Server-side home project (Laravel)

Submit Assignment

Due Apr 19 by 11:59pm **Points** 60 **Submitting** a file upload **File Types** zip
Available until Apr 19 at 11:59pm

Your task is to implement a very simple online book rental system (BRS).

There are functions which are open for anonymous users. They can

- search for books by author or title,
- list books by genre,
- view the data sheet of a selected book.

There are two types of users in this BRS: readers and librarians. A registered and authenticated (logged in) reader can

- borrow a book,
- view his/her active book rentals, and
- view the details of a selected book rental.

A librarian can

- add, edit or delete a book,
- add, edit or delete a genre,
- list book rentals,
- view the details of a book rental,
- change some status on a book rental, like status, deadline, note.

The home project must be implemented as a Laravel application using a local SQLite database. The tables must be created by migrations and seeded with some fake data. During evaluation we will run the following commands to start up your application (so you have to try them in this order before you submit your solution):

```
composer install
npm install
npm run prod
php artisan migrate:fresh
php artisan db:seed
php artisan serve
```

The solution must be submitted to Canvas as a zip archive. **Before uploading, the `vendor` and `node_modules` folder must be deleted!**

Models

- User
  - *Note: This table is already created. You just need to extend it!*
  - id
  - name (string)
  - email (string, unique)
  - email_verified_at (timestamp, nullable)
  - password (string)
  - is_librarian (boolean, default:false)
  - remember_token
  - timestamps (created_at, updated_at)
- Book
  - id
  - title (string, 255)
  - authors (string, 255)
  - description (text, nullable)
  - released_at (date)
  - cover_image (string, 255, nullable)
  - pages (integer)
  - language_code (string, 3, default:hu)
  - isbn (string, 13, unique)
  - in_stock (integer)
  - timestamps (created_at, updated_at)
- Genre
  - id
  - name (string, 255)
  - style (enum: primary, secondary, success, danger, warning, info, light, dark)
  - timestamps (created_at, updated_at)
- Borrow
  - id
  - reader_id (unsignedBigInteger, foreign, references:id, on:users)
    - *The reader who would like to borrow it*
  - book_id (unsignedBigInteger, foreign, references:id, on:books)
    - *The book to be borrowed*
  - status (enum: PENDING, ACCEPTED, REJECTED, RETURNED)
    - The status of the rental:
      - *PENDING: the reader started a rental process*

        - *REJECTED: the librarian rejected the rental request*
        - *ACCEPTED: the librarian has accepted the rental request, i.e. the book is at the reader*
        - *RETURNED: the reader brought back the book, and the librarian approved it*
    - request_processed_at (datetime, nullable)
        - *The time when the rental request (pending) became rejected or accepted*
    - request_managed_by (unsignedBigInteger, nullable, foreign, references:id, on:users)
        - *Librarian id who administered the rental request*
    - deadline (datetime, nullable)
        - *If the librarian accepts the rental request, he/she can specify a deadline by which the book must be returned to the library*
    - returned_at (datetime, nullable)
        - *The time when the librarian administered the return of the book*
    - return_managed_by (unsignedBigInteger, nullable, foreign, references:id, on:users)
        - *Librarian id who administered the return of the book*
    - timestamps (created_at, updated_at)

- A book may have many genres, and a genre may belong to many books (Book N >--< Genre)
- A book may have many rentals, but a rental belongs to only one book (Book 1 --< Borrow)
- A reader may have many book rentals, but a rental belongs to only one user (User 1 --< Borrow.reader_id)
- A librarian may manage many rental requests, but a rental request can only managed by one librarian (User 1 --< Borrow.request_managed_by)
- A librarian may manage many book return, but a book return can only managed by one librarian (User 1 --< Borrow.return_managed_by)

In the `User` model you can define multiple relationships between two models by providing the foreign key reference column:

```
class User extends Authenticatable
{
  // ...

  public function readerBorrows() {
    return $this->hasMany(Borrow::class, 'reader_id');
  }

  public function managedRequests() {
    return $this->hasMany(Borrow::class, 'request_managed_by');
  }

  public function managedReturns() {
    return $this->hasMany(Borrow::class, 'return_managed_by');
  }
}
```

You can extend your models by introducing business methods according to your needs. For example the `Book` model can be extended with methods like this:

```
class Book extends Model
{
  // ...

  public function borrows() {
    return $this->hasMany(Borrow::class, 'book_id');
  }

  public function activeBorrows() {
    return $this->getAllBorrows()->where('status', '=', 'ACCEPTED');
  }
}
```

Database

Create a local SQLite database file in the `database/` folder, like `database/database.sqlite`.

Design

Try to create a good design. For this you can use any CSS framework and component library, e.g. Twitter Bootstrap is highly recommended.

Guides

Below you can find materials or tips for the task. Basically focus on the function of the tasks and fulfill the expected requirements! The default Bootstrap design is perfect option, but the overall design is fully entrusted to you.

- **Laravel documentation** (https://laravel.com/docs)
- **PHP documentation** (https://www.php.net/manual/en/)
- **Bootstrap documentation** (https://getbootstrap.com/docs/)
- **DB Browser for SQLite** (https://sqlitebrowser.org/)
- Other tips
  - **Fontawesome icon set** (https://fontawesome.com/)
  - **Tailwind CSS** (https://tailwindcss.com/)
  - **Materialize** (https://materializecss.com/)
  - **Bulma** (https://bulma.io/)

In addition to the listed ones, any other CSS framework can be used.

Tasks

Public functions

1. **Navigation bar** The application has to use a common layout, at least containing a navigation bar
2. **Main page** The main page shows the following informations:

- Number of users in the system
- Number of genres
- Number of books
- Number of active book rentals (in accepted status)
- List of genres. Each list item must be a link, referring to the *List by genre* page.
- Search for books. See *Search*.

3. **List by genre** This page shows the specific genre name (coming as an URL parameter) and lists all the books belonging to that genre (book title, author, date, description). Clicking on a book the browser should show the *Book detail* page.

4. **Search** On the *Main page* you can search for a book by giving a filter string for the title or the author. It has to match each book that has a title or author field containng the filter text. The result list should be displayed on a separate page, listing the corresponding books (book title, author, date, description). Clicking on a book the browser should show the *Book detail* page.

5. **Book detail** A page which shows every informations about the specific book:

   - title
   - author
   - genre
   - date of publish
   - number of pages
   - language
   - ISBN number
   - Number of this book in the library (in_stock)
   - Number of available books (which are not borrowed)
   - description
   - cover image if it is given

6. **Registration** A guest user can create a new user account in the system on a registration form. The registered user will be a reader by default. We cannot create librarians on the user interface. You can use Laravel's built-in solution.

7. **Login** A guest user can log in to the system. After a successful login the user is redirected to the *Main page*. **Please prepare (seed) at least one reader and one librarian in the system with emails** `reader@brs.com` **and** `librarian@brs.com`**, respectively, and with password** `password`**!** If you use any other credentials, display them on the login page!

Reader functions

8. **Authorization** The following functions can be accessed only by authenticated users.

9. **Borrow a book** If the user is a reader, then on the *Book detail* page two more items show up:

   - whether this reader has an ongoing rental process with this book; and

- a "Borrow this book" button appears on the page.

Clicking on this button, a new rental request is created in the system with PENDING status, and the browser is redirected to the *Book details* page.

The reader cannot borrow the very same book while there is an ongoing rental request on the very same book by the user (not RETURNED).

10. **My rentals** This page can be accessed from the navigation bar after logging in. This page shows five lists:

    1. Rental requests with PENDING status
    2. Accepted and in-time rentals (before the deadline)
    3. Accepted late rentals (after the deadline)
    4. Rejected rental requests
    5. Returned rentals

    The list items show the book title, author, date of rental, and the deadline. Each list item is a link which leads to the *Rental details* page.

11. **Rental details** The data sheet of a rental contains the following informations:

    - Book
      - Title, author, date
      - Link to the *Book detail* page
    - Rental data
      - Name of the borrower reader
      - Date of rental request (created_at)
      - Status
      - if it is not PENDING
        - Date of procession
        - Librarian's name
      - if it is RETURNED
        - Date of return
        - Librarian's name
      - if the rental is late, show this information, too.

Librarian functions

12. **Authorization** The following functions can be accessed only by librarian users.

13. **Add new book** This function can be accessed from the navigation bar. This page shows a form where we can add the data of the new book:

    - *title*, text, required, max. 255 characters
    - *authors*, text, required, max. 255 characters
    - *released_at*, date, required, date, before now (`before:now`)

- *pages*, number, required, at least 1
- *isbn*, text, required, regex pattern: `/^(?=(?:\D*\d){10}(?:(?:\D*\d){3})?$)[\d-]+$/i`
- *description*, textarea, can be empty
- *genres*, checkboxes, array of ids
- *in_stock*, number, required, at least 0

If the validation fails, show the error messages, and keep the form state. If the validation succeeds, save the book, and redirect the browser back to the main page.

14. **Edit a book** On the *Book details* page show an "Edit" button. Clicking this, a new page shows the *Add new book* page-like form, prefilled with the data of the selected book. The data can be modified and submitted. Validation rules are the same, on error show the messages and keep the form state, on success show the *Book details* page!

15. **Delete a book** On the *Book details* page show a "Delete" button. Clicking this the book is deleted from the system. After this show the *Main page*!

16. **Genre list** This function can be accessed from the navigation bar. This page shows the list of genres stored in the system (name, style).

17. **Add new genre** On the *Genre list* page above the list show an "Add new genre" button. This button leads to a form where we can add the data of the new genre:

    - *name*, text, required, at least 3, at most 255 characters
    - *style*, dropdown, with values: `primary, secondary, success, danger, warning, info, light, dark`

    If the validation fails, show the error messages, and keep the form state. If the validation succeeds, save the genre, and redirect the browser back to the *Genre list* page.

18. **Edit a genre** On the *Genre list* page beside each list item there is an "Edit" button. Clicking this a new page shows the *Add new genre* page-like form, prefilled with the data of the selected genre. The data can be modified and submitted. Validation rules are the same, on error show the messages and keep the form state, on success show the *Genre list* page!

19. **Delete a genre** On the *Genre list* page beside each list item there is a "Delete" button. Clicking this the genre is deleted from the system. After this show the *Genre list* page!

20. **Rental list** This page can be accessed from the navigation bar after logging in. This page shows five lists:

    1. Rental requests with PENDING status
    2. Accepted and in-time rentals (before the deadline)
    3. Accepted late rentals (after the deadline)
    4. Rejected rental requests
    5. Returned rentals

    The list items show the book title, author, date of rental, and the deadline. Each list item is a link which leads to the *Rental details* page.

21. **Rental details (librarian)** This page is the same or contains the same data as *Rental details* page. Under the data sheet put a form for the librarian where he/she can change the status and set a deadline for the rental. If there are errors, show them by keeping the form state, otherwise save them, and show the *Rental details (librarian)* page again.

Common function

22. **Profile page** The profile page can be accessed from the navigation bar. The profile page shows the user's name, email address and role (reader/librarian).

Other features

23. **Soft delete** Try to use **soft deletes (https://laravel.com/docs/9.x/eloquent#soft-deleting)** for deleting genres, books.

Statement

The solution must contain a `README.md` file that must include the following information:

```
<Name>
<Neptun ID>
This solution was submitted and prepared by stundent named above for the home assignment of the Web engineering c
ourse.
I declare that this solution is my own work.
I have not copied or used third party solutions.
I have not passed my solution to my classmates, neither  made it public.
Students’ regulation of Eötvös Loránd University (ELTE Regulations Vol. II. 74/C. § ) states that as long as a st
udent presents another student’s work - or at least the significant part of it - as his/her own performance, it w
ill count as a disciplinary fault. The most serious consequence of a disciplinary fault can be dismissal of the s
tudent from the University.
```