user->delete();

return redirect()->route('user.index');

}

public function adminIndex()

{

$data['allData'] = DB::table('users')

->where('role\_id', '=', 2)

->whereNotNull('last\_seen')

->orderBy('last\_seen', 'DESC')

->get();

return view('admin.crud.users.adminView', $data);

}

public function visitorIndex()

{

$users = DB::table('users')

->where('role\_id', '=', 1)

->whereNotNull('last\_seen')

->orderBy('last\_seen', 'DESC')

->get();

$flags = UserFlags::all();

return view('admin.crud.users.visitorView', ['users'=>$users, 'flags'=>$flags]);

}

public function visitorBook($id)

{

$data['allData'] = AssignBook::all()->where('user\_id', $id);

return view('admin.crud.books.assigned-books', $data);

}

public function comments($id)

{

$data['comments'] = User::find($id)->comments;

return view('admin.crud.users.comments.comments', $data);

}

}

**Б/ Файл 2: HomeController.php**

<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;

use Illuminate\Http\Request;

class HomeController extends Controller

{

/\*\*

\* Create a new controller instance.

\*

\* @return void

\*/

public function \_\_construct()

{

$this->middleware('auth');

}

/\*\*

\* Show the application dashboard.

\*

\* @return \Illuminate\Contracts\Support\Renderable

\*/

public function index()

{

if (Auth::user()['roles']['name'] == 'admin') {

return redirect()->route('dashboard.index');

}else if (Auth::user()['roles']['name'] == 'visitor') {

return redirect()->route('visitor.view.dashboard');

}

}

}

**B/Файл 3: Login Controller**

<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Support\Facades\Auth;

use Illuminate\Foundation\Auth\AuthenticatesUsers;

class LoginController extends Controller

{

/\*

|--------------------------------------------------------------------------

| Login Controller

|--------------------------------------------------------------------------

|

| This controller handles authenticating users for the application and

| redirecting them to your home screen. The controller uses a trait

| to conveniently provide its functionality to your applications.

|

\*/

use AuthenticatesUsers;

protected function authenticated()

{

if (Auth::user()->role\_id == '2' && Auth::user()->flag\_id == '3') {

return redirect()->route('dashboard.index');

} else if (Auth::user()->role\_id == '1' && Auth::user()->flag\_id == '2' || Auth::user()->flag\_id == '1' ) {

return redirect()->route('visitor.view.dashboard');

}

}

/\*\*

\* Where to redirect users after login.

\*

\* @var string

\*/

// protected $redirectTo = RouteServiceProvider::HOME;

/\*\*

\* Create a new controller instance.

\*

\* @return void

\*/

public function \_\_construct()

{

$this->middleware('guest')->except('logout');

}

}

1. Автор: Start.bg, Адрес: <http://php.start.bg>,

   Име на статията: Какво е PHP, Достъпно към: 15.04.2022 [↑](#footnote-ref-1)
2. Reasons to Use MySQL Адрес: <http://www.databasequest.com>

   [https://bg.theastrologypage.com/web-development](%20https:/bg.theastrologypage.com/web-development)

   [https://laravel.com/](%20https:/laravel.com/)

   Достъпно към: 15.04.2022 [↑](#footnote-ref-2)