League of teams and players

It is necessary to implement a web application with the Laravel framework following the model-vistacontroller architecture seen in class, which allows the management of the teams and players of a sports league.

Communication between classes and methods must be done with objects from the application data model. No code that is not properly encapsulated will be accepted.

Methods and classes must be properly documented. For control methods, documentation is required their behavior in all cases. The application must have an index (home) entry page with app information and thatcontains text and images.

All pages must have a homogeneous format. They must contain a heading and footer of page.

They must also contain a common menu with the options (Home, Manage teams, Manage players).

The definition of team and player fields is specified later.

The team-player relationship is 1xn. Therefore, a team can have 0 or more players and one player may be at 0 or a team.

The data of all forms must be validated, both on the client side and on the server side.

Create the model with migrations and seeders to generate a sufficient set of test data.

Create one controller for teams and another for players.

Separate the views of teams and players into different directories.

Create a suitable style set and give a correct format to the pages and forms.

In this initial version you do not need to define users for the application, nor control sessions, login/logout or roles user. Yes, you need to define a database user for the application.

It is a requirement to use the appropriate mechanisms (exceptions or others) in order to adequately treat all the cases that can be presented and accurately inform the user of the results of the actions.

Manage teams page

The page contains an Add team button, which links to a page to add a team.

The button below shows the table of all the teams and the number of teams shown. Every line it will contain the name, the stadium, the team's membership number and a button to edit the team (nave on the page team editing) and another to delete it. This last action requires confirmation from the user.

If the team has players, the action should not be allowed, as players must be removed from the team before erasing it. It is necessary to always show the feedbacks to the user indicating whether the actions have been carried out or not and why.

Add teams page

Contains a form with all the fields needed to enter a team data (Team):

- id (disabled) (autoincremental)
- name (string: unique)
- Stadium
- numMembers (int)
- budget

Buttons: New Team Button and Cancel button (return to the Manage teams screen)

The id of the team must be disabled and has no use in this form, since the id of the team is self-incremental and therefore managed by the database management system.

Edit team page

Contains a form with all the fields needed to enter a team data (Team):

- id (disabled) (autoincremental)
- name (string: unique)
- stadium (string)
- numMembers (int)
- budget (double)

New Team Button and Cancel button (return to the Manage teams screen)

The team id must be disabled, but it must contain the id of the team being edited.

Following the form, you must show in a table the list of players registered in the team, the number of players shown and a button to register a new player on the team (enrol).

Each row will have a button for unsubscribe the player from the team.

Enrol player to team page

We will navigate to this page from that of team editing when the button of registering a player is pressed the team that is being edited. It will show a table with name, surname and team to which each player in the league is enrolled, and a button for

Register it for the team in question. If the player is already registered with another team, they will inform which and You will ask for confirmation to unsubscribe from your current team and make the new registration. Needs to ensure the atomicity of both tasks.

Finally, the corresponding feedback must be given to the user.

Manage players page

The page contains an Add player button, which links to a page to add a player.

The button below shows the table of all players and the number of players shown. Every line it will contain the name, surname, position and button to edit the team (nave on the team edit page) and another to erase it.

This last action requires confirmation from the user. If the player It belongs to a team, the action must not be allowed, as it must be removed from the team before erasing it.

It is necessary to always show the feedbacks to the user indicating whether the actions have been carried out or not and why.

Add player page

Contains a form with all the fields necessary to enter a team data (Player):

- id (disabled) (autoincremental)
- name (string)
- surname (string)
- position (string)
- salary (double)

New Player Add Button

Button to Cancel (return to the Manage players screen)

The player id must be disabled and has no use in this form, since the player id is self-incremental and therefore managed by the database management system.

Edit player page

Contains a form with all the fields needed to enter a player's data (Player):

- id (disabled) (autoincremental)
- name (string)
- surname (string)
- position (string)
- salary (double)

New Player Add Button

Button to Cancel (return to the Manage players screen)

The player id must be disabled, but it must contain the player id being edited.

With this information, we will begin the project. Here are the first steps in order:

- Laravel Project creation
  - Database project creation
    - Migration – To create the tables
    - Seeders - To fill the tables
  - Model
    - Define data structure

Our objective in these first steps is to get the data and define what we will work with.

Database creation

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
CREATE DATABASE leaguedb DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci;
```

[Edit inline] [Edit] [Create PHP code]

Database user creation

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0008 seconds.)

```
CREATE USER 'leagueusr'@'localhost' IDENTIFIED BY 'leaguepass';
```

[Edit inline] [Edit] [Create PHP code]

Granting all privileges on the database to the user

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
GRANT ALL PRIVILEGES ON *.* TO 'leagueusr'@'localhost';
```

[Edit inline] [Edit] [Create PHP code]

Checking privileges for the database and user created

Users having access to "leaguedb"

	User name	Host name	Type	Privileges	Grant	Action	
☐	leagueusr	localhost	global	ALL PRIVILEGES	No	Edit privileges	Export
☐	root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit privileges	Export
☐	root	::1	global	ALL PRIVILEGES	Yes	Edit privileges	Export
☐	root	localhost	global	ALL PRIVILEGES	Yes	Edit privileges	Export

Migrations

First we have to create the teams table, and after that the players table will reference teams

php artisan make:migration create_teams_table

```
aymane@dev-aymane:~/Documents/CODE/M07/UF4/Entregues_UF4_M07/PT_UF4_M07
_23-24$ php artisan make:migration create_teams_table

  INFO  Migration [database/migrations/2024_02_28_180121_create_teams_
table.php] created successfully.
```

First we have to create the teams table, and after that the players table will reference teams

php artisan make:migration create_players_table

```
aymane@dev-aymane:~/Documents/CODE/M07/UF4/Entregues_UF4_M07/PT_UF4_M07_23-24
$ php artisan make:migration create_players_table --create=PT1_UF4_M07_Aymane

  INFO  Migration [database/migrations/2024_02_28_170447_create_players_tabl
e.php] created successfully.
```

After that we can run the migrations

php artisan migrate

```
  INFO  Running migrations.

  2019_12_14_000001_create_personal_access_tokens_table ......... 34ms DONE
  2024_02_28_180121_create_teams_table ........................... 23ms DONE
  2024_02_28_180301_create_players_table ......................... 62ms DONE
```

Seeder - Adding test data to the database

The seeder will create as Players and Teams based on the specifications set at the factory

```
aymane@dev-aymane:~/Documents/CODE/M07/UF4/Entregues_UF4_M07/PT_UF4_M07_23-24
$ php artisan db:seed

  INFO  Seeding database.
```

We can verify at the database that the tables have been populated

	id	first_name	last_name	salary	position	team_id
Edit Copy Delete	1	Bianka	Ryan	6328.28	Small Forward	1
Edit Copy Delete	2	Icie	Boehm	5867.28	Center	1
Edit Copy Delete	3	Mazie	Mertz	6742.56	Small Forward	1
Edit Copy Delete	4	Moshe	Herman	5057.13	Center	1
Edit Copy Delete	5	Giovanny	Gislason	8938.49	Center	1
Edit Copy Delete	6	Josephine	Hayes	5178.59	Small Forward	1
Edit Copy Delete	7	Muriel	Hintz	9781.75	Small Forward	1
Edit Copy Delete	8	Delfina	Jast	7030.71	Small Forward	2
Edit Copy Delete	9	Ivory	Bosco	8335.07	Shooting Guard	2
Edit Copy Delete	10	Josephine	Kemmer	8448.04	Center	2
Edit Copy Delete	11	Ima	Gerlach	5766.17	Center	2
Edit Copy Delete	12	Abdul	Hoeger	5513.08	Center	2
Edit Copy Delete	13	Marcelino	Klocko	6314.63	Center	2
Edit Copy Delete	14	Kaylie	Olson	9839.19	Point Guard	2
Edit Copy Delete	15	Demarcus	Gibson	7612.5	Small Forward	3
Edit Copy Delete	16	Pearline	Adams	6007.59	Shooting Guard	3
Edit Copy Delete	17	Emelia	Olson	7463.39	Center	3
Edit Copy Delete	18	Oliver	Dickinson	8798.71	Power Forward	3
Edit Copy Delete	19	Lane	Jacobi	9869.51	Point Guard	3
Edit Copy Delete	20	Samir	Yost	9971.69	Power Forward	3
Edit Copy Delete	21	Sanford	Zulauf	9816.59	Point Guard	3
Edit Copy Delete	22	Bernita	Hudson	8650.65	Shooting Guard	4
Edit Copy Delete	23	Kareem	Stark	8919.61	Shooting Guard	4
Edit Copy Delete	24	Angela	Lind	7559.57	Shooting Guard	4
Edit Copy Delete	25	Autumn	Hudson	9218.4	Shooting Guard	4

	id	name	stadium	numMembers	budget
Edit Copy Delete	1	Kubton Fuga	Bianka Auer DVM	numMembers 0	78441.08
Edit Copy Delete	2	North Eveline Corporis	Adam Grimes	numMembers 7	59807.16
Edit Copy Delete	3	North Aliviafurt Quae	Dr. Wilbert Wisoky	numMembers 9	99328.89
Edit Copy Delete	4	Abbottville Asperiores	Bell Ziemann	numMembers 3	84147.54
Edit Copy Delete	5	West Katarinabury Sed	Mr. Alexander Mraz DVM	numMembers 3	84320.28
Edit Copy Delete	6	Melanyport Cupiditate	Prof. Giovanna Lebsack	numMembers 1	83610.83
Edit Copy Delete	7	Michaelbury Dolores	Prof. Citlalli Hamill	numMembers 3	97805.04
Edit Copy Delete	8	Lake Gregoryborough Omnis	Maymie Schumm	numMembers 6	52904.72
Edit Copy Delete	9	Lake Ariannachester Autem	Alysha Balistreri	numMembers 0	78851.62
Edit Copy Delete	10	Elwinmouth Rerum	Ezra Ullrich	numMembers 9	60931.4
Edit Copy Delete	11	Juniusville Ut	Andres Thompson II	numMembers 0	67365.03
Edit Copy Delete	12	New Cristinaland Eos	Dr. Bart Kris	numMembers 3	99699.72
Edit Copy Delete	13	Marvinport Voluptas	Dr. Enola Smitham DDS	numMembers 0	71091.87
Edit Copy Delete	14	East Juston Alias	Stephany Gibson III	numMembers 2	94259.34
Edit Copy Delete	15	West Eleanoreberg Et	Shanelle Upton	numMembers 1	81311.34

We can re-do the seeding aswell using the command:

php artisan migrate:refresh –seed

Validating the forms in 4 steps

This approach adheres to Laravel's "thin controller, fat model" best practice by keeping the controller methods clean and focusing on their primary responsibilities.

Step 1 – Create the middleware

php artisan make:middleware ValidateTeamForm

```
aymane@dev-aymane:~/Documents/CODE/M07/UF4/Entregues_UF4_M07/PT_UF4_M07_23-24$ php artisan make:middleware ValidateTeamForm

  INFO  Middleware [app/Http/Middleware/ValidateTeamForm.php] created successfully.
```

Step 2 – Populate the middleware with the validation (for instance regex)

```php
class ValidateTeamForm
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        $validator = Validator::make($request->all(), [
            "name" => "required|min:2|regex:/^[\pL'\s.]+$/u",
            "stadium" => "required|min:2|regex:/^[\pL'\s.]+$/u",
            "numMembers" => 'required|numeric|min:1',
            "budget" => "numeric|min:0",
        ]);

        if ($validator->fails()) {
            return redirect()->back()->withErrors($validator)->withInput();
        }

        return $next($request);
    }
}
```

Step 3 - Add it to **app/Http/Kernel.php**

```php
    protected $middlewareAliases = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
        'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRe
        'signed' => \App\Http\Middleware\ValidateSignature::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
        //Custom middleware added for team validation
        'validate.team.form' => \App\Http\Middleware\ValidateTeamForm::class,

    ];
```

Step 4 - Use it directly as a filter at web.php, for example:

```php
Route::post '/team/update'  TeamController::class 'updateTeam'  ->name 'team.update'  -
>middleware 'validate.team.form'  // Apply middleware for validation on team add
```

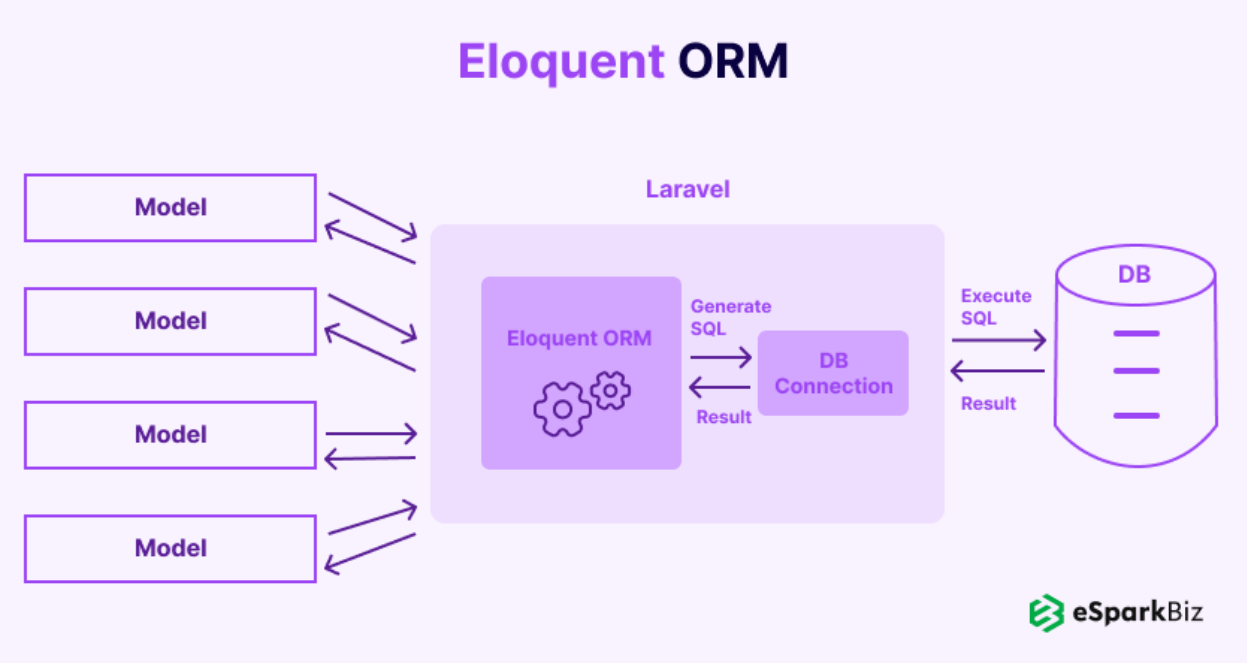
The same steps are applied for the players middleware

Updating the Request types – Separation of concerns
Git 2a395a4 vs Git 2b17a57 - Avoiding the use of Route::match

The use of ::match for routing both GET and POST requests to the same controller function was not appropriate for handling distinct actions like showing a form (GET) and processing form submissions (POST) separately.

Each request type should be directed to its specific function to maintain clear separation of concerns and ensure that each function is responsible for a single, clear purpose, using Eloquent ORM for the remaining operations.

This separation of concerns is specially important when we consider that Eloquent as a the tool that laravel uses to Map the data. Eloquent acts as a bridge between your application's database and its business logic.


Eloquent ORM
Laravel
Model
Model
Model
Model
Eloquent ORM
Generate SQL
Result
DB Connection
Execute SQL
Result
DB
eSparkBiz