2nd Homework - a Laravel MVC Application

You should supplement a simple training project *Ticketsys*, which is created using Laravel MVC framework. The application implements the minimal functionality of the event ticket ordering system. The design is very simple and implemented with Bootstrap, so do not pay attention to it. The code does not include a lot of comments, since it is quite understandable, but in order to fully understand the code, you should study Laravel documentation (course materials cover approximately 90% of the code). The code is designed to make it as close to the course materials as possible, therefore, sometimes it may not follow the best practices.

Existing Functionality

The system stores a list of events and tickets of them. Each event has a name, description, start time, location, rating, number of rows, number of seats and category which is connected with the event by means of a One-to-Many relationship. Each ticket corresponds to a certain row and seat and has a certain price. The system also includes a shopping cart, where users can add and remove tickets. A user may add several tickets for various events in a shopping cart. Some users of the system are administrators. Administrators can add categories, events and tickets.

Homework Assignment

To supplement the ticketsys application with the MVC components, adding a checkout functionality.

Detailed Assignment Requirements

Basic Requirements (8 points)

There are blank or partially completed MVC components. You should supplement them to implement the following functionality (you should not create any new components):

1. After viewing his/her cart, a user can place an order (shopping cart functionality has already been implemented)
2. It should be checked, whether the user is authenticated. If the user is authenticated, then he/she should be immediately redirected to the ordering page. If not, then a user should be authenticated, and then forwarded to the ordering form. That can be achieved by adding *auth* middleware to the OrderController constructor for your new methods.
3. The ordering page shows events that the user has selected, the tickets, their prices and the total cost of the order (the same information as available in the shopping cart), and the user must enter his/her delivery address and a telephone number.
4. After placing an order, the user is redirected to a page which displays all order details.
5. There is an item "Orders" already included in the menu. When a user clicks this menu item, a page with a list of all orders made by a user must be displayed. Each order in that page must have a hyperlink to another page which shows an information about each individual order.

Model

The project includes models **Order** and **TicketOrder** (empty files). You should add code to them to use them in your application. To use models, you should also think out and define the necessary table structure by using migrations **2019_05_03_091414_create_orders_table.php** and **2019_05_03_091605_create_ticket_orders_table.php**.

There should be a One-to-Many relationship between **User** and **Order** models. There should also be a One-to-Many relationship between **Order** and **TicketOrder**. Finally, there should also be a One-to-Many or One-to-One relationship between **Ticket** and **TicketOrder** models (TicketOrder is a bridge table between Order and Ticket). All mentioned relationships must be defined both in the code of models and in the database table definitions (migrations).

Order model must include a shipping address (string data type) and phone number (integer data type). Do not forget to define foreign keys in the schema to make One-to-Many relationships work.

View

The project includes empty views **order_create.blade.php**,
orders.blade.php and **order_show.blade.php**.

You should implement the order placing functionality in the first view (order_create.blade.php), i.e. it must be possible to enter a shipping address and telephone number. This view should display all the selected events, ticket prices and the total amount for the entire order. You should implement input fields using a form (Laravel Collective Form class). You should show error messages if the fields are not filled or the phone number contains not only digits. If the form is filled correctly and submitted after saving the order information in the database, the user must be forwarded to the view order_show.

The view order_show.blade.php must show all order details - all that was shown and entered in the first view, all the information about the user. This view can only be available to the customer that made that order.

The view orders.blade.php must show the list of all orders placed by a user and must include hyperlinks to the detailed information about each order (the view order_show.blade.php). This view can only be available to the customer that made these orders.

Controller

The project includes **OrderController**, which only includes mock-up methods. You should add the order placing functionality to it.

You must implement the corresponding methods in the controller class. Certain methods must show order placing and observing views, another method must process the ordering form.

The controller should check, whether the user is logged in at the time of ordering and make the appropriate forwarding.

The order placing form must be appropriately validated using `validate` helper function or `Validator` class. Model instances must be created, model properties must be assigned appropriate values, models must be related and saved in the database.

Advanced Requirements (1 point)

Add the following functionality to the system:

1. Administrators can view a list of all orders by all users, including his/her own, (in addition to the basic requirements, you should add the administrators access) and must be able to see details about every order (second and third views of the basic requirements)
2. Administrators can mark orders as fulfilled
3. Fulfilled and unfulfilled orders should be visually distinguishable (no matter how – different color, background, different lists on one page, separate pages for fulfilled/unfulfilled orders)

The implementation is entirely of your choice, but it should be described and available when visiting http://ticketsys.test/admin. Unlike in case of the basic requirements, you can add new files to the project.

i-option (1 point)

Add the advanced validation functionality to the system:

1. When users buy tickets, such tickets must be visually distinguishable in the view event.blade.php and must become unavailable for other users to order (an error must be shown if one tries to add a ticket already bought by other user to the shopping cart or make an order).
2. When an administrator adds a new event ticket to the system and specifies a row/seat and a price, the system must check that the entered row and seat are less than the total number of rows and seats defined for the event ('rows' and 'seats' columns of the events table) and that there is no ticket present in the tickets table with the same row and seat for the same event.

Downloading the Project and Running it on your Computer

The project can be obtained and installed like in Laravel instructions included into the course materials. Before installing this project, it is recommended to create a new Laravel project in accordance with the instructions in the course materials. It will allow you to configure the development environment. The actions needed to be performed to install this project follow.

Obtaining and Installing the Source Code

If you do not have git installed on your Windows computer, you should install it. You can download git here and install it with the default settings.

Add MySQL user *ticketsys* with a password *ticketsys*. Create a database with the name *ticketsys*. Assign all privileges for the database *ticketsys* to the user *ticketsys*. These are the database settings used in the project.

Open the console and go to the Web server root folder:

- Windows: *cd c:|wamp|www* (if you are using 64x version *www* must be substituted with *www64* here and later in the instutions)
- Linux example setup: *cd /var/www/html*

Execute a git command: *git clone https://github.com/webtech2/ticketsys.git ticketsys*

Go to the new project folder *ticketsys: cd ticketsys*

Run composer to install the dependencies: *composer install*

If you are using Linux, assign privileges: *sudo chown -R :www-data storage*

Rename *.env.example* file to *.env*. Enter the database connection details to *.env* and APP_NAME=TicketSys. Execute the following command in the project's directory: *php artisan key:generate*. Copy the generated key into the *APP_KEY* value of the *.env* file.

Apache Virtual Host Configuration

If you are using Windows computer, go to *c:|wamp|bin|apache|apache…|conf|extra* and copy virtual host definition into *httpd-vhosts.conf*: (after that restart Apache)

```
<VirtualHost *:80>
  ServerName ticketsys.test
  DocumentRoot ${INSTALL_DIR}/www/ticketsys/public
  <Directory ${INSTALL_DIR}/www/ticketsys/public>
    AllowOverride all
  </Directory>
</VirtualHost>
```

If you are using Linux, create a new file */etc/apache2/sites-available/cinema.conf* (you can do that with the command *nano /etc/apache2/sites-available/cinema.conf*). The contents of that file should be: (after that execute the commands *a2ensite cinema* and *service apache2 reload*)

```
<VirtualHost *:80>
  ServerName ticketsys.test
  DocumentRoot /var/www/html/ticketsys/public
  <Directory /var/www/html/ticketsys/public>
    AllowOverride all
  </Directory>
</VirtualHost>
```

Host Configuration (on the computer with Netbeans)

To open an address http://ticketsys.test/ in the browser, make the following changes

- In case of Windows change *C:|Windows|system32|drivers|etc|hosts* and add there the following row *127.0.0.1 ticketsys.test* (or if you are using Linux example setup virtual machine *172.16.172.2 ticketsys.test*)

Before Launching the Application

Before launching the application for the first time, execute the following command in the project's directory: *php artisan migrate* (this will create database tables).

If you decide to reload the database structure, execute the following command *php artisan migrate:refresh.*

After migrating tables or if you decide to reload the database data, execute the following command *php artisan db:seed*

After that your database should include the necessary tables and the initial data.

Usage

You can authenticate as an administrator with the username admin@ticketsys.test and password *secret*.

Submission

Please, submit the archive of your project excluding the directory *vendor*.