Aplicações para a Internet

Engenharia Informática	2º Ano	2º Semestre	2023-24	Avaliação Periódica
Projeto	Prazo para divulgação resultados: 1 Julho 2024			
Data: 20 Abril 2024	**Data de Entrega: 17 Junho 2024**			

Project – CineMagic

OBJECTIVE

The aim of this project is to implement a server-based web application, using the Laravel Framework, for the company CineMagic, which organizes movie screening sessions in its theatres and will use the web application to sell tickets and control entries to movie screening sessions.

SCENARIO

Users (including anonymous users) of the "CineMagic" cinema application will be able to access information about the films showing in the "CineMagic" theaters (name, poster, synopsis, *trailer*, etc.), as well as information about each of the screening sessions (theater, movie, date, time), including the seats occupied and available for each screening.

CineMagic customers will be able to buy tickets for the movie screenings through the application. Ticket purchases are made via a shopping cart and payment is made online (the payment process is simulated). Registered customers of the CineMagic application will have a discount for each ticket purchased and have online access to their tickets and purchases' history – this requires a registration process and user credentials (email + password) to enter the application. Non-registered customers (anonymous users) will be able to buy tickets, but do not have online access to their history (tickets and purchases). Instead, they should receive their tickets and/or purchases receipts on their email or print that information after successfully completing a purchase.

The "CineMagic" employees will be responsible for controlling access to the screening sessions, confirming via the app that the tickets presented at the entrance are valid. If the tickets are valid, they allow access to the screening session and use the application to invalidate the tickets. They have the power to manually invalidate any ticket without allowing access to the screening session (e.g. a child trying to view an adult movie or an intoxicated person trying to enter screening session).

The administrators of the "CineMagic" will use the application to configure some business parameters, manage the theaters and their seats, manage the movies being shown and their screening sessions. They are also responsible for user management, i.e. creating, changing, and deleting administrators and employees, and blocking/unblocking customers. Finally, they will also have access to statistical information about the "CineMagic" business (tickets sold, overall

occupancy percentages, by movie, by month, by day of the week, etc.), as well as access to all purchases (including receipts) and tickets.

GROUP OD FUNCTIONALITIES

The web application must implement a set of groups of functionalities, which will be described below. Their implementation must consider the information described in the scenario, in the database (in the structure and description of the database) and in the description of the groups of functionalities, as well as all the standards and good practices of the Laravel Framework. In addition to the business model, requirements and restrictions that can be inferred from the statement and the database structure, students are free to decide on the user interface and usability, as well as on aspects of the business that are not described.

1. AUTHENTICATION, PROFILE AND USER MANAGEMENT (15%)

The web application to be developed should support authentication through a login with the credentials e-mail + password - the login process is the same for any type of user. Once authenticated, any user can change their password and log out of the application. If the user can't remember their password, they should be able to *reset it* - the application should send them an email with a link to do this ("*password reset*").

Anonymous users (unauthenticated users), in addition to accessing the login for authentication, can register as customers. Customer registration is carried out by the application without the intervention of administrators or staff. Once the registration process has been completed, the application will send an e-mail to verify/confirm the validity of the e-mail.

Registered customers have access to their user profile, where they can view and change their personal details - name, NIF number (optional), default payment details (optional) and photo/avatar (optional). Employees do not have access to their user profile, as only administrators can view and change their personal information (but they can change the password or log out of the application).

Administrators are responsible for managing the users of employees and administrators, which means they can consult, filter, create, change, or remove the accounts of employees and administrators (except for removing themselves), as well as access and change the user profile of any employee or administrator. On the other hand, even administrators cannot access registered customers' user profiles - they can only consult and filter the registered customer list, block/unblock or delete (using *soft delete*) registered customer accounts.

Note: emails should be sent via the mailtrap.io.

2. MOVIES ON SHOW (15%)

All users of the application, including anonymous users (any user is anonymous before logging in) should be able to consult the movies on show at the "CineMagic" theaters. A movie is considered

to be on show, if it has at least one screening session for today or any day in the next 2 weeks. For each movie, the application should show the title, poster, synopsis (summary), trailer (if there is one), as well as the screening sessions of the movie (only the screening sessions for today or in the next 2 weeks). For each screening session the application should show the theater, date, start time and something to indicate whether the screening session is sold out or not.

The application should also make it possible to search (filter) movies by genre and by a sub-string of the title or synopsis.

3. BUYING TICKETS (15%)

Ticket purchases are made via a shopping cart, in which users (anonymous or registered customers) can add or remove tickets for any screening session that started up to 5 minutes ago or that starts at a later date and time (for example, if a session starts at 3pm on a certain day, the user can add tickets for that session until 3:05pm on the same day, regardless of the time the shopping cart is completed).

At any time, the user can delete or add tickets to the shopping cart and clear the shopping cart completely (without finalizing the purchase).

After the user adds all tickets to the shopping cart, he will have to make the final confirmation of the purchase, where he adds the final details for the purchase: the customer's name, email and NIF, and the payment details (payment type and payment reference). If the user is a registered customer (has logged in as a registered customer), ticket prices will have a discount (current ticket prices and discounts are defined on the "configuration" table) and the final details for the purchase (name, email, NIF, payment type and payment reference) are pre-filled with his own values (defined on the customer profile).

Payment can be made by Visa card (16 digits + 3-digit CVC code), PayPal (email) or MBWay (mobile phone number). The payment process is simulated (a class will be provided to simulate the payment) and must be done immediately before the purchase is finalized. If the payment process is completed successfully (payment simulation returns true), the application will finalize the purchase process storing the purchase on the database, generating the corresponding tickets, and clearing the shopping cart. If the payment is not completed successfully (payment simulation returns false), the purchase is not finalized (nothing is stored in the database), the shopping cart remains as it is, and the user is warned that the payment is invalid.

Each purchase will include one or more tickets and the information needed to create the receipt for the purchase, namely: date, total price, NIF (optional), email and customer's name, payment details (type and reference). Each ticket must include an ID (which can be used as a reference when controlling access to screening sessions), the screening session (theater, movie, date and start time), the reserved seat (e.g. D3), the price and customer information (name, email and NIF).

4. CHOICE OF SEATS (10%)

As mentioned above, each ticket will be associated with a screening session (a movie in a certain theater at a certain time) and a specific seat in the theater (for that screening session). The application must ensure that each seat in a screening session is associated with a single ticket, or zero tickets while that seat is empty.

When a user adds a ticket to their shopping cart, they should choose the seat associated with the ticket in a way as appealing as possible, preferably with a visual representation of all the seats in the theater and including a way of identifying the occupied and free seats for the chosen session.

5. HISTORY, PURCHASES, RECEIPTS AND TICKETS (10%)

Once the ticket purchase process has been successfully completed (including payment), the application stores the purchase information and generates a receipt (PDF file) associated with the purchase and sends it (the receipt) automatically to the customer email (field "customer_email" of the purchase).

A copy of the receipt (PDF file) should be stored on the application storage folder. Both the purchase information page and the receipt (PDF file) should be immutable and available forever to any CineMagic administrator and to the associated registered customer (if the purchase was made by a registered customer).

The receipt must include the purchase number (automatic ID), date of purchase, payment details (type of payment and payment reference), customer's name, email and NIF, total price and the list of tickets associated with the purchase. Each ticket must include the ticket ID; theater, movie, date and time of the screening session; and the seat associated with the ticket.

Once the purchase process is complete, in addition to generating the purchase receipt, the application should also generate all the tickets associated with the purchase and automatically send them to the customer by email - in the same email message in which the receipt is sent. There are three options to generate ticket PDF files:

- Generate one PDF file for each ticket.
- Generate one PDF file that includes all tickets of the purchase.
- The receipt PDF file also includes all the tickets – tickets will not generate their own PDF files.

Ticket information should always be available in HTML format, i.e. each ticket should be available through a URL address to any CineMagic administrator and to the associated registered customer (if the purchase was made by a registered customer). Unlike the PDF receipts, application does not need to store the PDF ticket files – instead, it will generate them when the purchase is finalized and whenever the user (administrator or registered customer that owns the ticket) downloads it from the ticket page (downloads are only available for valid tickets).

Each ticket (in HTML or PDF) must include the ticket ID; theater, movie, date and time of the screening session; seat associated with the ticket; and the name, email and photo/avatar (if any) of the registered customer who bought the ticket (if it was bought by a registered customer).

The HTML version of the ticket should also include information on whether the ticket is still valid. The PDF version of the ticket should include a QR Code with an URL associated to the ticket (field "qrcode_url" of the tickets table) for access control purposes at the entrance to the session. Note that the "qrcode_url" must be a full URL and not easily replicated (should include a long random unique string) to avoid ticket theft.

6. SCREENING SESSIONS ACCESS CONTROL (10%)

The CineMagic employees will be responsible for controlling access to the screening sessions, confirming through the application that the tickets presented at the entrance are valid.

To do this, they must access a page where they define which screening session (a particular movie, in a particular theater, on a particular day and time) they are controlling. When a person presents a ticket, the employee scans the QR Code which opens the page specified by the QR Code URL (field "qrcode_url" of "tickets" table). Alternatively, he can use a page to enter the ticket ID (if the application does not support QR Codes) or add the QR Code URL manually on the browser (to simulate the QR code behavior without a scanning device). If the ticket is invalid (because it doesn't exist, because it's from a different screening session or because it's already been invalidated) the application warns the employee, and he forbids the person access to the screening session.

If the ticket is considered valid for the screening session in question, the application should show the ticket details (including the customer's name and photo) and inform the employee that the ticket is valid. At this point, the employee either allows or forbids access of the person to the screening session (he can forbid a child from seeing an adult film, an intoxicated person from entering the theater, or forbid access in any situation that he considers inadequate). Either way (allowing or forbidding access) the ticket must be invalidated by the employee or automatically by the application.

7. BUSINESS MANAGEMENT (15%)

In addition to managing users (*functionality group "1- Authentication, profile and user management"*) administrators are responsible for administering the business by managing theaters and seats, movies and movie genres, screening sessions and by defining configuration parameters such as the current ticket price and registered customer ticket discount.

With theater management it should be possible to create, change or delete (via *soft deletes*) the theaters that the company has, and for each theater it should be possible to define the seats available. Teachers will value solutions that make it as simple as possible to define seats - through some process that automates the creation of seats (instead of creating seats manually).

The management of movies and movie genres implies the possibility of creating, modifying, and deleting movies and movie genres (via *soft deletes*) - it should only be possible to delete movies that have no active screening sessions (screening sessions for today or later). The same applies to movies genres – only delete genres whose movies have no active screening session. For each movie it should be possible to define the title, genre, synopsis (summary), poster (by *uploading* an image file), trailer (external URL with the address of the trailer video).

Screening sessions management allows the administrators to create, change and delete screening sessions, where each screening session is associated to a movie, theater, date and a time (start time of the movie). Screening sessions can only be changed or deleted if no tickets have yet been sold for the screening session in question.

Teachers will value solutions in which the definition of screening sessions is as simple as possible, for example, making it possible to create several screening sessions simultaneously for a given movie and theater and/or for a set of dates and times that are repeated every day (instead of creating the sessions manually).

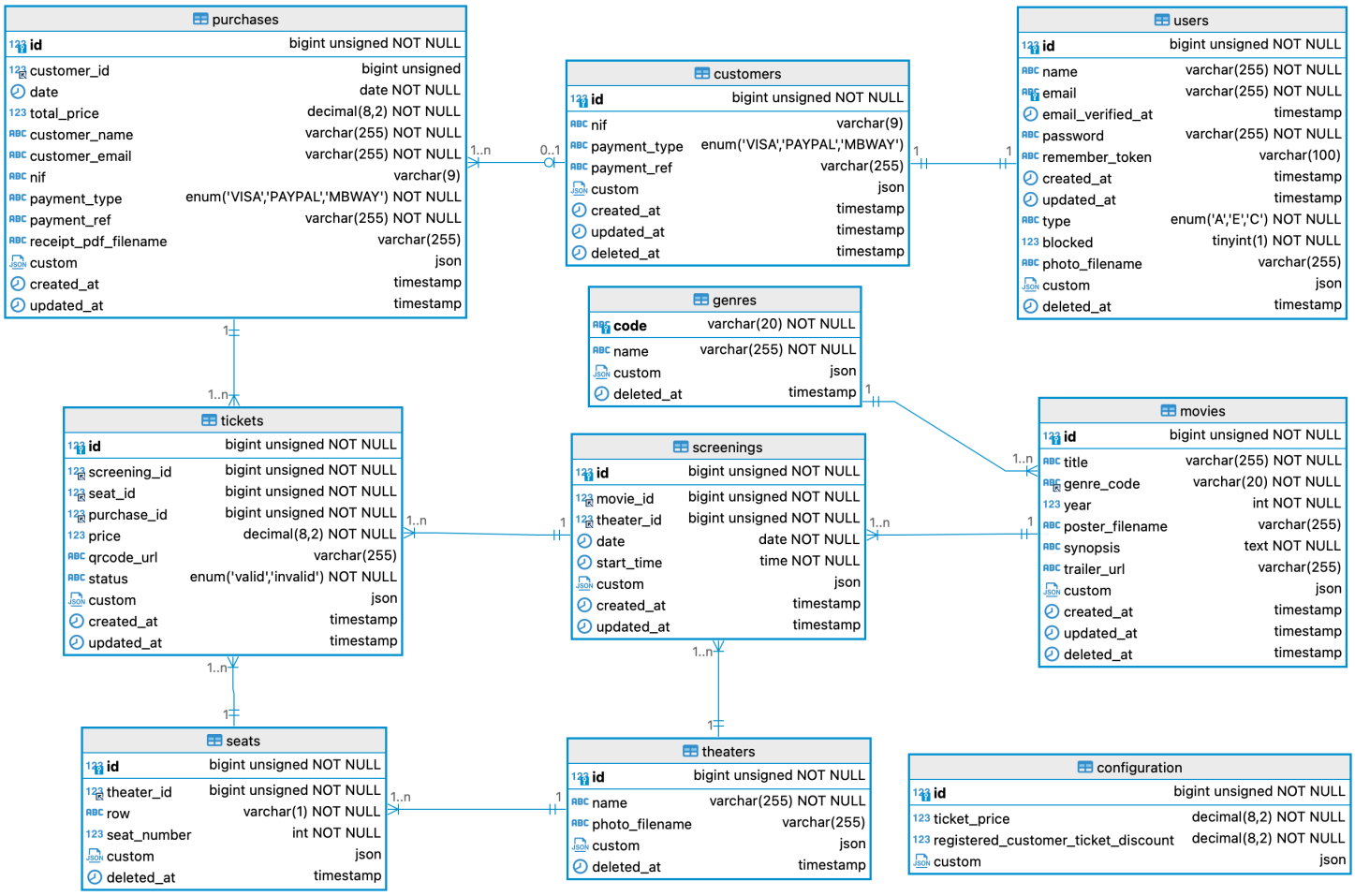
Note: The free seats for each screening session are defined indirectly by the seats in the theater that have not yet sold tickets for that particular session.

8. STATISTICS (10%)

Administrators should be able to view statistical information related to the "CineMagic" business. The decision on the form and type of information presented is the responsibility of the students. For example, the information can be presented in tables, graphs or other forms; it can be exported to Excel, CSV or other formats; it can include totals, averages, maximums, minimums of ticket sales by value or quantity, by month, by year, organized by films or categories of films, by customer; it can include percentages of theater occupancy by film, month, day of the week, overall, etc.

DATABASE

The structure of the database is provided (using Laravel migrations) and cannot be altered by the students, unless explicitly indicated by the teacher. Example data will also be provided through *database seeders*. The diagram for the database structure is the following:



TABLES AND COLUMNS

The tables and columns in the database are described below. The structure of the tables and columns (for example, the foreign keys, the data types of the columns, whether the columns are mandatory or not) and the description presented here are relevant to the data validation and to the business model of the application.

COMMON COLUMNS

All or some of the tables include a set of common columns that have the same meaning. The common columns are:

- "**custom**" – (optional) JSON object that students can use to store whatever they want. For a typical project all "custom" columns will be null. They will be used only when the project has some feature that requires extra information (not available on the provided schema).
  - ***"created_at"*** *(optional) - date and time the record was created - this column is automatically managed by Laravel.*

- *"**updated_at**" (optional) - date and time when the record was last changed - this column is automatically managed by Laravel.*
- *"**deleted_at**" (optional) - date and time the record was **deleted** (with soft delete) - this column is automatically managed by Laravel. If a table includes the "deleted_at" column, it means that the associated Eloquent model can support soft deletes.*

USERS

Table with the application's registered users - customers, employees, and administrators.

Users is a superclass of the customers entity, which means that some users are customers. Customer's data is distributed between 2 tables: users and customers. The users table will have all the data common to all users and the customers table will have all the data specific (exclusive) to customers.

- "**id**" - primary key - automatic integer.
- "**name**" - user name.
- "**email**" - the user's email address. The email must be unique. The email also serves as an authorization credential (together with the password).
- "**password**" - *hash of* the user's password.
- "**type**" - type of user. C (customer); E (employee) and A (administrator).
- "**blocked**" - Boolean (1 or 0) indicating whether the user is blocked or not. A blocked user should not be able to enter the application (authorization should fail).
- "**photo_filename**" (optional) - the ("relative") name of the file containing the user's photo (or avatar).

The next fields are managed automatically by Laravel and are not normally displayed in the application:

- *"**email_verified_at**" (optional) - date on which the user's email was verified - this column is managed by Laravel's authentication system.*
- *"**remember_token**" (optional) - to implement the "remember me" functionality - this column is managed by Laravel.*

CUSTOMERS

Table with the CineMagic registered customers. Customers entity is a subclass of users entity, which means that customers are also users. Customer's data is distributed between 2 tables: users and customers. Users table will have data that is common to all users (for example, the customer's name is in the users table) and the customers table will have the specific (unique) customer data.

- "**id**" - primary key - integer value **equal to** the user's "id". The "id" of the customers table is not filled in automatically, it will always have the same value as the user's "id" (the customer

is a user). In this case, the customer's "id" is also the foreign key responsible for the relationship between the customers table and the users table.

Note: when inserting a customer, it is always necessary to first insert a record in the "users" table and then the equivalent record in the "customers" table. The opposite should happen if you need to delete a customer: first delete the record in the "customers" table and then the equivalent record in the "users" table.

- "**nif**" (optional) - the customer's NIF (Tax Identification Number). The NIF (when filled in) must always have 9 digits - for example: 148928093. The NIF in the customers table will be used to pre-fill the NIF on customer's purchases - it corresponds to the default value of the NIF.
- "**payment_type**" (optional) - Payment type used by default on customer's purchases. Accepted values are: "VISA" (Visa credit card); "PAYPAL" (PayPal); and "MBWAY" (MBWay).

  If the payment type is filled in (value is not null) then the payment reference should also filled in (value is not null). If the payment type is null, then payment reference should also be null.
- "**payment_ref**" (optional) - Payment reference used by default on customer's purchases. If the payment type is "VISA", then the value of the payment reference corresponds to the credit card number and must be 16 digits long. If the payment type is "PAYPAL", then the value of the payment reference corresponds to the email address of the PayPal account. If the payment type is "MBWAY" then the payment reference value must be a mobile phone number in Portugal (9 digits, with the first digit always being 9).

  If the payment type is filled in (value is not null) then the payment reference should also filled in (value is not null). If the payment type is null, then payment reference should also be null.

CONFIGURATION

Table with the application's configuration parameters. This table should have **one and only one** row.

- "**id**" - primary key - automatic integer.

  Note: as the table always has one row and one row only, this field is irrelevant. However, defining a primary key makes it easier to integrate the table with Laravel.
- "**ticket_price**" - current price of each ticket for the public (without any discount).
- "**customer_ticket_discount**" –registered customer's discount that is applied for each ticket.

  Ticket price for a registered customer = "ticket_price" – "customer_ticket_discount"

GENRES

Table of movie genres.

- "**code**" - primary key - genre code (examples: DRAMA; COMEDY; SCIFI; etc.)
- "**name**" - gender name.

MOVIES

Table with movies. A movie is considered to be showing if it is associated with a session taking place today or at a later date.

- "**id**" - primary key - automatic integer.
- "**title**" - Title of the movie.
- "**gender_code**" - Genre of the movie. Foreign key (related to the genres table).
- "**year**" – Year of the movie.
- "**poster_filename**" (optional) - name of the file (relative name) containing the image of the movie poster. The poster image must be stored in the application's "*storage"* folder.
- "**synopsis**" - Text with the synopsis (summary) of the movie.
- "**trailer_url**" (optional) - absolute URL with the address of the video with the trailer for this movie. Unlike the poster, the trailer will always be external to the application (the application does not save videos with trailers).

THEATERS

Table with the theaters owned by the company "CineMagic".

- "**id**" - primary key - automatic integer.
- "**name**" - name of the theater.
- "**photo_filename**" (optional) - name of the file (relative name) containing an image with a photo of the theater. This image must be stored in the application's "*storage"* folder.

SEATS

Table with the seats in a theater. Each seat in a theater corresponds to a record in this table. Each ticket sold is associated to one, and only one seat for a specific movie screening. A seat in a movie screening is considered empty if there are no tickets associated with that seat for that movie screening.

- "**id**" - primary key - automatic integer.
- "**theater_id**" - foreign key (related to the theaters table) - identifies the theater to which this seat belongs to.
- "**row**" - row of seats. Usually, the row is represented by a letter (e.g. D)
- "**seat_number**" – number (position) of the seat in the row.
    - Seats are usually represented by a letter and a number, for example: "D3", where "D" is the seat row and 3 is the seat_number.

SCREENINGS

Table with the screening sessions. One screening session corresponds to a session where a movie is viewed in a specific theater on a given date and time (start_time).

- "**id**" - primary key - automatic integer.
- "**movie_id**" - foreign key (related to the movies table) - identifies the movie associated with the screening.
- "**theater_id**" - foreign key (related to the theaters table) - identifies the theater associated with the movie screening (where the movie will be viewed).
- "**date**" - the day of the screening (when the movie is viewed).
- "**start_time**" - time (e.g. 20:30) at which the movie screening begins (the movie starts).

PURCHASES

Table with purchases. Only purchases that are completed and paid for (payment process successfully completed) are stored on the "purchases" table. Each purchase includes one or more tickets. It is not possible to buy tickets (even one single ticket) without registering a purchase.

Purchases maintain information about the customer, either a public buyer or a registered customer, payment, receipt, and tickets.

Note: while the shopping cart is being built, no purchase *should be stored on the database. Only after the buyer confirms the shopping cart and the payment transaction is executed and considered valid, will the application store the purchase and generate the tickets - the shopping cart is kept in the server session and not in the database.*

- "**id**" - primary key - automatic integer.
- "**customer_id**" (optional) - foreign key (related to the customers table) - identifies the registered customer of the purchase. When the value of "customer_id" is null, it means that the customer is not a registered customer – it is a public buyer (anonymous user).
- "**date**" - date of purchase - you only need the day (the time doesn't matter).
- "**total_price**" - total price of the tickets associated to the purchase (number of tickets multiplied by the price of each ticket – as defined currently in the "configuration" table). The price is different for public buyer or registered customers.
- "**customer_name**" - Customer's name. When a registered customer is completing/confirming the purchase, this field should default to the customer's name. When a public buyer is completing/confirming the purchase, this field is initially empty. Either way, the buyer is free to choose any name.
- "**customer_email**" - Customer's email. When a registered customer is completing/confirming the purchase, this field should default to the customer's email. When a public buyer is

completing/confirming the purchase, this field is initially empty. Either way, the buyer is free to choose any email. This email will be used to receive the receipt and/or tockets.

- "**nif**" (optional) - Customer's NIF (Tax Identification Number). When filled in, NIF should always be 9 digits long - for example: 148928093. When a registered customer is completing/confirming the purchase, this field should default to the customer's NIF (if the customer himself has specified the NIF value). When a public buyer is completing/confirming the purchase, this field is initially empty. Either way, the buyer is free to choose any NIF.
- "**payment_type**" - Type of payment used to pay the purchase. Accepted values are: "VISA" (Visa credit card); "PAYPAL" (Paypal); and "MBWAY" (MBWay). When a registered customer is completing/confirming the purchase, this field should default to the customer's payment type (if the customer himself has specified a payment type). When a public buyer is completing/confirming the purchase, this field is initially empty. Either way, the buyer is free to choose any valid payment type.
- "**payment_ref**" - Payment reference used to pay the purchase. If the payment type is "VISA", then the value of the payment reference corresponds to the credit card number and must be 16 digits long. If the payment type is "PAYPAL", then the value of the payment reference corresponds to the email address of the PayPal account. If the payment type is "MBWAY" then the payment reference value must be a mobile phone number in Portugal (9 digits, with the first digit always being 9). When a registered customer is completing/confirming the purchase, this field should default to the customer's payment reference (if the customer himself has specified a payment reference). When a public buyer is completing/confirming the purchase, this field is initially empty. Either way, the buyer is free to choose any valid payment reference.
- "**receipt_pdf_filename**" (optional) - name of the file (relative name) containing a PDF with the receipt of the purchase (if the application implements this feature). If the value is null, the PDF receipt has not been generated. Once the receipt PDF has been generated, it should be stored in the application's *storage* folder for future reference - the PDF should only be generated once (the document is considered immutable).

TICKETS

Table with tickets. Each completed and paid purchase will generate a set of tickets, each ticket corresponding to a specific seat in a specific movie screening.

- "**id**" - primary key - automatic integer.
- "**screening_id**" - foreign key (related to the screenings table) - identifies the movie screening associated with the ticket.
- "**seat_id**" - foreign key (related to the seats table) - identifies the seat associated with the ticket.

- "**purchase_id**" - foreign key (related to the purchases table) - identifies the purchase associated with the ticket. With this relationship it is possible to identify all tickets associated with a given purchase.
- "**price**" - price of the ticket. Since ticket prices can vary over time, this field lets you know the actual price of the ticket when it was sold (which may not be the same as the current price).
- "**qrcode_url**" (optional) – URL associated to the QR Code of the ticket (if the application implements this feature). Ticket information can be confirmed by the employee using this URL (either entering the URL manually or scanning the QR Code).

  Note that the "qrcode_url" must be a full URL and not easily replicated (should include a long random unique string) to avoid ticket theft.
- "**status**" - indicates the ticket status. Value can be:
  - "**valid**" when the ticket is valid and was not used yet.
  - "**invalid**" when the ticket is invalid because it has been used to enter the corresponding screening session, or because it was invalidated manually by an employee or administrator (e.g. a child trying to view an adult movie), or by the application (e.g. the application might invalidate all tickets that were not used after the screening has ended).

PAYMENT SIMULATION

The payment process is simulated using the Payment class (provided in the project resources) which should be placed in the "`app/services`" folder (this folder must be manually created). This class includes 3 static methods that return *true* if the payment is considered valid and *false* if the payment is considered invalid. The methods of the class are as follows:

- **payWithVisa** ($card_number, $cvc_code) - Simulates payment with a visa card;
- **payWithPaypal** ($emailAddress) - Simulates payment with Paypal;
- **payWithMBway** ($phone_number) - Simulates payment with MBWay;

PROJECT ARCHITECTURE AND IMPLEMENTATION RULES

The project must be implemented with the framework Laravel, version 11 or newer, and use a MySQL database (with the provided structure). The implemented web application must use a server-based architecture, which means that all the content is created by the server – this means that JavaScript is not required to develop any part of the application, and JavaScript based frameworks like Vue.js, ReactJS or similar, cannot be used during development. JavaScript is allowed, but only as a complement for partial implementation details that require some interaction within the browser (e.g. show and hide elements, implement some visual effects, create popup dialog boxes).

The project **can use external packages** (not included with default Laravel installation). These are some allowed/recommended packages:

- Laravel Telescope (https://laravel.com/docs/telescope) and Laravel Telescope Toolbar (https://github.com/fruitcake/laravel-telescope-toolbar) for debugging.
- A Laravel package to generate PDF files – at students' discretion.
- A Laravel package to generate QR Codes – at students' discretion.
- A Laravel package to create charts – at students' discretion.

There are some restrictions for the external packages to use. The use of the **following packages or types of packages is forbidden**:

- Laravel Nova (https://nova.laravel.com).
- Filament (https://filamentphp.com).
- Any type of package or tool that automatically generates views or any other content.

If it is not clear whether a package is allowed or forbidden, confirm it with the theory teacher (marco.monteiro@ipleiria.pt).

DELIVERY

Project delivery must include 2 files (the project and group report) per group and 1 individual report file per student. For example, if the group has 3 students, 5 files must be delivered (2 files related to the group and 3 individual report files) – one student delivers 3 files (the project, the group report and his own individual report), and the remaining 2 students deliver only their own individual reports. Files:

- prj_GG.zip – zip file that includes a copy of all the project's folders and files (source code and other files), except the folders "vendor", "node_modules", "database", "storage" and ".git" (".git", when exists is typically an hidden folder).
- grp_report_GG.xlsx – the group report file - includes group elements identification and information about the project implementation. The model for the report will be provided.
- individual_report_NNNNNNN.xlsx – the individual report file - includes self-assessment and peer assessment. The model for the report will be provided.

Note: replace GG with your group number and NNNNNNN with your student number

EVALUATION

The project's evaluation criteria are defined by the functionality groups identified above. The weights of the functionality groups are as follows:

Number	Weight	Functionality group
1	15%	Authentication, profile and user management
2	15%	Movies on show
3	15%	Buying tickets
4	10%	Choice of seats
5	10%	History, Purchases, Receipts and Tickets
6	10%	Screening sessions access control
7	15%	Business management
8	10%	Statistics

The assessment of each group of features depends on the quantity and quality of features implemented, the integration of the features into the application and the usability and correctness (correct operation) of the features. Also taken into account will be the layout and visual appearance (consistent appearance and well-structured layout) and some non-functional requirements that are transversal to the entire application, such as (among others):

- The implementation of the application must follow the MVC architectural pattern (*Model- View- Controller*) and the structure, design patterns and good practices defined in the Laravel Framework.
- The methods (GET, POST, PUT, PATCH, DELETE) and URLs of the routes should follow the best practices and conventions of web applications.
- The application should use the appropriate Laravel features and components to solve the application's various challenges. For example, use Eloquent whenever possible (instead of the DB class) to access the database; use "Form Request" to implement validations on the server, use the authentication system provided by Laravel (instead of implementing your own), use the hash method provided by Laravel to store passwords, use Laravel *policies for* authorization, etc.
- The application should follow the DRY principle (*Dont't Repeat Yourself*), i.e. it should avoid repeating code whenever possible, without breaking Laravel's best practices. For example, use subviews or view components whenever there are sections of HTML that are repeated in several views or add methods to models that return a certain set of data that will be used in several controllers.
- The application should have the best performance possible. Fulfilling this requirement depends largely on the interaction with the database. As far as database access performance is concerned, two main principles should guide the implementation of the application:

reducing the number of SQL commands that are executed on the database as much as possible and ensuring that each query to the database brings up only the necessary information (only the columns and rows that are strictly necessary).

The application should also take into account other features that improve the application's overall performance, such as: using *queues* for heavier operations; using caching systems properly (research caching in Laravel); ensuring that the resources returned by the server are as small as possible (reducing the size of HTTP responses), etc.

- The application must guarantee maximum security and privacy for its users. The application must guarantee that all the operations and resources it makes available are only accessible to those who are authorized to do so. Users' private information must always be protected from unauthorized access.

  The implementation of the mechanisms that guarantee security, privacy and authorization must be done in accordance with Laravel's good practices and using the appropriate Laravel features and components.

  > *Note: one of the main features that make web applications more secure and better protect privacy is the use of a secure protocol for communication between the client and the server - the use of HTTPS instead of HTTP. As in this project we are only dealing with the application layer of the system and not the communication layer, it is NOT necessary to use HTTPS.*