Virtuagym’s Financial Dev Test

Technologies used:

- Laravel Framework,
- Docker Container with Sail,
- Unit Testing Laravel

This Laravel application is designed to manage invoices and track users checking in to fitness clubs. It includes a system of RESTful API endpoints to perform CRUD operations on invoices, with each invoice having a date, status, description, amount, and invoice lines. The invoice state can be outstanding, paid, or void, and each invoice can have multiple lines, each containing an amount and a description.

In addition, the application includes a membership system, where each membership is associated with a user and can be active or canceled. Memberships have a set amount of credits and a state, including a start date and end date. Users can check-in to a club, with their membership losing one credit when they do. Users cannot check-in if their membership is canceled, if they have no credits available, or if the membership's end date is reached.

Finally, the application includes an endpoint allowing users to check-in to a fitness club via turnstile. When a user checks in, an invoice line is created for the month's invoice, and if the invoice does not exist, it gets created

Structure.

1. InvoiceController - This controller can handle CRUD operations on the invoice model. It can include methods such as index(), create(), store(), show(), edit(), update(), and destroy(). The model associated with the invoice controller is the Invoice model, which will have properties such as date, status, description, amount, and invoice lines.
2. MembershipController - This controller can handle the membership system, which is associated with the user model.
3. UserController - This controller can handle check-ins at the fitness clubs. It can include a method called checkin(), which will create an invoice line for the current month's invoice if it doesn't already exist.

By separating the code into these three controllers, we can improve the code's organization and make it easier to maintain and scale the application in the future. Additionally, we can have three separate models, Invoice, Membership, and User models, each with their specific properties and methods. This separation of concerns is a common practice in software development and follows the Model-View-Controller (MVC) architectural pattern.

Invoice Controller

This is a PHP Laravel API controller that handles CRUD (create, read, update, and delete) operations for invoices and invoice lines.

The controller defines several methods, each corresponding to an HTTP verb and API endpoint:

- **index()** retrieves a list of all invoices with their associated invoice lines and returns a JSON response with the data.
- **store(Request $request)** creates a new invoice with the specified data and returns a JSON response with the data.
- **storeByInvoiceLine($id)** creates a new invoice line associated with a specified invoice, creating the invoice if necessary, and returns a JSON response with the data.
- **show(string $id)** retrieves a specific invoice by ID with its associated invoice lines and returns a JSON response with the data.
- **update(Request $request, string $id)** updates a specific invoice with the specified data and returns a JSON response with the updated data.
- **destroy(string $id)** deletes a specific invoice by ID and returns a JSON response with the deleted data.

The controller makes use of Laravel's Eloquent ORM to interact with the database and Carbon library to work with dates. It also uses Laravel's validation system to validate incoming request data.

The controller returns JSON responses with a success status, a message, and the requested or modified data.

Membership Controller

This is a PHP code for a controller class named **MembershipController** in the **Api** namespace. It extends the **Controller** class and defines methods to handle various API requests related to memberships.

The **index()** method retrieves all memberships from the database and returns them as a JSON response. If no memberships are found, it returns a failure response.

The **store()** method creates a new membership with the provided data, validates the request data, and returns the new membership as a JSON response. If the user already has a membership, it returns a failure response.

The **show()** method retrieves a membership with the specified ID and returns it as a JSON response. If no membership is found with the given ID, it returns a failure response.

The **disable()** method updates the status of a membership with the provided status data and returns the updated membership as a JSON response.

The **cancelMembership()** method cancels the membership if the current date is greater than or equal to the membership's end date. It returns the updated membership as a JSON response.

The **updateAmount()** method subtracts 1 from the amount of a membership and returns the updated membership as a JSON response.

The **destroy()** method deletes a membership with the specified ID and returns it as a JSON response.

User Controller

This is a PHP class named UserController, which is responsible for checking in a user with a valid membership. It has a single public method named CheckIn, which takes a user ID as input and returns a JSON response indicating success or failure.

The method starts by creating instances of the InvoiceController and MembershipController classes. Then, it finds the user with the given ID using the User model.

If the user doesn't have a membership, the method returns a JSON response with the "success" field set to false and the "message" field set to "Membership not found."

If the user's membership has ended, the method returns a JSON response with the "success" field set to false and the "message" field set to "Membership date is due."

If the user's membership has been canceled, the method returns a JSON response with the "success" field set to false and the "message" field set to "Membership is canceled."

If the user has no credits remaining, the method returns a JSON response with the "success" field set to false and the "message" field set to "No more credits."

If none of the above conditions are true, the method creates a new invoice using the InvoiceController and updates the user's membership amount using the MembershipController.

The method always returns a JSON response, whether the check-in is successful or not. If the check-in is successful, the "success" field is set to true and there is no "message" field.

Test Classes

1- MembershipTest Class

This is a unit test class for the **MembershipController** in a Laravel application. The class has five test methods, **testIndex()**, **testStore()**, **testShow()**, **testUpdateAmount()**, and **testDestroy()**.

The **testIndex()** method sends a GET request to the route named "memberships" and asserts that the response has a status code of 200.

The **testStore()** method creates a new membership with the given data using a POST request to the "memberships.store" route, and asserts that the response has a status code of 200, success is true, and the response data matches the input data. It also checks that the membership has been added to the database.

The **testShow()** method creates a membership record in the database, sends a GET request to the show route for the membership, and asserts that the response has a status code of 200, success is true, and the response data matches the membership record. It also tests the case where a non-existent membership is requested, and asserts that the response has a status code of 404.

The **testUpdateAmount()** method creates a new membership for the user, sends a PUT request to the route "memberships.update_amount" for the created membership, and asserts that the response has a status code of 200. It also checks that the membership amount has been updated by 1.

The **testDestroy()** method creates a new membership for the user, sends a DELETE request to the "memberships.destroy" route for the created membership, and asserts that the response has a status code of 200. It also checks that the membership has been deleted from the database.

These unit tests are useful for ensuring that the **Membership** model is working correctly and that the application routes are returning the expected data.

2- InvoiceTest Class

This is a PHP test class for an InvoiceController that tests various CRUD operations on the Invoice model. Here's a summary of each test method:

1. **testIndexInvoice**: Tests the **index** method of the controller by creating a new Invoice, saving it to the database, and calling the **index** method. The test asserts that the response has a status code of 200.
2. **testStoreInvoice**: Tests the **store** method of the controller by creating a fake request with required fields, creating a new instance of the controller, and calling the **store** method with the fake request object. The test asserts that the response contains the expected data and that an Invoice object was created with the expected data.
3. **testShowInvoice**: Tests the **show** method of the controller by creating an Invoice, two InvoiceLines, and attaching the InvoiceLines to the Invoice. The test sends a GET request to the **show** endpoint for the Invoice and asserts that the response has a status code of 200 and that the response contains the expected data. The test also asserts that a non-existent Invoice returns a 404 response.
4. **testUpdateInvoice**: Tests the **update** method of the controller by creating an Invoice, making a PUT request to update the Invoice status, and asserting that the response contains the expected data.

3- UserTest Class

This is a unit test class for the **UserController** class in a Laravel application. The purpose of this test class is to test the **checkin()** method of the **UserController** class. The **checkin()** method checks whether a user can check-in to a gym or not based on the user's membership status and credits.

The test class contains five test methods, each testing a different scenario:

1. **testCheckInWithValidMembership()**: This test checks whether a user with a valid membership can check-in to the gym or not. The test creates a user with a valid membership, makes a POST request to the **checkin()** method, and checks whether the response contains the expected JSON data. The test also checks whether an invoice was created and the user's membership amount was updated or not.
2. **testCheckInWithExpiredMembership()**: This test checks whether a user with an expired membership can check-in to the gym or not. The test creates a user with an expired membership, makes a POST request to the **checkin()** method, and checks whether the response contains the expected JSON data. The test also checks whether no invoice was created and the user's membership amount was not updated.
3. **testCheckInWithNoCredits()**: This test checks whether a user with no credits can check-in to the gym or not. The test creates a user with no credits, makes a POST request to the **checkin()** method, and checks whether the response contains the expected JSON data. The test also checks whether no invoice was created and the user's membership amount was not updated.
4. **testCheckInWithCanceledMembership()**: This test checks whether a user with a canceled membership can check-in to the gym or not. The test creates a user with a

canceled membership, makes a POST request to the **checkin()** method, and checks whether the response contains the expected JSON data. The test also checks whether no invoice was created and the user's membership amount was not updated.

5. **testCheckInWithoutMembership()**: This test checks whether a user without a membership can check-in to the gym or not. The test creates a user without a membership, makes a POST request to the **checkin()** method, and checks whether the response contains the expected JSON data. The test does not check for any invoice creation or membership amount update in this scenario.

The **use DatabaseTransactions** statement at the beginning of the class indicates that the tests will run inside a database transaction, and the transaction will be rolled back after each test. This ensures that the database remains in a consistent state throughout the testing process.

Overall, these tests cover the basic CRUD operations on the Invoice model and ensure that the controller methods are working as expected.

Routes

This is a PHP file containing the API routes for a Laravel application. It registers several endpoints for different controllers:

- UserController: **POST** request to **/user/{id}/checkin**
- MembershipController: **GET** request to **/memberships**, **POST** request to **/memberships/store**, **GET** request to **/memberships/{id}**, **PUT** request to **/memberships/{id}/disable**, **PUT** request to **/memberships/{id}/updateAmount**, **DELETE** request to **/memberships/{id}/destroy**
- InvoiceController: **GET** request to **/invoices**, **POST** request to **/invoices/store**, **GET** request to **/invoices/{id}**, **PUT** request to **/invoices/{id}/update**, **POST** request to **/invoices/{id}/storeInvoiceLine**, **DELETE** request to **/invoices/{id}/destroy**

Each endpoint is associated with a specific method within the corresponding controller. The **name()** method is used to assign a name to each route, which can be used later to generate URLs. These routes are all protected by the "api" middleware group, which adds some extra security and authentication to the endpoints.