Project documentation

Dashboard Section

1. **Admin Controller**
   1. **all(Request $request) Method**

🔹 Overview  
This method retrieves a list of admin users with role ID 7. It supports optional pagination, based on the paginate parameter provided in the request.

🔹 **Key Functionality**

1-**Check for Pagination Requirement**

* Verifies if the request includes a paginate parameter.
* If paginate is 1: Fetch paginated admin data.
* If paginate is 0: Fetch all admin records without pagination.
* If paginate is not provided: Defaults to pagination with 20 items per page.

2-**Fetch Admin Records**

* Retrieves users from the users table where role\_id = 7.
* Sorts the results by latest (most recently created admins come first).
* Uses Laravel’s paginate() or get() method depending on the condition.

3-**Format Response with Resources**

* Wraps the result in EmployeeResource::collection() for consistent API output.
* If paginated, constructs a pagination array including:
  + total items
  + per\_page count
  + current\_page
  + total\_pages

4-**Return Standardized API Response**

* Uses a returnData() method to return the response with a message (e.g., \_\_('api.employee\_all')).
* If an exception occurs, catches it and returns a 422 error with a message using returnError().

🔹 **Error Handling**

* All logic is wrapped in a try-catch block.
* Any exceptions are caught and returned as a structured error response.
  1. **get($id) Method**

**Overview**  
This method retrieves a specific admin user by ID, only if they have the admin role (role\_id = 7).

🔹 **Key Functionality**

1-**Find Admin by ID**

* Uses Eloquent’s find() method to retrieve a user with the given ID and ensures they have role\_id = 7.
* If a matching user is found, proceeds to the next step.

2-**Return Admin Data**

* Wraps the admin record in EmployeeResource for consistent formatting.
* Returns the response using returnData() with a success message (\_\_('api.admin\_get')).

3-**Handle Not Found Case**

* If no admin is found, returns a 404 error using returnError() and a not-found message (\_\_('api.admin\_not\_found')).

🔹 **Error Handling**

* Encloses the logic in a try-catch block.
* If an exception occurs during the process, catches it and returns a 422 error with an error message (\_\_('api.error\_happened') + exception message).
  1. **add(StoreAdminRequest $request) Method**

🔹 **Overview**  
This method handles the creation of a new admin user along with a linked client profile for system integration. It processes uploaded images, saves user data, and ensures a consistent relationship between the admin and client models.

🔹 **Key Functionality**

1-**Handle Image Upload (Optional)**

* Checks if the request contains an uploaded image.
* If present, processes the image using handleFile() method.
* The processed path is stored and used when creating the user.

2-**Create Admin User**

* Creates a new user with the role of admin (role\_id = 7).
* Uses data from the validated StoreAdminRequest.
* Includes fields such as name, email, phone, password, status, city\_id, area\_id, and the uploaded image.

3-**Create Linked Client Profile**

* If the admin was created successfully, a corresponding Client record is created.
* The client is assigned a default role (type = 3), active status (status = 1), and default activation\_code.
* Links the client to the admin via admin\_id.
* Sets default values like gender and complete\_data.

4-**Link Admin to Client**

* After creating the client, sets the client\_id field on the admin record.
* Saves the updated admin instance.

5-**Return API Response**

* Returns the newly created admin data wrapped in EmployeeResource.
* Sends a success message using \_\_('api.admin\_add').

🔹 **Error Handling**

* All logic is wrapped in a try-catch block.
* On failure, a 422 error is returned with the caught exception message appended to a localized error string (\_\_('api.error\_happened')).
  1. **update($id, UpdateAdminRequest $request) Method**

🔹 **Overview**  
This method updates an existing admin user’s profile (with role\_id = 7) and their associated client profile. It handles optional updates to profile information, password, and image uploads, including deletion of the old image if replaced.

🔹 **Key Functionality**

1-**Find Admin by ID**

* Searches for the user with the given $id and verifies the role is admin (role\_id = 7).
* If not found, returns a 404 error response.

2-**Handle New Image Upload (If Present)**

* Checks if the request contains a new image file.
* If provided:
  + Processes and saves the new image using handleFile().
  + Deletes the old image from the server if it exists using File::delete() to avoid clutter.

3-**Update Admin Details**

* Updates the admin’s profile with values from the request:
  + Uses existing values as fallback if the request values are null.
  + Updates name, email, phone, role\_id, status, city\_id, area\_id, and image.

4-**Update Password (If Provided)**

* If the password is included in the request, updates it separately.

5-**Update Associated Client Record**

* Finds the Client associated with the admin’s client\_id.
* Updates client fields such as username, email, phone, status, and city\_id, using admin request data or defaulting to existing values.

6-**Return API Response**

* Returns the updated admin data using EmployeeResource.
* Sends a localized success message with \_\_('api.admin\_update').

🔹 **Error Handling**

* All logic is wrapped in a try-catch block.
* If any error occurs during the process, returns a 422 error with the exception message appended to \_\_('api.error\_happened').
  1. **destroy($admin\_id) Method**

🔹 **Overview**  
This method deletes an admin user (with role\_id = 7) from the system based on the provided ID. It also removes the associated profile image from the file system if it exists.

🔹 **Key Functionality**

1-**Find Admin by ID**

* Searches for the admin with the given $admin\_id and ensures the user has a role\_id of 7.
* If no such admin exists, a 404 error response is returned using \_\_('api.admin\_not\_found').

2-**Delete Admin Image (If Exists)**

* Checks if the admin has a profile image saved.
* If so:
  + Retrieves the full path of the image using public\_path().
  + Deletes the image file using File::delete() if it exists on the server.

3-**Delete Admin Record**

* Calls the delete() method on the admin model to remove the database record.

4-**Return API Response**

* Returns a success response with a localized message: \_\_('api.admin\_delete').

🔹 **Error Handling**

* Wrapped in a try-catch block to handle any unexpected errors.
* If an exception occurs, it returns a 422 error response with the exception message appended to \_\_('api.error\_happened').

1. **Area Controller**
   1. **all(Request $request) Method**

🔹 **Overview**  
This method retrieves a list of all areas in the system, optionally filtered by a specific city ID. It also loads related data for each area such as points and representatives.

🔹 **Key Functionality**

1-**Initialize Area Query**

* Starts a query on the Area model using Area::query() to allow conditional clauses.

2-**Optional Filtering by City**

* If the city\_id is present in the request, the query filters areas that belong to the specified city.

3-**Eager Load Relationships**

* Loads related points and representatives using Eloquent’s with() to reduce the number of queries.

4-**Fetch & Return Results**

* Retrieves the areas ordered by the latest entries using latest()->get().
* Wraps the results using AreaResource::collection() for consistent API formatting.
* Returns the formatted data with a localized success message (\_\_('api.area\_all')).

🔹 **Error Handling**

* The operation is enclosed in a try-catch block.
* If any exception occurs, it returns a 422 error response with a descriptive message:  
  \_\_('api.error\_happened') . $error->getMessage().
  1. **get($id) Method**

🔹 **Overview**  
This method retrieves a specific area by its ID, including its associated points and representatives.

🔹 **Key Functionality**

1-**Find Area by ID**

* Uses Area::find($id) with with(['points', 'representatives']) to eager load related data.
* Ensures efficient querying by fetching related points and representatives in a single call.

2-**Check if Area Exists**

* If no area is found, returns a 404 error with the message from \_\_('api.area\_not\_found').

3-**Return Area Data**

* If found, wraps the area data using AreaResource for consistent API formatting.
* Returns the result with a success message from \_\_('api.area\_get').

🔹 **Error Handling**

* Enclosed in a try-catch block to catch any unexpected errors.
* If an exception occurs, returns a 422 error with a descriptive message:  
  \_\_('api.error\_happened') . $error->getMessage().
  1. **add(storeAreaRequest $request) Method**

🔹 **Overview**  
Create a new area and associates it with points and representatives if provided.

🔹 **Key Functionality**

1-**Create New Area**

* Creates a new Area using the fields:
  + name
  + city\_id
  + status

2-**Create Points**

* Uses $area->points()->createMany($request->points) to associate multiple points with the area using Laravel’s createMany() method.
* Assumes each item in $request->points is an array with the correct structure for a Point model.

3-**Assign Representatives (Optional)**

* If representative\_id is passed, loops through it to build a bulk insert array for the RepresntiveRegions model.
* Bulk inserts representative-region associations using RepresntiveRegions::insert($data).

🔹 **Return Response**

* Returns the newly created area wrapped in an AreaResource.
* Response includes a localized success message from \_\_('api.area\_add').

🔹 **Error Handling**

* Wrapped in a try-catch block to catch and report any unexpected issues.
* Returns a 422 error with a custom message on failure.
  1. **update(updateAreaRequest $request) Method**

**🔹 Overview**

Updates an existing area by modifying its main details and replacing its related points and representatives.

**🔹 Key Functionality**

**1-Find Area**

* Retrieves the area record based on the id provided in the request.
* If the area is not found, returns a 404 error with a localized message.

**2-Update Area Fields**

* Updates the area's core attributes such as:
  + name
  + city\_id
  + status
* Retains the existing values for any fields not provided in the request.

**3-Update Points**

* Deletes all existing points related to the area.
* Replaces them with the new set of points provided in the request using the relationship.
* Assumes $request->points contains a valid array of points.

**4-Update Representatives (Optional)**

* If representative\_id is included in the request:
  + Deletes old representative associations for this area.
  + Loops through the new representative\_id values.
  + Prepares and performs a bulk insert into the RepresntiveRegions table to associate representatives with the area.

**🔹 Return Response**

* Returns the updated area wrapped in AreaResource.
* Includes a localized success message from \_\_('api.area\_update').

**🔹 Error Handling**

* Entire logic is wrapped in a try-catch block.
* If any exception occurs, it returns a 422 error response with a localized error message indicating something went wrong.
  1. **destroy($id) Method**

### 🔹 Overview

Deletes an existing area and all its associated points from the database.

### 🔹 Key Functionality

#### 1-Find Area

* Searches for the area using the provided id.
* If not found, returns a 404 error with a localized message.

#### 2-Delete Points

* Deletes all related points of the area using the relationship $area->points()->delete().

#### 3-Delete Area

* After points are deleted, removes the area itself from the database using $area->delete().

### 🔹 Return Response

* Returns a success message using \_\_('api.area\_delete').

### 🔹 Error Handling

* Entire operation is wrapped in a try-catch block.
* On any exception, returns a 422 error with a localized error message indicating the issue.
  1. **allPoints() Method**

### 🔹 Overview

Retrieves all areas with their associated points and returns only the latitude and longitude of each point.

### 🔹 Key Functionality

#### 1-Retrieve Areas with Points

* Uses Eloquent's with('points') to eager-load all areas along with their related points.

#### 2-Extract Coordinates

* Uses map() to iterate over each area and its points.
* For each point, extracts only the lat and lng fields and builds a simplified structure.

#### 3-Format as Array

* Converts the mapped collection into a plain array using toArray().

### 🔹 Return Response

* Returns the collected coordinates under the points key in a standardized success response.

### 🔹 Error Handling

* Entire method is wrapped in a try-catch block.
* Returns a 422 error with a localized message if an exception occurs.

1. **Auth Controller**
   1. **login(LoginRequest $request) Method**

### 🔹 Overview

Handles user login by verifying credentials and assigning an activation code if the login is successful.

### 🔹 Key Functionality

#### 1-Authentication Attempt

* Uses auth()->attempt() to verify that the provided email and password are valid.

#### 2-Assign Activation Code

* On successful authentication, fetches the user using their email.
* Sets a static activation code (1234) and saves it to the user record.  
  (Typically, this should be a randomly generated secure code.)

#### 3-Prepare and Send SMS

* Constructs an Arabic message with the activation code.
* SMS sending logic is present but currently commented out (sendSms() call).

### 🔹 Return Response

* Returns a success response with a localized message indicating that the activation code has been sent.
* On authentication failure, returns an error with a 403 status code and a message that the email or password is incorrect.

### 🔹 Error Handling

* Wrapped in a try-catch block to catch any unexpected exceptions.
* On error, returns a 403 response with the exception message.
  1. **verifyCode(Request $request) Method**

### 🔹 Overview

Verifies the activation code (OTP) sent to the user and issues an API token upon successful verification.

### 🔹 Key Functionality

#### 1-Retrieve User

* Fetches the user based on the provided email.

#### 2-Verify Activation Code

* Compares the user's stored activation code (activation\_code) with the one provided in the request (otp).

#### 3-Token Generation (On Success)

* If the OTP matches, a new API token is generated using Laravel Sanctum (createToken).
* The token is then prefixed with "Bearer" and added to the response data.

### 🔹 Return Response

* **On Success**:  
  Returns the authenticated user wrapped in AuthResource along with a success message (تم تسجيل الدخول بنجاح).
* **On Failure**:  
  Returns a 403 error with a message indicating that the activation code is incorrect.

### 🔹 Error Handling

* Enclosed in a try-catch block.
* If any exception occurs during execution, a 403 error is returned with the exception message.

1. **Brand Controller**
   1. **all(Request $request) Method**

### 🔹 Overview

Retrieves all brands from the database, with support for optional pagination based on the incoming request.

### 🔹 Key Functionality

#### 1-Pagination Check

* Checks the request for the paginate parameter:
  + If paginate = 1: returns paginated data (10 items per page).
  + If paginate = 0: returns all data without pagination.
  + If paginate is not provided: defaults to paginated data (10 items per page).

#### 2-Fetch Brands

* Uses Brand::latest() to order brands by the latest created.
* Depending on pagination logic, either:
  + Uses paginate(10) to get paginated results, or
  + Uses get() to retrieve all results.

#### 3-Format Pagination Data

* If pagination is used, constructs a custom pagination array:
  + total
  + per\_page
  + current\_page
  + total\_pages

### 🔹 Return Response

* Wraps the result in brandsResource::collection($brands) for consistent API formatting.
* Returns a localized message from \_\_('api.brand\_all').
* If pagination is used, also returns the pagination metadata.

### 🔹 Error Handling

* Entire logic is wrapped in a try-catch block.
* On failure, returns a 422 error with the exception message using \_\_('api.error\_happened').
  1. **get($brand\_id) Method**

### 🔹 Overview

Retrieves a single brand by its unique identifier (brand\_id).

### 🔹 Key Functionality

#### 1-Find Brand

* Uses Brand::find($brand\_id) to locate the brand by its ID.
* Checks if the brand exists:
  + If **not found**, returns a 404 error with a localized message \_\_('api.brand\_not\_found').

#### 2-Wrap Brand Resource

* If the brand exists, wraps it in a brandsResource to ensure consistent API response structure.

### 🔹 Return Response

* Returns the found brand in a key "brand" using brandsResource.
* Includes a localized success message from \_\_('api.brand\_get').

### 🔹 Error Handling

* Wrapped in a try-catch block to gracefully handle unexpected errors.
* On exception, returns a 422 error with the exception message and \_\_('api.error\_happened').
  1. **add(StoreBrandRequest $request) Method**

### 🔹 Overview

Creates a new brand and optionally uploads an image if provided in the request.

### 🔹 Key Functionality

#### 1-Handle Image Upload (Optional)

* Checks if the request includes an uploaded image using $request->file('image').
* If present, uses a helper method handleFile() to upload the image and retrieve its path.
* The uploaded image path is stored and later saved with the brand.

#### 2-Create Brand

* Creates a new Brand using:
  + name from the request.
  + image path if uploaded, otherwise null.
  + status from the request.

### 🔹 Return Response

* Returns the newly created brand wrapped in brandsResource.
* Includes a localized success message from \_\_('api.brand\_add').

### 🔹 Error Handling

* No explicit try-catch in this method, which assumes StoreBrandRequest handles validation and file upload is managed correctly.
* If needed, consider adding try-catch for improved robustness.
  1. **update(UpdateBrandRequest $request, $brand\_id) Method**

### 🔹 Overview

Updates an existing brand's information, including its name, image (if provided), and status.

### 🔹 Key Functionality

#### 1-Find Brand

* Retrieves the Brand model by its id ($brand\_id).
* If the brand is not found, returns a 404 error with the message 'api.brand\_not\_found'.

#### 2-Handle Image Upload (Optional)

* If a new image is uploaded in the request ($request->file('image')), it handles the upload via the handleFile() method and updates the image path.
* If no new image is provided, the existing image path is retained ($brand->image).

#### 3-Update Brand

* Updates the Brand using the provided fields in the request:
  + name (defaults to the current brand's name if not provided).
  + image (updates to the new image path if provided, otherwise retains the existing one).
  + status (updates to the new status from the request).

### 🔹 Return Response

* Returns the updated brand wrapped in brandsResource.
* Includes a localized success message from \_\_('api.brand\_update').

### 🔹 Error Handling

* The method is wrapped in a try-catch block to handle unexpected errors.
* If an error occurs, returns a 422 error with the custom message \_\_('api.error\_happened') along with the error message from the exception.
  1. **destroy($brand\_id) Method**

### 🔹 Overview

Deletes a brand along with its associated image if available.

### 🔹 Key Functionality

#### 1-Find Brand

* Retrieves the Brand model by its id ($brand\_id).
* If the brand is not found, returns a 404 error with the message 'api.brand\_not\_found'.

#### 2-Delete Image (If Exists)

* If the brand has an associated image ($brand->image), it deletes the image from the public disk using Storage::disk('public')->delete($brand->image).

#### 3-Delete Brand

* Deletes the brand from the database using $brand->delete().

### 🔹 Return Response

* If successful, returns a success message indicating that the brand was deleted, using the localized success message from \_\_('api.brand\_delete').

### 🔹 Error Handling

* The method is wrapped in a try-catch block to handle unexpected errors.
* If an error occurs, returns a 422 error with the custom message \_\_('api.error\_happened') along with the error message from the exception.

1. **CancelingReasons Controller**
   1. **all(Request $request) Method**

### 🔹 Overview

Retrieves a list of cancellation reasons, with optional pagination.

### 🔹 Key Functionality

#### 1-Check for Pagination

* The method checks if the request contains a paginate parameter and whether it's set to 1 (for pagination) or 0 (for no pagination).

#### 2-Pagination Logic

* **When Pagination is Enabled (paginate = 1):**
  + Retrieves a paginated list of cancellation reasons (CancelReason::latest()->paginate(10)), with a limit of 10 per page.
  + Constructs a pagination object containing:
    - Total number of records.
    - Records per page.
    - Current page.
    - Total pages.

#### 3-No Pagination Logic (paginate = 0 or Missing)

* **When Pagination is Disabled (paginate = 0):**
  + Retrieves all cancellation reasons without pagination (CancelReason::latest()->get()).
* **When Pagination is Not Specified:**
  + Defaults to paginated results similar to the first case.

#### 4-Format the Response

* The method uses CancelingReasonResource::collection() to format the cancellation reasons into a structured resource format.
* If pagination is enabled, both the reasons and the pagination information are included in the response.

### 🔹 Return Response

* **Success:**
  + If the process is successful, it returns the cancellation reasons along with the pagination information (if applicable), using the localized success message from \_\_('api.reason\_all').
* **Error Handling:**
  + The method is wrapped in a try-catch block to catch any unexpected errors.
  + If an error occurs, it returns a 422 error with the exception's message.
  1. **get($reason\_id) Method**

### 🔹 Overview

Retrieves a single cancellation reason based on the given reason\_id.

### 🔹 Key Functionality

#### 1-Find Cancellation Reason by ID

* The method attempts to find a cancellation reason using the provided reason\_id with CancelReason::find($reason\_id).
* If the cancellation reason exists, it proceeds to format and return the data.

#### 2-Check If Reason Exists

* If the cancellation reason is found, it uses the CancelingReasonResource to format the response into a structured resource.
* If the cancellation reason does not exist, it returns an error message indicating that the reason was not found.

### 🔹 Return Response

#### ****Success:****

* **When the reason is found**, it returns the data wrapped in a data key with the formatted cancellation reason using CancelingReasonResource.
* The response also includes a localized success message from \_\_('api.reason\_get').

#### ****Error Handling:****

* **When an error occurs**, either due to an exception or if the reason is not found, the method catches the exception and returns a 422 error along with the exception message.
* **When the reason is not found**, it specifically returns a 404 error with a localized message from \_\_('api.reason\_not\_found').

### 🔹 Return Data Structure

* **Success:**
  + A data key containing:
    - reason: The formatted cancellation reason object.
* **Error:**
  + A 404 error with a localized message (\_\_('api.reason\_not\_found')) if the reason is not found.
  1. **add(StoreCancellingReasonRequest $request) Method**

### 🔹 Overview

Adds a new cancellation reason using the provided request data.

### 🔹 Key Functionality

#### 1-Create New Cancellation Reason

* The method attempts to create a new cancellation reason by calling CancelReason::create($request->all()), which creates the reason using all the data from the request.
* The request is assumed to be validated and structured properly by the StoreCancellingReasonRequest.

### 🔹 Return Response

#### ****Success:****

* If the cancellation reason is created successfully, the method returns the newly created reason wrapped in a data key.
* The response also includes a success message, localized from \_\_('api.reason\_add').

#### ****Error Handling:****

* If an error occurs while creating the cancellation reason, the method catches the exception and returns a 422 error with the exception message.

### 🔹 Return Data Structure

* **Success:**
  + A data key containing:
    - reason: The newly created cancellation reason, formatted using the CancelingReasonResource.
* **Error:**
  + A 422 error with the exception message if something goes wrong during the creation process.
  1. **update($reason\_id, UpdateCancellingReasonRequest $request) Method**

### 🔹 Overview

Updates an existing cancellation reason based on the provided ID and updated data.

### 🔹 Key Functionality

#### 1-Find the Cancellation Reason

* The method attempts to find the cancellation reason by its ID using CancelReason::find($reason\_id).
* If no reason is found with the provided ID, it returns an error message.

#### 2-Update the Cancellation Reason

* If the cancellation reason is found, the method proceeds to update the record using $reason->update($request->all()).
* It updates the reason with the data provided in the request.

### 🔹 Return Response

#### ****Success:****

* If the update is successful, it returns the updated cancellation reason wrapped in a data key.
* The response includes a success message, localized from \_\_('api.reason\_update').

#### ****Error Handling:****

* If no cancellation reason is found with the provided ID, it returns a 404 error with the message 'api.reason\_not\_found'.
* If an exception occurs during the update process, it returns a 422 error with the exception message.

### 🔹 Return Data Structure

#### ****Success:****

* **data**: Contains the updated cancellation reason, formatted using the CancelingReasonResource.

#### ****Error:****

* A 404 error if the cancellation reason is not found.
* A 422 error if an exception occurs during the update process.
  1. **destroy($reason\_id) Method**

### 🔹 Overview

Deletes a cancellation reason by its ID.

### 🔹 Key Functionality

#### 1-Find the Cancellation Reason

* The method first attempts to find the cancellation reason using the provided ID via CancelReason::find($reason\_id).
* If the cancellation reason is not found, it returns an error message indicating the reason was not found.

#### 2-Delete the Cancellation Reason

* If the cancellation reason is found, the method proceeds to delete the record using $reason->delete().
* It then returns a success message.

### 🔹 Return Response

#### ****Success:****

* If the deletion is successful, it returns a success message, localized from \_\_('api.reason\_delete').

#### ****Error Handling:****

* If no cancellation reason is found with the provided ID, it returns a 404 error with the message 'api.reason\_not\_found'.
* If an exception occurs during the deletion process, it returns a 422 error with the exception message.

### 🔹 Return Data Structure

#### ****Success:****

* A success message indicating the cancellation reason was successfully deleted.

#### ****Error:****

* A 404 error if the cancellation reason is not found.
* A 422 error if an exception occurs during the deletion process.

1. **Car Controller** 
   1. **getCars(Request $request) Method**

🔹 **Overview**  
Fetches a list of cars, optionally filtered by a client’s ID.

🔹 **Key Functionality**  
1-**Initialize Query**  
The method begins by initializing a query on the Car model.

2-**Filter by Client (Optional)**  
If a client\_id is provided in the request, the method applies a filter to retrieve cars associated with that specific client by using $cars->where('user\_id', $request->client\_id).

3-**Retrieve Cars**  
It then fetches the cars, ordering them by the most recent entries (using latest()), and returns them as a resource collection, which structures the data properly for API responses.

🔹 **Return Response**

* **Success:**  
  If the cars are successfully retrieved, a JSON response is returned with the list of cars and a 200 status code.
* **Error Handling:**  
  If an exception occurs during the process, a 403 status code is returned along with the error message.

🔹 **Return Data Structure**

* **Success:**
  + A 200 status with a cars key, containing an array of car data (formatted via the CarResource).
* **Error:**
  + A 403 status with the exception message if any error occurs during the execution.
  1. **addCar(Request $request) Method**

🔹 **Overview**  
Adds a new car entry to the system with the provided details.

🔹 **Key Functionality**  
1-**Input Validation**  
The method starts by defining validation rules for the car data. The rules ensure that certain fields, such as brand\_id, car\_type\_id, and color\_id, are provided and valid. If validation fails, the method returns a 403 error with the first validation error message.

2-**Fetch Brand Image (Optional)**  
If the brand\_id is provided in the request, the method fetches the brand’s image from the Brand model to associate with the new car entry.

3-**Create Car Record**  
The method then proceeds to create a new car record in the database, populating the fields with the request data. The car is associated with the client (user) based on client\_id, and the brand\_id and car\_type\_id are linked with the respective models.

4-**Return Success**  
Upon successfully creating the car, it returns a success message, indicating the car was added successfully.

🔹 **Return Response**

* **Success:**  
  A success message is returned with a 200 status code, indicating the car was added successfully.
* **Error Handling:**  
  If validation fails, a 403 status code is returned along with the validation error message. If an exception occurs during the process, a 422 status code is returned along with the exception message.

🔹 **Return Data Structure**

* **Success:**  
  A 200 status with a success message indicating the car was successfully added.
* **Error:**
  + A 403 status with the validation error message if validation fails.
  + A 422 status with the exception message if an error occurs during car creation.
  1. **updateCar(Request $request) Method**

🔹 **Overview**  
Updates an existing car's details in the system based on the provided car\_id and other request data.

🔹 **Key Functionality**  
1-**Input Validation**  
The method defines validation rules for the request data. It ensures the car\_id exists in the cars table, and validates the brand\_id, car\_type\_id, and color\_id fields to ensure they exist in their respective tables. If validation fails, it returns a 403 error with the first validation error message.

2-**Find the Car to Update**  
Using the provided car\_id, the method attempts to find the car that needs to be updated using Car::find($request->car\_id).

3-**Fetch Brand Image (Optional)**  
If a new brand\_id is provided in the request, the method fetches the associated brand's image to update the car's image.

4-**Update Car Record**  
The method updates the car record with the new data from the request, including fields like name, car\_size\_id, car\_plate\_number, image, color\_id, brand\_id, and car\_type\_id.

5-**Return Success**  
If the car is successfully updated, a success message is returned, indicating the car was updated successfully.

🔹 **Return Response**

* **Success:**  
  If the update is successful, a 200 status is returned with a success message indicating the car was updated.
* **Error Handling:**  
  If validation fails, a 403 status is returned along with the first validation error message. If an exception occurs during the update, a 422 status is returned with the exception message.

🔹 **Return Data Structure**

* **Success:**  
  A 200 status with a success message indicating the car was successfully updated.
* **Error:**
  + A 403 status with the validation error message if validation fails.
  + A 422 status with the exception message if an error occurs during the update.
  1. **colors() Method**

🔹 **Overview**  
Retrieves all available colors from the colors table.

🔹 **Key Functionality**

1-**Fetch Color Data**  
The method uses the DB::table() method to retrieve all the records from the colors table, selecting the columns id, color\_name, and hex\_code.

2-**Return Color Data**  
After fetching the data, it returns a JSON response with the data under the colors key.

🔹 **Return Response**

**Success:**  
If the data retrieval is successful, the method returns the list of colors in the response, structured as follows:  
{ "colors": [...] }.

**Error Handling:**  
If an exception occurs during the data retrieval process, the method returns a 422 error with the exception message.

🔹 **Return Data Structure**

**Success:**

* A JSON response containing an array of color objects with id, color\_name, and hex\_code.

**Error:**

* A 422 error with the exception message in case of an error during the data retrieval process.
  1. **deleteCar() Method**

🔹 **Overview**  
Deletes a car record by its car\_id from the database.

🔹 **Key Functionality**

1-**Validate Request Data**  
The method defines validation rules to ensure that the car\_id is required and exists in the cars table.  
If the validation fails, it returns a 403 error with the first validation error message.

2-**Find the Car Record**  
After validation, the method attempts to find the car using the provided car\_id via Car::find($request->car\_id).

3-**Delete the Car Record**  
If the car is found, the method proceeds to delete the car using $car->delete().

4-**Return Success Response**  
It then returns a success message indicating that the car has been deleted successfully.

🔹 **Return Response**

**Success:**  
If the car is successfully deleted, the method returns a success message, localized from \_\_('api.deleteCar').

**Error Handling:**

* If the car is not found, it will return a 403 error with the validation error message.
* If any other exception occurs during the process, it returns a 422 error with the exception message.

🔹 **Return Data Structure**

**Success:**

* A success message indicating the car was successfully deleted.

**Error:**

* A 403 error if the car\_id is invalid or not found in the cars table.
* A 422 error if an exception occurs during the deletion process.

1. **CarPlate Controller**
   1. **all() Method**

🔹 **Overview**  
Retrieves all car plates, with optional filtering based on user role and pagination.

🔹 **Key Functionality**

1-**Check User Role**  
The method first checks the role of the authenticated user. If the user has a role ID of 6 (indicating an investor), it filters car plates based on the investor's email and phone number.

2-**Determine Pagination**  
The method checks if the request includes a paginate parameter:

* If paginate is set to 1, it paginates the results with 10 items per page.
* If paginate is set to 0, it retrieves all the car plates without pagination.
* If paginate is not specified, it defaults to pagination with 10 items per page.

3-**Retrieve Car Plates**

* The method retrieves car plates either filtered by the investor or all car plates depending on the user’s role.
* It uses CarPlate::latest() to fetch the car plates in the latest order.

4-**Return Data**

* If pagination is applied, the method includes pagination information in the response.
* If pagination is not applied, it returns all the car plates.

🔹 **Return Response**

**Success:**

* If the car plates are successfully retrieved, the method returns the car plates, along with pagination data if applicable, and a success message localized from \_\_('api.car\_plates\_all').

**Error Handling:**

* If an exception occurs during the process, it returns a 422 error with a localized message and the exception details.

🔹 **Return Data Structure**

**Success:**

* A list of car plates, along with pagination details if pagination is used.

**Error:**

* A 422 error if an exception occurs during the retrieval process, with the message 'api.error\_happened' followed by the exception details.
  1. **get() Method**

🔹 **Overview**  
Retrieves a specific car plate by its ID.

🔹 **Key Functionality**

1-**Find the Car Plate**  
The method first attempts to find the car plate using the provided car\_plate\_id via CarPlate::find($car\_palte\_id).

2-**Check if Car Plate Exists**  
If the car plate is not found, it returns a 404 error with the message 'api.car\_plate\_not\_found'.

3-**Return Car Plate Data**  
If the car plate is found, the method returns the car plate data wrapped in a CarPlateResource, along with a success message localized from \_\_('api.car\_plate\_get').

🔹 **Return Response**

**Success:**

* If the car plate is found, it returns the car plate data in the data key with a success message.

**Error Handling:**

* If no car plate is found with the given ID, it returns a 404 error with the message 'api.car\_plate\_not\_found'.
* If an exception occurs during the process, it returns a 422 error with a localized message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A data object containing the car plate details wrapped in a CarPlateResource.

**Error:**

* A 404 error if the car plate is not found.
* A 422 error if an exception occurs during the retrieval process, with the message 'api.error\_happened' followed by the exception details.
  1. **add() Method**

🔹 **Overview**  
Adds a new car plate to the system.

🔹 **Key Functionality**

1-**Create the Car Plate**  
The method attempts to create a new car plate using the provided data (code and investor\_id) via CarPlate::create([...]).

2-**Return Success**  
If the car plate is successfully created, the method returns the newly created car plate data wrapped in a CarPlateResource along with a success message localized from \_\_('api.car\_plate\_add').

🔹 **Return Response**

**Success:**

* If the car plate is successfully created, it returns the car plate data in the data key along with a success message.

**Error Handling:**

* If an exception occurs during the creation process, it returns a 422 error with a localized message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A data object containing the newly created car plate details wrapped in a CarPlateResource.

**Error:**

* A 422 error if an exception occurs during the creation process, with the message 'api.error\_happened' followed by the exception details.
  1. **update() Method**

🔹 **Overview**  
Updates an existing car plate by its ID.

🔹 **Key Functionality**

1-**Find the Car Plate**  
The method first attempts to find the car plate using the provided ID ($car\_plate\_id) via CarPlate::find($car\_plate\_id).

* If the car plate is not found, it returns an error message indicating that the car plate was not found.

2-**Update the Car Plate**  
If the car plate is found, the method proceeds to update the car plate fields. It uses the provided data from the request (code and investor\_id), and if any field is not provided, it retains the current value in the database.

3-**Return Updated Data**  
Once the car plate is updated, the method returns the updated car plate data wrapped in a CarPlateResource along with a success message.

🔹 **Return Response**

**Success:**

* If the car plate is successfully updated, it returns the updated car plate data in the data key along with a success message.

**Error Handling:**

* If the car plate is not found, it returns a 404 error with the message 'api.car\_plate\_not\_found'.
* If an exception occurs during the update process, it returns a 422 error with a localized message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A data object containing the updated car plate details wrapped in a CarPlateResource.

**Error:**

* A 404 error if the car plate is not found, with the message 'api.car\_plate\_not\_found'.
* A 422 error if an exception occurs during the update process, with the message 'api.error\_happened' followed by the exception details.
  1. **destroy() Method**

🔹 **Overview**  
Deletes a car plate by its ID.

🔹 **Key Functionality**

1-**Find the Car Plate**  
The method first attempts to find the car plate using the provided ID ($car\_plate\_id) via CarPlate::find($car\_plate\_id).

* If the car plate is not found, it returns an error message indicating that the car plate was not found.

2-**Delete the Car Plate**  
If the car plate is found, the method proceeds to delete the car plate from the database using $carPlate->delete().

* It then returns a success message indicating the deletion was successful.

🔹 **Return Response**

**Success:**

* If the car plate is successfully deleted, it returns a success message indicating the car plate has been deleted.

**Error Handling:**

* If the car plate is not found, it returns a 404 error with the message 'api.car\_plate\_not\_found'.
* If an exception occurs during the deletion process, it returns a 422 error with a localized message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A success message indicating the car plate was successfully deleted.

**Error:**

* A 404 error if the car plate is not found, with the message 'api.car\_plate\_not\_found'.
* A 422 error if an exception occurs during the deletion process, with the message 'api.error\_happened' followed by the exception details.

1. **CarType Controller**
   1. **all() Method**

🔹 **Overview**  
Retrieves a list of car types, with the option to filter by brand and paginate the results.

🔹 **Key Functionality**

1-**Build the Car Types Query**  
The method starts by initializing the car types query using CarType::query().

* If a brand\_id is provided in the request, the query is filtered by brand\_id.

2-**Handle Pagination**  
The method checks if pagination is requested via the paginate parameter in the request:

* **If paginate is 1**: The results are paginated with a page size of 10.
* **If paginate is 0**: The results are fetched without pagination.
* **Default**: If no pagination option is specified, the results are paginated by default.

3-**Return the Results**

* After fetching the car types, the method returns the data with the appropriate response structure:
  + If pagination is applied, the pagination details (total, per\_page, current\_page, total\_pages) are included.
  + The response contains the car types and pagination (if applicable).

🔹 **Return Response**

**Success:**

* The response includes the list of car types, along with pagination details if requested.

**Error Handling:**

* If an exception occurs during the data retrieval process, a 422 error is returned with a message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* **With Pagination**: The response contains the car types and pagination data.
* **Without Pagination**: The response contains only the list of car types.

**Error:**

* A 422 error with the message 'api.error\_happened' followed by the exception details.
  1. **get() Method**

🔹 **Overview**  
Fetches a specific car type based on the provided car\_type\_id.

🔹 **Key Functionality**

1-**Find the Car Type**  
The method attempts to find the car type using CarType::find($car\_type\_id) based on the provided ID.

2-**Handle Missing Car Type**

* If no car type is found (i.e., the result is null), it returns a 404 error with the message 'api.car\_type\_not\_found'.

3-**Return Car Type Data**

* If the car type is found, the method wraps the result in the CarTypesResource and returns the car type data with the success message 'api.car\_type\_get'.

🔹 **Return Response**

**Success:**

* The response includes the car type data wrapped in the CarTypesResource and a success message.

**Error Handling:**

* If the car type is not found, a 404 error with the message 'api.car\_type\_not\_found' is returned.
* If an exception occurs during the data retrieval process, a 422 error is returned with the message 'api.error\_happened' followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* The response contains the requested car type data.

**Error:**

* A 404 error if the car type is not found.
* A 422 error if an exception occurs during the process.
  1. **add() Method**

🔹 **Overview**  
Adds a new car type to the database.

🔹 **Key Functionality**

1-**Create the Car Type**  
The method uses CarType::create() to insert a new record into the car\_types table. The car type is created with the name, brand\_id, and status values taken from the request data.

2-**Return the Created Car Type Data**  
After successfully creating the car type, the method returns the newly created car type wrapped in a CarTypesResource with the success message 'api.car\_type\_add'.

🔹 **Return Response**

**Success:**

* The response contains the newly created car type data, wrapped in the CarTypesResource and a success message indicating the car type was successfully added.

**Error Handling:**

* No explicit error handling is included in this method. If any exception occurs, the global error handling will capture it.

🔹 **Return Data Structure**

**Success:**

* The response contains the car type data and a success message.

**Error:**

* If an exception occurs during the creation process, it will be handled by the global error response (likely a 422 error with the exception message).
  1. **update() Method**

🔹 **Overview**  
Updates an existing car type in the database.

🔹 **Key Functionality**

1-**Find the Car Type**  
The method first attempts to find the car type using the provided car\_type\_id. If the car type does not exist, it returns a 404 error with the message 'api.car\_type\_not\_found'.

2-**Update Car Type Details**  
If the car type exists, the method updates the car type's details. It checks if the provided values for name, brand\_id, and status are present in the request. If a value is missing, it keeps the existing value from the database.

3-**Return Updated Car Type**  
After updating, the method returns the updated car type data wrapped in a CarTypesResource with a success message 'api.car\_type\_update'.

🔹 **Return Response**

**Success:**

* The response includes the updated car type data, wrapped in the CarTypesResource, and a success message indicating the car type was successfully updated.

**Error Handling:**

* If the car type is not found, it returns a 404 error with the message 'api.car\_type\_not\_found'.
* If any exception occurs during the update process, it returns a 422 error with the exception message.

🔹 **Return Data Structure**

**Success:**

* The response contains the updated car type data wrapped in the CarTypesResource and a success message.

**Error:**

* A 404 error if the car type is not found.
* A 422 error if an exception occurs during the update process.
  1. **destroy() Method**

🔹 **Overview**  
Deletes a car type by its ID.

🔹 **Key Functionality**

1-**Find the Car Type**  
The method first attempts to find the car type using the provided car\_type\_id. If the car type does not exist, it returns a 404 error with the message 'api.car\_type\_not\_found'.

2-**Delete the Car Type**  
If the car type exists, the method proceeds to delete the car type record from the database using the delete() method.

3-**Return Success Message**  
After successful deletion, the method returns a success message indicating the car type was successfully deleted.

🔹 **Return Response**

**Success:**

* The response includes a success message indicating that the car type was successfully deleted.

**Error Handling:**

* If the car type is not found, it returns a 404 error with the message 'api.car\_type\_not\_found'.
* If any exception occurs during the deletion process, it returns a 422 error with the exception message.

🔹 **Return Data Structure**

**Success:**

* A success message indicating the car type was successfully deleted.

**Error:**

* A 404 error if the car type is not found.
* A 422 error if an exception occurs during the deletion process.