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.

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

🔹 **Overview**  
Retrieves a list of cities, with optional support for pagination based on the request parameters.

🔹 **Key Functionality**

1-**Check for Pagination Request**  
The method checks whether the request includes the paginate parameter:

* If paginate is set to 1, results are paginated.
* If paginate is set to 0, all cities are retrieved without pagination.
* If paginate is not provided, it defaults to paginating the results.

2-**Fetch Cities**  
The cities are retrieved from the database, ordered by the latest entries.

* When paginated, only a fixed number of records (10 per page) are returned.
* When not paginated, all city records are retrieved.

3-**Build Pagination Metadata**  
If pagination is applied, a pagination array is prepared containing:

* Total number of records.
* Number of records per page.
* Current page number.
* Total number of pages.

4-**Format and Return Response**  
The city records are wrapped in a resource collection (citiesResource) and returned.

* If pagination is applied, the pagination metadata is also included.

🔹 **Return Response**

**Success:**

* A list of cities wrapped in a resource collection.
* Pagination metadata is included if pagination is enabled.
* A success message localized from \_\_('api.city\_all').

**Error Handling:**

* If any exception occurs during processing, a 422 error is returned with the message \_\_('api.error\_happened') followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A data object containing:
  + cities: A list of formatted city records.
  + pagination: (optional) Pagination details if applicable.

**Error:**

* A 422 error if an exception occurs, with a localized message and the exception details.
  1. **add() Method**

🔹 **Overview**  
Adds a new city after checking for duplicates.

🔹 **Key Functionality**

1-**Duplicate Check**  
Checks if a city with the same name already exists in the cities table.  
If a match is found, a 409 Conflict response is returned with a localized message indicating the city already exists.

2-**Create New City**  
If the city doesn't already exist, a new city is created using the provided name from the request.

3-**Resource Wrapping**  
The newly created city is wrapped in a resource for consistent API formatting in the response.

🔹 **Return Response**

🔹 **Success**  
Returns the newly created city inside a data array.  
**Message:** localized success message indicating the city was added successfully.

🔹 **Duplicate Error**  
Returns a 409 Conflict status.  
**Message:** localized message indicating the city already exists.

🔹 **Exception/Error**  
Returns a 422 Unprocessable Entity status if any exception is thrown.  
**Message:** localized generic error message along with the exception message.

🔹 **Return Data Structure**

🔹 **Success**

* data contains the created city formatted using a resource.
* message contains a localized success message.

🔹 **Duplicate Error**

* error is set to 409.
* message contains a localized duplicate city message.

🔹 **Exception**

* error is set to 422.
* message contains a localized error message along with exception details.
  1. **get() Method**

🔹 **Overview**  
Fetches the details of a specific city using its ID.

🔹 **Key Functionality**

1-**City Lookup**  
Attempts to retrieve the city by its ID from the database.  
If the city is not found, it returns a 404 Not Found response with a localized error message.

2-**Resource Wrapping**  
If the city is found, it is wrapped in a resource for standardized API formatting.

🔹 **Return Response**

🔹 **Success**  
Returns the city inside a data object.  
**Message:** localized message indicating successful retrieval.

🔹 **City Not Found**  
Returns a 404 Not Found status if no city is found with the given ID.  
**Message:** localized message indicating the city was not found.

🔹 **Exception/Error**  
Returns a 422 Unprocessable Entity status if an exception occurs.  
**Message:** localized error message along with the exception details.

🔹 **Return Data Structure**

🔹 **Success**

* data contains the city formatted using a resource.
* message contains a localized success message.

🔹 **City Not Found**

* error is set to 404.
* message contains a localized message indicating the city was not found.

🔹 **Exception**

* error is set to 422.
* message contains a localized error message with exception details.
  1. **update() Method**

🔹 **Overview**  
Updates the details of an existing city using the provided city ID.

🔹 **Key Functionality**

1-**Find City by ID**  
The method searches for the city using the given ID:

* If the city is **not found**, it returns a 404 error with a localized message from \_\_('api.city\_not\_found').

2-**Update City Information**  
If the city is found, its name is updated using the value provided in the request:

* If the name is **not provided**, the current name is retained.

3-**Format and Return Response**  
After the update, the updated city record is wrapped in a citiesResource for consistent API formatting and returned in the response.

🔹 **Return Response**

**Success:**

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

**City Not Found:**

* Returns a 404 error with a localized message from \_\_('api.city\_not\_found').

**Error Handling:**

* If an exception occurs, returns a 422 error with the message \_\_('api.error\_happened') followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* A data object containing:
  + city: The updated city record formatted using citiesResource.
* message: A localized success message.

**City Not Found:**

* error: 404
* message: A localized not-found message.

**Exception:**

* error: 422
* message: A localized error message with the exception details.
  1. **destroy() Method**

🔹 **Overview**  
Deletes a city based on its ID after verifying that no related neighborhoods exist.

🔹 **Key Functionality**

1-**Find City by ID**  
The method looks up the city using the provided ID:

* If the city is **not found**, a 404 error is returned with a localized message from \_\_('api.city\_not\_found').

2-**Check for Related Neighborhoods**  
Before deleting, the method checks whether the city is linked to any neighborhoods:

* If there **are related neighborhoods**, a 409 conflict error is returned with a custom message indicating that the city cannot be deleted until its neighborhoods are removed.

3-**Delete City**  
If no related neighborhoods are found, the city is soft-deleted from the database.

🔹 **Return Response**

**Success:**

* Returns a localized success message from \_\_('api.city\_delete') after successful deletion.

**City Not Found:**

* Returns a 404 error with a localized message from \_\_('api.city\_not\_found').

**Conflict (Related Neighborhoods Exist):**

* Returns a 409 error with a custom message indicating that the city is linked to neighborhoods and cannot be deleted.

**Error Handling:**

* If any exception occurs, returns a 422 error with the message \_\_('api.error\_happened') followed by the exception details.

🔹 **Return Data Structure**

**Success:**

* message: A localized success message confirming deletion.

**City Not Found:**

* error: 404
* message: A localized not-found message.

**Conflict (Neighborhoods Exist):**

* error: 409
* message: A message explaining the deletion restriction due to related neighborhoods.

**Exception:**

* error: 422
* message: A localized error message with the exception details.

1. **Client Controller**

**10.1 all(Request $request) Method**

🔹 **Overview**  
Retrieves a list of clients with flexible filtering, searching, sorting, and pagination options. It also supports detailed analytics when is\_notify is enabled.

🔹 **Key Functionality**

1-**Initialize Query**  
Starts building the client query using Client::query() and eager-loads the count of completed reservations (status = 6) using withCount().

2-**Search Filtering**  
If the search parameter is present, it filters clients of type 0 by:

* username
* phone
* email

Each field is searched using a partial match (LIKE %value%).

3-**Sorting Options**  
If the sort parameter is present:

* most-orderd: Sorts by highest number of completed reservations.
* least-orderd: Sorts by lowest number of completed reservations.  
  If not present, defaults to sorting by id in descending order.

4-**Status Filtering**

* status=zero-orders:  
  Clients without reservations or only those whose reservations have status = 6.
* status=zero-packages:  
  Clients without any packageSubscriptions.
* status=one-packages:  
  Clients who **have** packageSubscriptions.

All of these filters are scoped to clients of type 0.

5-**City-Based Filtering**

If city\_id is provided, filters clients whose **last reservation** is in the specified city.

6-**Pagination Handling**

Based on the paginate flag:

🔸 **paginate=1 and is\_notify=1:**

* Returns paginated client results (perPage = 500).
* Builds an array with categorized client stats:
  + All clients
  + Clients without reservations
  + Clients with/without packages
  + Clients without reservations for the past:
    - Week
    - Month
    - 24, 48, and 72 hours
* Each group includes:
  + id
  + username (label)
  + count (number of clients in that group)

🔸 **paginate=1 without is\_notify:**

* Regular paginated results (perPage = 10)
* Returns transformed clients via ClientResource with pagination meta.

🔸 **paginate=0:**

* Returns **all clients** as a flat list without pagination.
* Transformed using ClientResource.

🔸 **If paginate not provided:**

* Defaults to paginated results (perPage = 500).

🔹 **Return Response**

Always returns data using the helper returnData():

**Structure:**

{

"data": {

"clients": [...],

"pagination": {

"total": X,

"per\_page": X,

"current\_page": X,

"total\_pages": X

}

},

"message": "Localized message (\_\_('api.client\_all'))"

}

For is\_notify=1, the clients array includes stats for analytical grouping as well.

🔹 **Return Data Structure (if notify mode)**

Each item in the clients array represents:

* A specific category of clients.
* username: a label (e.g., "كل العملاء", "عملاء لم يحجزوا منذ اسبوع")
* count: total number of clients in that group.

**10.2 get() Method**

### 🔹 Overview

Fetches the details of a specific client by their unique ID and returns the information in a structured resource format.

**🔹 Key Functionality**

**1-Find Client by ID**

The method attempts to locate a client using the provided ID:

* If the client does not exist, it returns a 404 error with a localized "client not found" message.

**2-Return Client Data**

If the client is found, the method wraps the client's information using a resource transformer and returns it in the response.

**🔹 Return Response**

**🔹 Success:**

Returns a successful response containing the client's data, along with a localized success message.

**🔹 Client Not Found:**

Returns a 404 error response with a localized message indicating that the client was not found.

**🔹 Exception:**

In case of an unexpected error, returns a 422 error response including a localized error message and the exception details.

**🔹 Return Data Structure**

**Success:**

* Status: Success
* Data: Contains the client information
* Message: Localized message confirming successful retrieval

**Client Not Found:**

* Error Code: 404
* Message: Localized message indicating the client could not be found

**Exception:**

* Error Code: 422
* Message: Localized message describing the error that occurred

**10.3 add() Method**

### 🔹 Overview

Creates and stores a new client in the system after validating the input and handling an optional image upload.

**🔹 Key Functionality**

**1-Receive Validated Data**

Receives input from a validated request (StoreClientRequest), which ensures all required fields meet the defined validation rules before proceeding.

**2-Initialize and Fill Client**

A new client instance is created and populated with the input data, excluding the image field, which is handled separately.

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

If an image is provided:

* It is stored in the uploads/clients directory within the public storage disk.
* The stored image path is assigned to the client's image attribute.

**4-Save Client**

The client record, including the uploaded image if provided, is saved to the database.

**🔹 Return Response**

**🔹 Success:**

Returns a successful response containing the newly created client's data, wrapped in a resource, along with a localized success message.

**🔹 Exception:**

In the event of any unexpected error during the process, a 422 error is returned with a localized message and details about the exception.

**🔹 Return Data Structure**

**Success:**

* Status: Success
* Data: Contains the created client information
* Message: Localized message confirming successful addition

**Exception:**

* Error Code: 422
* Message: Localized error message with exception details

**10.4 update() Method**

### 🔹 Overview

Updates the information of an existing client using validated input, with optional handling for password and image changes.

**🔹 Key Functionality**

**1-Find Client by ID**

* Searches for the client using the provided client\_id.
* If no matching client is found, a 404 error is returned with a localized "not found" message.

**2-Update Client Fields**

* Populates the client model with validated request data, excluding the image.
* If a new password is provided, it is updated separately.

**3-Handle Image Replacement (Optional)**

If a new image is provided:

* Deletes the old image from storage, if it exists.
* Uploads the new image to the uploads/clients directory within the public storage disk.
* Updates the client’s image path with the new file.

**4-Save Changes**

* The updated client record is saved to the database, including the new image and/or password if applicable.

**🔹 Return Response**

**🔹 Success:**

* Returns a success response containing the updated client's data wrapped in a resource.
* Includes a localized message confirming the update.

**🔹 Client Not Found:**

* Returns a 404 error if the client with the given ID does not exist.
* Includes a localized "client not found" message.

**🔹 Exception:**

* If an error occurs during the update process, a 422 error is returned.
* Includes a localized error message and the exception details.

**🔹 Return Data Structure**

**Success:**

* Status: Success
* Data: Contains the updated client information
* Message: Localized message confirming successful update

**Client Not Found:**

* Error Code: 404
* Message: Localized message indicating the client was not found

**Exception:**

* Error Code: 422
* Message: Localized error message with exception details

**10.5 destroy() Method**

### 🔹 Overview

Soft-deletes a client record after performing necessary cleanup such as modifying the phone number and removing the profile image from storage.

**🔹 Key Functionality**

**1-Find Client by ID**

* Looks up the client using the provided client\_id.
* If the client does not exist, returns a 404 error with a localized "client not found" message.

**2-Modify Phone Number Before Deletion**

* Updates the client’s phone number to include a \_deleted suffix along with the current timestamp.
* This ensures phone number uniqueness and avoids future conflicts.

**3-Delete Profile Image from Storage**

* If the client has a stored image, it is removed from the storage disk before deletion.

**4-Soft Delete Client**

* Performs a soft delete on the client record, preserving the record in the database but marking it as deleted.

**🔹 Return Response**

**🔹 Success:**

* Returns a localized success message confirming the client was successfully deleted.

**🔹 Client Not Found:**

* Returns a 404 error if the client ID does not exist.
* Includes a localized message indicating the client was not found.

**🔹 Exception:**

* If any exception occurs during the process, returns a 422 error.
* Includes a localized error message with the exception details.

**🔹 Return Data Structure**

**Success:**

* Status: Success
* Message: Localized success message confirming deletion

**Client Not Found:**

* Error Code: 404
* Message: Localized "client not found" message

**Exception:**

* Error Code: 422
* Message: Localized error message with exception details

**10.6 exportNoOrdersNotifications() Method**

### 🔹 Overview

Sends notifications to clients who have not placed any orders (i.e., reservations) or have only unconfirmed ones. The method also supports sending notifications to a custom list of client IDs passed in the request.

**🔹 Key Functionality**

**1-Retrieve Client IDs from Request**

* Accepts a list of client\_ids from the request.
* If the list is sent as a JSON string, it is decoded into an array.

**2-Identify Clients with No Orders**

* Performs a left join between the clients and reservations tables.
* Filters to include only clients of type 0 (e.g., regular clients).
* Groups results by client ID.
* Applies a havingRaw condition to detect clients who either:
  + Have no reservations, or
  + Only have unconfirmed reservations (status = 0).

**3-Filter for “No Order” Clients If Needed**

* If the request contains 0 in the client IDs list, it is interpreted as a request to notify all clients who have no orders.
* In that case, the client ID list is replaced with the list of "no order" clients identified in the previous step.

**4-Create and Send Notifications**

* Iterates through the final list of client IDs.
* For each client:
  + A new notification record is created and saved.
  + A helper method (sendNotification) is invoked to actually send the notification.

**🔹 Return Response**

**🔹 Success:**

* Returns a localized success message confirming that notifications were sent successfully.

**🔹 Return Data Structure**

**Success:**

* Status: Success
* Message: Localized message confirming that notifications were successfully sent to targeted clients.

**10.7 sendNotification() Method**

### 🔹 Overview

This private helper method is responsible for sending a push notification to a specific client using their device token.

**🔹 Key Functionality**

**1-Retrieve Client by ID**

* Looks up the client based on the provided client\_id.
* Ensures the client exists before proceeding.

**2-Send Notification**

* If the client is found and valid, a push notification is sent using the send\_notification() helper function.
* The notification payload includes:
  + A title from the request.
  + A body message from the request.

**🔹 Return Response**

**🔹 Always returns true:**

* This indicates that the notification logic completed, regardless of whether the client was found or not.

**🔹 Notes**

* This method assumes that the send\_notification() helper handles actual communication with a push notification service (e.g., Firebase).
* There’s no error thrown if the client is not found; the method simply skips sending.

**10.8 getFrequentlyRequestingClients() Method**

### 🔹 Overview

This method retrieves a list of the most frequently requesting clients based on their reservation history within a specified date range. It also supports pagination for the results.

**🔹 Key Functionality**

**1-Date Filtering**

* The method accepts two optional query parameters: from and to dates.
* It dynamically builds the query based on the provided dates to filter the reservations:
  + If both dates are provided, it filters the reservations within the date range.
  + If only the from date is provided, it filters reservations from that date onward.
  + If only the to date is provided, it filters reservations up to that date.
  + If no dates are provided, it defaults to fetching all reservations with a status of 3.

**2-Fetch Clients with Reservation Count**

* Clients are retrieved with a count of their related reservations (filtered by the status 3).
* The result is sorted by the reservation count in descending order, showing the most frequent clients first.

**3-Pagination (Optional)**

* If the paginate query parameter is set to 1, pagination is applied:
  + The results are sliced based on the current page and a fixed number of items per page (200).
  + Pagination metadata such as total records, current page, and total pages is returned along with the client data.

**🔹 Return Response**

**🔹 Success Response**

* If pagination is requested, the response includes:
  + clients: A collection of clients with their reservation counts.
  + pagination: Metadata including total, per\_page, current\_page, and total\_pages.
* If pagination is not requested, the response only includes the clients data.

**🔹 Error Handling**

* If an error occurs, the method returns a 422 error with the message details.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: "العملاء الاكثر طلبا" (Localized message for "Most Requested Clients").
* **Data**:
  + **clients**: A collection of ClientResource data (client information along with reservation counts).
  + **pagination** (if applicable):
    - total: Total number of clients matching the criteria.
    - per\_page: Number of items per page (fixed at 200).
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**: 422
* **Message**: The error message detailing the issue.

**10.9 clientReservations() Method**

### 🔹 Overview

This method retrieves a list of reservations made by a specific client, paginated to display a set number of records per page. It sorts the reservations by the latest and returns the paginated results along with the necessary pagination metadata.

**🔹 Key Functionality**

**1-Retrieve Reservations**

* The method fetches reservations associated with a given client\_id.
* It retrieves the reservations in descending order based on the most recent reservation (using the latest() method).
* A fixed number of records per page (10) is applied using pagination.

**2-Pagination**

* The method calculates the pagination metadata, including:
  + total: Total number of reservations.
  + per\_page: Number of reservations displayed per page (set to 10).
  + current\_page: The current page of results.
  + total\_pages: The total number of pages based on the number of records and per-page limit.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **reservations**: A collection of reservations in ReservationResource format.
  + **pagination**: Metadata including total, per\_page, current\_page, and total\_pages.
* **Message**: Returns a localized success message from \_\_('api.reservation\_all') indicating the retrieval of all client reservations.

**🔹 Error Handling**

* If an error occurs, the method returns a 422 error with the exception message details.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized message confirming the successful retrieval of client reservations.
* **Data**:
  + **reservations**: A collection of reservation data.
  + **pagination**:
    - total: Total number of reservations.
    - per\_page: Number of reservations per page (set to 10).
    - current\_page: Current page of the results.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**: 422
* **Message**: The error message detailing the issue.

**10.10 getFrequentlyRequestingClientsInMorning() Method**

### 🔹 Overview

This method retrieves the list of clients who made reservations between 6:00 AM and 11:59 AM, focusing on reservations that have a specific status. It can paginate the results and return metadata along with the client data.

**🔹 Key Functionality**

**1-Time Filtering**

* The method defines the morning time range between 6:00 AM and 11:59 AM.
* It filters reservations by checking whether the reservation's from time falls within this range.
* It uses a raw SQL query to handle the time format conversion, adjusting the from time based on whether it's in Arabic AM/PM format or standard English format.

**2-Status Filter**

* Only reservations with a status of 3 are included in the result set, filtering out other statuses.

**3-Client Query**

* The method uses withCount to count the number of matching reservations for each client.
* The clients are then ordered by the number of reservations, starting with those who have the most.

**4-Pagination**

* The method supports pagination if the paginate parameter is passed in the request:
  + It paginates the clients, showing a maximum of 200 clients per page.
  + Metadata is included in the response to indicate the total number of clients, the current page, and the total pages.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of clients, represented by ClientResource.
  + **pagination** (if paginated):
    - total: Total number of clients.
    - per\_page: Number of clients displayed per page (set to 200).
    - current\_page: Current page number.
    - total\_pages: Total number of pages.
* **Message**: Returns a localized success message "الطلبات الاكثر طلبا في الصباح" (Most requested orders in the morning) in Arabic.

**🔹 Error Handling**

* If any error occurs, the method returns a 422 error with the exception message.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of clients who made the most requests in the morning.
* **Data**:
  + **clients**: A collection of client data.
  + **pagination** (if paginated):
    - total: Total number of clients.
    - per\_page: Number of clients per page (set to 200).
    - current\_page: Current page of the result.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**: 422
* **Message**: The error message detailing the exception.

**10.11 getClientMorningReservations() Method**

### 🔹 Overview

This method retrieves the list of morning reservations (from 6:00 AM to 11:59 AM) for a specific client. It also supports pagination and returns reservation details with metadata.

**🔹 Key Functionality**

**1-Client Existence Check**

* The method first checks if the client exists by querying the Client model with the provided clientId.
* If the client is not found, it returns a 404 error with the message "العميل غير موجود" (Client not found).

**2-Time Filtering**

* The method defines the morning time range between 6:00 AM and 11:59 AM.
* It uses a raw SQL query to convert the reservation's from time to a consistent format and filters those within the morning time range.
* The time format conversion adjusts for Arabic AM/PM formatting (ص for AM and م for PM).

**3-Status Filter**

* It filters the reservations to include only those with a status of 3 (likely representing completed or confirmed reservations).

**4-Pagination**

* The method paginates the results to show up to 10 reservations per page.
* Metadata is included in the response to provide the total number of reservations, the current page, and the total pages.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **reservations**: A collection of morning reservations for the specified client, represented by ReservationResource.
  + **pagination**: Metadata providing details about pagination.
* **Message**: Returns a localized success message "الحجوزات الصباحية للعميل" (Morning reservations for the client) in Arabic.

**🔹 Error Handling**

* If the client does not exist, the method returns a 404 error.
* If any other error occurs during the query, the method returns a 422 error with the exception message.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of the client's morning reservations.
* **Data**:
  + **reservations**: A collection of reservation data.
  + **pagination** (if paginated):
    - total: Total number of morning reservations for the client.
    - per\_page: Number of reservations displayed per page (set to 10).
    - current\_page: Current page of the result.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**: 404 if the client is not found, or 422 for other errors.
* **Message**: The error message detailing the exception.

**10.12 getFrequentlyRequestingClientsInEvening() Method**

### 🔹 Overview

This method retrieves the list of clients who have made the most evening reservations (from 12:00 PM to 11:59 PM). It supports dynamic date filtering and pagination, returning clients sorted by the number of reservations they made.

**🔹 Key Functionality**

**1-Evening Time Range**

* The method defines the evening time range between 12:00 PM and 11:59 PM.
* It ensures that the from time of the reservation falls within this range.

**2-Date Threshold Filtering**

* The method filters the reservations to include only those that occurred after a specified threshold date (2024-02-20).

**3-Status Filter**

* The reservations are filtered to include only those with a status of 3 (likely representing confirmed or completed reservations).

**4-Pagination**

* The method supports pagination. If the paginate query parameter is set to 1, it slices the results into pages, returning up to 200 clients per page.
* Metadata is included in the response to provide the total number of clients, the current page, and the total number of pages.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of clients with the most evening reservations, represented by ClientResource.
  + **pagination**: Pagination metadata (if requested).
* **Message**: Returns a localized success message "الطلبات الاكثر طلبا في المساء" (Most requested clients in the evening) in Arabic.

**🔹 Error Handling**

* If an error occurs during the query execution, the method returns a 422 error with the exception message.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of the top clients with evening reservations.
* **Data**:
  + **clients**: A collection of client data.
  + **pagination** (if paginated):
    - total: Total number of top clients.
    - per\_page: Number of clients displayed per page (set to 200).
    - current\_page: Current page of the result.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**: 422 error if there is a problem with the query.
* **Message**: The error message detailing the exception.

**10.13 getClientReservationsInEvening() Method**

### 🔹 Overview

This method retrieves the list of reservations for a specific client made during the evening (from 12:00 PM to 11:59 PM). It supports pagination, returning up to 10 reservations per page. If the client doesn't exist, an error response is provided.

**🔹 Key Functionality**

**1-Evening Time Range**

* The method defines the evening time range between 12:00 PM and 11:59 PM.
* It filters the reservations based on the from time being within this range.

**2-Status Filter**

* The method filters the reservations to include only those with a status of 3 (likely representing confirmed or completed reservations).

**3-Pagination**

* The method supports pagination. It returns up to 10 reservations per page.
* Pagination metadata is included, such as the total number of reservations, current page, and total pages.

**4-Client Existence Check**

* The method checks if the client exists in the database before attempting to retrieve their reservations.
* If the client is not found, a 404 error is returned with the message 'العميل غير موجود' (Client not found).

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **reservations**: A collection of reservations made by the client during the evening, represented by ReservationResource.
  + **pagination**: Pagination metadata, including total reservations, current page, and total pages.
* **Message**: Returns a localized success message "حجوزات العميل في المساء" (Client evening reservations) in Arabic.

**🔹 Error Handling**

* If an error occurs during the query execution or if the client doesn't exist, the method returns a 422 error with the exception message, or a 404 if the client is not found.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of the client's evening reservations.
* **Data**:
  + **reservations**: A collection of reservation data for the client.
  + **pagination**: Metadata for pagination:
    - total: Total number of reservations.
    - per\_page: Number of items per page (set to 10).
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

**🔹 Error**

* **Error**:
  + 404 error if the client is not found.
  + 422 error if there is an issue during the query execution.
* **Message**: An error message detailing the exception or the client not being found.

**10.14 getMostRatedClients() Method**

### 🔹 Overview

This method retrieves the top clients based on the number of reviews (ratings) they have made for their reservations. It supports pagination and returns clients in descending order of their review count. The method also includes a fallback to return all clients if pagination is not requested.

**🔹 Key Functionality**

**1-Join Operations**

* **Reservations**: The method joins the clients table with the reservations table to retrieve reservations related to each client.
* **Reviews**: The method further joins with the reviews table to count the number of reviews (ratings) each client has made for their reservations.

**2-Count of Reviews**

* The selectRaw method is used to count the number of reviews for each client, giving us the number of rated orders (reservations\_count).

**3-Grouping Data**

* The method groups the data by the client’s id, username, and phone to prevent SQL errors from multiple aggregation results.

**4-Sorting**

* Clients are ordered in descending order by the reservations\_count, meaning the clients with the most reviews appear first.

**5-Pagination**

* If the paginate parameter is provided in the request with a value of 1, pagination is applied.
  + The number of clients per page is set to 200.
  + Pagination metadata is returned, including total records, current page, and total pages.
* If pagination is not required, all clients are returned in one response.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of clients, each with their id, username, phone, and reservations\_count (the count of reviews they received).
  + **pagination**: Pagination metadata (if requested), including:
    - total: Total number of clients.
    - per\_page: Number of items per page (set to 200).
    - current\_page: Current page number.
    - total\_pages: Total number of pages.
* **Message**: A localized success message: "العملاء الاكثر تقييما للطلبات" (The most rated clients).

**🔹 Error Handling**

* The method returns the appropriate success response without explicit error handling for database issues, as it's assumed the query will run without issues under normal circumstances.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of the top-rated clients.
* **Data**:
  + **clients**: A collection of client data, each client having:
    - id: Client's unique identifier.
    - username: Client's username.
    - phone: Client's phone number.
    - reservations\_count: The total number of reviews the client has received.
  + **pagination** (if requested):
    - total: Total number of clients.
    - per\_page: Number of items per page.
    - current\_page: Current page number.
    - total\_pages: Total pages based on the number of clients and items per page.

**🔹 Error**

* **Error**:
  + Not explicitly defined in the code, as it assumes successful execution if no issues occur during the query.

**10.15 getReviewedReservations() Method**

### 🔹 Overview

This method retrieves a list of reservations for a specific client that have been reviewed. It supports pagination and returns the reservations in the latest order.

**🔹 Key Functionality**

**1-Filter by Client ID**

* The method filters the reservations based on the provided clientId to ensure that only the specific client's reservations are considered.

**2-Ensure Reviewed Reservations**

* The method uses the whereHas() method to ensure that the reservations being retrieved have an associated review.

**3-Sorting**

* The reservations are ordered by the most recent first, using the latest() method.

**4-Pagination**

* The method retrieves a paginated list of reviewed reservations, with 10 items per page.
  + Pagination metadata is returned, including:
    - total: The total number of reservations that match the criteria.
    - per\_page: The number of reservations per page (set to 10).
    - current\_page: The current page number.
    - total\_pages: The total number of pages based on the number of reservations and items per page.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **reservations**: A collection of reservations, each having detailed information about the reservation.
  + **pagination**: Pagination metadata, which includes:
    - total: The total number of reservations.
    - per\_page: The number of reservations displayed per page.
    - current\_page: The current page of results.
    - total\_pages: The total number of pages.
* **Message**: A localized success message: "تم استرجاع الطلبات بنجاح" (Reservations retrieved successfully).

**🔹 Error Handling**

* If there is an error during the execution of the method (e.g., database issues), it returns an error message with status code 422 and the exception's message.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: A localized success message confirming the retrieval of reviewed reservations.
* **Data**:
  + **reservations**: A collection of reservations, each reservation containing the relevant details (based on the ReservationResource).
  + **pagination** (optional if paginated):
    - total: The total number of reservations.
    - per\_page: The number of reservations per page (set to 10).
    - current\_page: The current page of results.
    - total\_pages: The total number of pages based on the number of reservations and items per page.

**🔹 Error**

* **Error**:
  + If an error occurs, the response will contain the error code 422 and the exception message ($error->getMessage()).

**10.16 getHighestRatingClients() Method**

### 🔹 Overview

This method retrieves a list of clients with the highest average ratings based on their associated reviews. It supports pagination and returns the clients ordered by the highest average rating.

**🔹 Key Functionality**

**1-Join with Reservations and Reviews**

* The method performs a leftJoin with the reservations and reviews tables to get the total ratings and average rating score for each client.

**2-Aggregate Rating Data**

* The method calculates:
  + total\_rating: The sum of all review scores for the client's reservations.
  + avg\_rating: The average rating score for the client's reservations.

**3-Group by Client**

* The method groups the data by client (clients.id, clients.username, clients.phone) to ensure each client’s ratings are aggregated correctly.

**4-Sorting**

* The method sorts the clients by their avg\_rating in descending order, ensuring that clients with the highest average rating come first.

**5-Pagination**

* The method supports pagination:
  + It retrieves the top clients with their aggregated ratings.
  + Pagination metadata is returned, including:
    - total: The total number of clients matching the criteria.
    - per\_page: The number of items displayed per page (set to 200).
    - current\_page: The current page number.
    - total\_pages: The total number of pages based on the number of clients and items per page.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of clients, each containing:
    - The client’s ID, username, phone, total rating, and average rating.
  + **pagination** (optional if paginated):
    - total: The total number of clients matching the query.
    - per\_page: The number of clients per page (200 in this case).
    - current\_page: The current page number.
    - total\_pages: The total number of pages based on the number of clients and items per page.
* **Message**: A localized success message: "العملاء الاعلي تقييما للطلبات" (The highest-rated clients for orders).

**🔹 Error Handling**

* If any error occurs, the method will return an error response with:
  + **Error Code**: 422.
  + **Error Message**: The exception’s message will be included.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: The success message confirming the retrieval of the highest-rated clients.
* **Data**:
  + **clients**: A collection of clients with their details (id, username, phone, total\_rating, avg\_rating).
  + **pagination** (if paginated):
    - total: The total number of clients in the result set.
    - per\_page: Number of clients per page.
    - current\_page: The current page.
    - total\_pages: The total pages of results.

**🔹 Error**

* **Error**: If an error occurs, an error code 422 will be returned along with the error message.

**10.17 getHighestRatedReservationsForClient() Method**

### 🔹 Overview

This method retrieves a list of the highest-rated reservations for a specific client. It supports pagination and sorts the reservations by the review score in descending order, showing the highest-rated reservations first.

**🔹 Key Functionality**

**1-Eager Load Review Data**

* The method eagerly loads the review relationship for each reservation to ensure the review details are available when filtering by score.

**2-Filter by Client ID**

* The method filters the reservations by the specific client\_id provided as a parameter to ensure only the reservations related to the given client are considered.

**3-Join with Reviews**

* It performs a leftJoin with the reviews table to pull in the review data for each reservation.

**4-Ensure Reviews Have a Score**

* The method uses whereNotNull('reviews.score') to ensure only reservations with a valid review score are included in the results.

**5-Sorting**

* The reservations are sorted by the review score in descending order (orderByDesc('reviews.score')), ensuring the highest-rated reservations come first.
* Additionally, the reservations are ordered by their creation date in descending order (latest('reservations.created\_at')), so the most recent reservations appear first if multiple reservations have the same score.

**6-Pagination**

* The method supports pagination:
  + It retrieves the reservations with their associated reviews, ordered by the highest review scores.
  + Pagination metadata is returned, including:
    - total: The total number of reservations matching the criteria.
    - per\_page: The number of items displayed per page (set to 10).
    - current\_page: The current page number.
    - total\_pages: The total number of pages based on the number of reservations and items per page.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **reservations**: A collection of the highest-rated reservations for the client, each including the reservation details and review score.
  + **pagination**: Pagination metadata:
    - total: The total number of reservations matching the query.
    - per\_page: The number of reservations per page (10 in this case).
    - current\_page: The current page number.
    - total\_pages: The total number of pages based on the number of reservations and items per page.
* **Message**: A localized success message: "تم استرجاع الطلبات بنجاح" (Reservations have been successfully retrieved).

**🔹 Error Handling**

* If any error occurs, the method will return an error response with:
  + **Error Code**: 422.
  + **Error Message**: The exception’s message will be included.

**🔹 Return Data Structure**

**🔹 Success**

* **Message**: The success message confirming the retrieval of the highest-rated reservations for the client.
* **Data**:
  + **reservations**: A collection of reservations, including details like id, from, to, and the associated review score.
  + **pagination** (if paginated):
    - total: The total number of reservations in the result set.
    - per\_page: Number of reservations per page.
    - current\_page: The current page.
    - total\_pages: The total pages of results.

**🔹 Error**

* **Error**: If an error occurs, an error code 422 will be returned along with the error message.

**10.18 getMostPopularHours() Method**

### 🔹 Overview

This method retrieves statistics for the most popular hours for reservations based on the provided year and month (or just the year) as input. It supports both specific month filtering and year-based filtering with detailed hour-wise reservation data.

**🔹 Key Functionality**

**1-Filter by Year and Month (if provided)**

* **If both year and month are provided**:
  + The method will filter reservations by the specified year and month (whereYear('date', $request->year) and whereMonth('date', $request->month)).
* **If only year is provided**:
  + The method will handle different cases based on the year:
    - If the year is **2024**, the date range will be from **March 1st, 2024** to **December 31st, 2024**.
    - For other years, the method will select reservations between **January 1st** and **December 31st** of that year.

**2-Time Conversion**

* The method uses REGEXP\_REPLACE to adjust the from time field, converting the Arabic AM/PM indicators (ص, م) into English (AM, PM).
* It uses STR\_TO\_DATE and TIME\_FORMAT to parse the time into a standardized format, ensuring it works correctly with both 12-hour and 24-hour time formats.

**3-Group by Hour**

* The reservations are grouped by hour (using groupByRaw("hour")), and the total number of reservations for each hour is counted (COUNT(\*) as total\_reservations).

**4-Filtering and Sorting**

* **Non-null hours**: Only hours that have reservations are included in the result (havingRaw("hour IS NOT NULL")).
* The results are ordered by the total number of reservations in descending order, ensuring the most popular hours appear first.

**5-Data Preparation**

* The results are mapped to a simpler structure with the hour and total reservation count:
  + hour: The specific hour of the day (in the format hh:mm AM/PM).
  + total\_reservations: The total number of reservations for that hour.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **statistics**: A list of hours with the corresponding total reservation count for each hour.
* **Message**: A localized success message: "Hourly reservation statistics."

**🔹 Error Handling**

* If any error occurs, an error response with a message will be returned, though the current implementation does not have explicit error handling for this method.

**10.19 clients\_wallets() Method**

### 🔹 Overview

This method retrieves a list of clients with their wallet operations, filtered based on user role and with pagination options. It allows users with certain roles to access client wallet details.

**🔹 Key Functionality**

**1-User Role Check**

* **Role Validation**: The method checks if the authenticated user ($user) has a role ID of either **8** or **1**. These roles are authorized to access the clients' wallets data.
  + If the user has a valid role, the method proceeds to fetch the data.
  + If the user does not have the required role, an error response with status code 403 (Forbidden) is returned.

**2-Client Filtering**

* **Client Type Filtering**: The method filters clients by type = 0, which represents a specific type of client.
* **Wallet Operations**: The method also eager loads the wallet\_operations relationship for each client, which is assumed to contain data about the client's wallet transactions.

**3-Pagination**

* The method checks the paginate parameter passed in the request:
  + **If paginate is set to 1**: The method retrieves a paginated list of clients (paginate(10)), with pagination metadata included in the response.
  + **If paginate is set to 0**: The method retrieves all clients without pagination (get()).
  + **Default**: If the paginate parameter is not provided, the method defaults to paginating the client list (paginate(10)).

**4-Pagination Metadata**

* **Total**: Total number of clients.
* **Per Page**: Number of clients per page (10).
* **Current Page**: The current page number.
* **Total Pages**: The total number of pages.

**5-Response Structure**

* **Success**: The response includes:
  + wallets: A collection of clients with their wallet operations.
  + pagination: Pagination details (only when paginate is set to 1).
* **Error Handling**: If any exception occurs, the method catches it and returns a 422 status code with the error message.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **wallets**: A collection of client data with their associated wallet operations.
  + **pagination**: Pagination details (when paginate is set to 1).
* **Message**: A localized success message: "Client wallets".

**🔹 Error Response**

* **Forbidden Access**: If the user’s role is not authorized, a 403 error is returned with a message "Forbidden".
* **General Error**: If an exception is thrown, a 422 error is returned with the exception message.

**10.20 representatives\_wallets() Method**

### 🔹 Overview

This method retrieves a list of representatives (clients with type = 1 and status = 1) along with their wallet operations, filtered based on user role and pagination options. Only users with roles 1 or 8 are authorized to access this data.

**🔹 Key Functionality**

**1-User Role Check**

* **Role Validation**: The method checks if the authenticated user ($user) has a role ID of either **8** or **1**. These roles are authorized to access the representatives' wallets data.
  + If the user has a valid role, the method proceeds to fetch the data.
  + If the user does not have the required role, an error response with status code 403 (Forbidden) is returned.

**2-Client Filtering**

* **Client Type**: The method filters clients by type = 1, which represents representatives.
* **Client Status**: It also filters by status = 1, ensuring only active representatives are retrieved.
* **Wallet Operations**: The method eager loads the wallet\_operations relationship for each representative.

**3-Pagination**

* The method checks the paginate parameter passed in the request:
  + **If paginate is set to 1**: The method retrieves a paginated list of representatives (paginate(10)), with pagination metadata included in the response.
  + **If paginate is set to 0**: The method retrieves all representatives without pagination (get()).
  + **Default**: If the paginate parameter is not provided, the method defaults to paginating the list (paginate(10)).

**4-Pagination Metadata**

* **Total**: Total number of representatives.
* **Per Page**: Number of representatives per page (10).
* **Current Page**: The current page number.
* **Total Pages**: The total number of pages.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **wallets**: A collection of representatives' data with their associated wallet operations.
  + **pagination**: Pagination details (only when paginate is set to 1).
* **Message**: A localized success message: "Client wallets".

**🔹 Error Response**

* **Forbidden Access**: If the user’s role is not authorized, a 403 error is returned with a message "Forbidden".
* **General Error**: If an exception is thrown, a 422 error is returned with the exception message.

**10.21 walletOperationsOfClient() Method**

### 🔹 Overview

This method retrieves a list of wallet operations for a specific client, filtered by the client ID and status. It also calculates the total amount for the client’s wallet operations and supports pagination.

Only users with roles 1 or 8 are authorized to access this method. The method also returns pagination metadata and the total sum of wallet operations.

**🔹 Key Functionality**

**1-User Role Check**

* **Role Validation**: The method checks if the authenticated user ($user) has a role ID of either **8** or **1**. These roles are authorized to access the wallet operations data of clients.
  + If the user has a valid role, the method proceeds to fetch the data.
  + If the user does not have the required role, an error response with status code 403 (Forbidden) is returned.

**2-Wallet Operations Filtering**

* **Status**: The method filters wallet operations by status = 1, ensuring only active operations are retrieved.
* **Client ID**: It filters wallet operations based on the provided client\_id, ensuring that only the operations related to the specified client are fetched.

**3-Pagination**

* The method retrieves a paginated list of wallet operations for the given client using paginate(10), which returns 10 items per page.
* **Pagination Metadata**: The response includes metadata about the pagination:
  + **Total**: Total number of wallet operations.
  + **Per Page**: Number of wallet operations per page (10).
  + **Current Page**: The current page number.
  + **Total Pages**: The total number of pages.

**4-Total Amount Calculation**

* The method calculates the total sum of the wallet operations for the specified client (client\_id) by using sum('amount'). This gives the total value of all the client's wallet operations with status = 1.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **operations**: A collection of wallet operations for the client. Each operation includes attributes such as:
    - id: The operation ID.
    - type: The type of operation (deposit, withdrawal, etc.).
    - amount: The amount of the operation.
    - created\_at: The timestamp when the operation was created.
  + **total\_amounts**: The total sum of wallet operations for the client.
  + **pagination**: Pagination metadata for the response.
* **Message**: A localized success message: "Client operations".

**10.22 walletChargingOperationsOfClient() Method**

### 🔹 Overview

This method retrieves wallet charging operations of clients (type 0), with an optional status filter. It supports both pagination and fetching all records, depending on the request. The method also handles filtering by operation type and status.

Only authorized users can access this method. It returns the wallet charging operations along with pagination metadata if needed.

**🔹 Key Functionality**

**1-Filtering by Status**

* If the status is provided in the request, the method filters the wallet operations by that specific status.

**2-Client Type Filtering**

* The method retrieves wallet charging operations only for clients with type = 0.

**3-Operation Type**

* The method specifically filters for wallet operations of type 1, which represent charging operations.

**4-Pagination**

* **With Pagination**: If the request includes paginate=1, the method fetches the charging operations with pagination (10 items per page).
  + Pagination metadata is included in the response:
    - **Total**: Total number of wallet charging operations.
    - **Per Page**: Number of wallet charging operations per page (10).
    - **Current Page**: The current page number.
    - **Total Pages**: Total number of pages.
* **Without Pagination**: If pagination is not requested, all wallet charging operations are fetched without pagination.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **operations**: A collection of wallet charging operations for clients. Each operation includes attributes such as:
    - id: The operation ID.
    - type: The type of operation (charging).
    - amount: The amount of the operation.
    - created\_at: The timestamp when the operation was created.
  + **pagination**: Metadata related to pagination (only if paginate=1).
* **Message**: A localized success message: "Client operations".

**🔹 Error Response**

* If an exception is thrown, a 422 error is returned with the exception message.
* If the user does not have the required role, a 403 response is returned with a message: "Forbidden".

**10.23 client\_cars() Method**

### 🔹 Overview

This method retrieves all cars associated with a client. The method supports filtering by client\_id and pagination, making it easy to fetch car data for a specific client or all clients with the option of paginated results.

The method returns the car data, including pagination metadata if requested.

**🔹 Key Functionality**

**1-Filtering by client\_id**

* If the client\_id is provided in the request, it filters the cars to only include those owned by the specific client (using user\_id).

**2-Pagination**

* **With Pagination**: If the request includes paginate and it is truthy, the method fetches cars with pagination (10 cars per page).
  + Pagination metadata is included in the response:
    - **Total**: Total number of cars.
    - **Per Page**: Number of cars per page (10).
    - **Current Page**: The current page number.
    - **Total Pages**: Total number of pages.
* **Without Pagination**: If pagination is not requested, all cars are retrieved without pagination.

**3-Response Structure**

* **Success**: The response includes:
  + **cars**: A collection of cars.
  + **pagination** (only if pagination is requested): Metadata for pagination.
* **Error Handling**: In case of an exception, the method returns a 422 error with the exception message.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **cars**: A collection of car data. Each car includes attributes such as:
    - id: The car's ID.
    - make: The car's make (e.g., Toyota).
    - model: The car's model.
    - year: The car's year of manufacture.
    - user\_id: The ID of the client who owns the car.
    - created\_at: The timestamp when the car was created.
  + **pagination**: Pagination metadata (only if paginate is set to true in the request).
* **Message**: A localized success message: "All cars".

**Error Response**

* If an exception is thrown, a 422 error is returned with the exception message.

**10.24 getMonthlyClientsStats() Method**

### 🔹 Overview

This method calculates and returns the client statistics for a specific month. It provides the following for each day in the specified month:

* **Total Clients**: Number of clients who registered on that specific day.
* **Total Reservations**: Number of reservations made by those clients on the same day they registered.
* **Reservation Ratio**: The ratio of reservations to the total number of clients for each day.

The statistics are grouped by day, and the response contains data for every day of the specified month, even if no clients or reservations were made on certain days.

**🔹 Key Functionality**

**1-Fetch Client and Reservation Data**

* A LEFT JOIN is used between the clients and reservations tables to fetch the statistics for each day.
  + The clients.created\_at date is compared with the reservations.created\_at date to group reservations by the day clients registered.
  + Only reservations with a status that is not equal to 6 are considered.

**2-Filtering by Year and Month**

* The query is filtered by clients.created\_at for both the year and month provided in the request.

**3-Initialize Data for Each Day of the Month**

* An array is created to store the statistics for each day of the month.
* The array is initialized with default values:
  + **Total Clients**: Set to 0.
  + **Total Reservations**: Set to 0.
  + **Reservation Ratio**: Set to 0.

**4-Merge Client Stats into Data Array**

* The statistics for each day are merged into the data array.
* The **reservation ratio** is calculated as (total\_reservations / total\_clients) \* 100, rounded to two decimal places.

**5-Return Data**

* The data array is returned in a sequential format, containing the statistics for each day of the month.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **statistics**: A list of statistics for each day in the specified month.
    - For each day, the following fields are present:
      * day: The day of the month.
      * total\_clients: The number of clients registered on that day.
      * total\_reservations: The number of reservations made by those clients on that day.
      * reservation\_ratio: The ratio of reservations to clients on that day, represented as a percentage.
* **Message**: A localized success message: "Clients statistics for the month."

**🔹 Error Response**

* If an error occurs, the method returns a 422 error with the exception message.

**10.25 getClientsRegisteredInDay() Method**

### 🔹 Overview

This method retrieves the clients who were registered on a specific day, along with the count of reservations made by each client on that same day. The method provides pagination and handles errors gracefully.

**🔹 Key Functionality**

**1-Fetch Clients Registered on a Specific Day**

* The query fetches clients that were registered on the day provided in the request. This is done by using the whereDate method, which compares the created\_at field with the provided year, month, and day.

**2-Count Reservations for Each Client on the Same Day**

* Using the withCount method, the query counts the number of reservations for each client on the same day they were registered. This is done by applying the whereDate filter to the reservations.created\_at field.

**3-Pagination**

* The result is paginated with 10 items per page by default.
* The pagination data includes:
  + total: Total number of clients.
  + per\_page: Number of clients per page.
  + current\_page: Current page number.
  + total\_pages: Total number of pages.

**4-Return Data**

* The method returns the list of clients along with the count of reservations for each client, paginated. The ClientResource is used to format the client data.
* The response includes the pagination information for easy navigation of pages.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A paginated list of clients who were registered on the specified day.
    - Each client will include a reservations\_count\_of\_some\_day field, which contains the count of reservations they made on the same day they registered.
  + **pagination**: The pagination data for the results.
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.
* **Message**: A localized success message: "All clients registered on the specified day."

**🔹 Error Response**

* If an error occurs, the method returns a 422 error with the exception message.

**10.26 number\_of\_groups() Method**

### 🔹 Overview

This method calculates the number of groups needed based on the total number of clients, where each group contains up to 200 clients. It also provides a custom label for each group, including the date of the last send action for the group if available.

**🔹 Key Functionality**

**1-Fetch Total Number of Clients**

The method doesn't fetch the total number of clients directly but instead calculates the number of groups by determining the highest client ID (max('id')). The total number of clients is indirectly inferred from this.

**2-Group Calculation**

* The total number of clients is divided by a predefined groupSize of 200.
* The number of groups is calculated using ceil() to ensure that any remainder clients still form a new group.

**3-Fetching Last Send Date**

* The method fetches the last send date for each group from the type\_actions table, where the type column represents the group ID.
* The send\_date is formatted into a readable format (Y-m-d).

**4-Group Creation**

* A set of groups is created with the following properties:
  + **id**: Group number.
  + **value**: The label for the group, including the range of client IDs in that group (e.g., "Group 1 (from 1 to 200)") and the last send date if available.

**5-Returning Data**

The method returns the group data in a structured format, along with any associated send date if present. This information is returned via a returnData response, which will include:

* A list of groups.
* Each group will have an ID, a label with the range of clients in that group, and the send date (if available).

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **groups**: An array of groups, each containing:
    - id: Group ID.
    - value: A string description of the group, e.g., "Group 1 (from 1 to 200)", and an additional message about the last send date if available.

**🔹 Error Response**

* If an error occurs, the method returns a 422 error with the exception message.

**10.27 getClientsWithoutReservationsReport() Method**

### 🔹 Overview

This method fetches clients who do not have any reservations and returns the list either with pagination or as a full list, depending on the request. It also filters clients by their type and status before returning the data.

**🔹 Key Functionality**

**1-Check for Pagination**

* The method first checks if the paginate parameter is present in the request and if it's set to 1. If it is, pagination will be applied; otherwise, the entire list of clients will be returned.

**2-Fetching Clients Without Reservations**

* **Using whereDoesntHave('reservations')**: This condition ensures that only clients who do **not** have any associated reservations are retrieved.
* **Additional Filters**:
  + **Client type**: Only clients with a type of 0 are considered.
  + **Client status**: Only clients with a status of 1 are considered.

**3-Handling Pagination**

* If pagination is required:
  + **latest()**: The clients are sorted by their latest creation date.
  + **paginate(200)**: Limits the number of results per page to 200 clients.
  + A pagination object is created with information about the total, current page, items per page, and total pages.
* **If no pagination is required**: All matching clients are fetched with .get().

**4-Returning Data**

* **If paginated**: The response includes the list of clients and pagination details.
* **If not paginated**: The response includes just the list of clients.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of client data that matches the conditions (i.e., clients without reservations, of type 0, and with status 1).
  + **pagination**: If pagination is applied, it includes the following information:
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.

**🔹 Error Response**

* If an error occurs, the method returns a 422 error with the exception message.

**10.28 getClientsWithPackagesReport() Method**

### 🔹 Overview

This method fetches clients who are subscribed to active packages (i.e., packages with a status of 1). It allows for both paginated and non-paginated results based on the request, and filters clients by their type and status before returning the data.

**🔹 Key Functionality**

**1-Check for Pagination**

* The method first checks if the paginate parameter is provided in the request and if it is set to 1. If true, pagination will be applied; otherwise, all clients will be returned as a single result set.

**2-Fetching Clients with Active Package Subscriptions**

* **Using whereHas('packageSubscriptions')**: This condition ensures that only clients who are associated with at least one active package subscription are fetched.
  + **where('status', 1)**: The active status of the package subscription is 1, ensuring only clients with active subscriptions are retrieved.
* **Additional Filters**:
  + **Client type**: Only clients with a type of 0 are included.
  + **Client status**: Only clients with a status of 1 (active clients) are considered.

**3-Handling Pagination**

* If pagination is required:
  + **latest()**: The clients are sorted by their latest creation date.
  + **paginate(200)**: The results are paginated, with a limit of 200 clients per page.
  + A pagination object is created with:
    - total: Total number of clients matching the query.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.
* **If no pagination is required**: All matching clients are fetched using .get().

**4-Returning Data**

* **If paginated**: The response includes both the list of clients and pagination details.
* **If not paginated**: Only the list of clients is returned.

**🔹 Return Response**

**🔹 Success Response**

* **Data**:
  + **clients**: A collection of clients who have an active package subscription, are of type 0, and have a status of 1.
  + **pagination**: If pagination is applied, it includes the following information:
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.

**🔹 Error Response**

* If an error occurs (e.g., a database error), the method returns an error response with a status code 422 and the exception message.

**10.29 getClientsWithoutPackagesReport() Method**

### 🔹 Overview

This method is designed to fetch clients who either:

1. Do not have any package subscriptions.
2. Have package subscriptions, but the subscriptions are inactive (i.e., their status is 0).

It supports both paginated and non-paginated results, depending on the request. It also filters the clients based on their type and status.

### 🔹 Key Functionality

#### 1-****Check for Pagination****

* The method first checks if the paginate parameter is included in the request and if it is set to 1. If true, pagination will be applied. If not, all clients will be returned without pagination.

#### 2-****Fetching Clients without Active Package Subscriptions****

* **Using whereDoesntHave('packageSubscriptions')**: This condition filters clients who do not have any package subscriptions.
* **Using orWhereHas('packageSubscriptions', ...)**: This condition filters clients who have a package subscription, but the status of the subscription is 0 (inactive). This is combined with the whereDoesntHave() condition using the orWhere clause.
* **Additional Filters**:
  + **Client type**: Only clients with a type of 0 are included.
  + **Client status**: Only clients with a status of 1 (active clients) are considered.

#### 3-****Handling Pagination****

* If pagination is required:
  + **latest()**: Clients are ordered by their creation date (latest first).
  + **paginate(200)**: The results are paginated with a maximum of 200 clients per page.
  + A pagination object is created with:
    - total: Total number of clients matching the query.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.
* **If pagination is not required**: All matching clients are fetched using .get().

#### 4-****Returning Data****

* **If paginated**: The response includes both the list of clients and pagination details.
* **If not paginated**: Only the list of clients is returned.

### 🔹 Return Response

#### 🔹 ****Success Response****

* **Data**:
  + **clients**: A collection of clients who either do not have package subscriptions or have inactive subscriptions, and meet the filtering criteria (i.e., type 0 and status 1).
  + **pagination**: If pagination is applied, it includes the following information:
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.

#### 🔹 ****Error Response****

* If an error occurs (e.g., a database error), the method returns an error response with a status code 422 and the exception message.

**10.30 getClientsWithoutReservationsForWeekReport() Method**

### 🔹 Overview

This method is designed to fetch clients who:

1. **Do not have any reservations** in the past week. This is determined by checking if the reservations relationship exists and filtering by the reservation's date.

It supports both **pagination** and **no pagination** (fetching all clients) based on the request. Additionally, it filters clients by type (0) and status (1).

### 🔹 Key Functionality

#### 1-****Check for Pagination****

* The method checks if the paginate parameter is present in the request and if it's set to 1. If true, pagination is applied to the results. If not, all clients are returned.

#### 2-****Fetching Clients without Reservations for the Past Week****

* **Using whereDoesntHave('reservations')**: This condition filters clients who do not have any reservations.
* **Additional Condition on Reservations**:
  + The where clause within the whereDoesntHave ensures that the reservations date is checked. If no reservations are made in the past week, the client is included in the result.
  + **Date Condition**: The query checks if the reservation's date is greater than or equal to the date of 7 days ago (using Carbon::now()->subWeek()->toDateString()).
* **Additional Filters**:
  + **Client type**: Filters clients with a type of 0.
  + **Client status**: Filters clients with a status of 1 (active clients).

#### 3-****Handling Pagination****

* If pagination is required:
  + **latest()**: Orders the clients by their creation date, with the most recent ones appearing first.
  + **paginate(200)**: Limits the results to 200 clients per page.
  + Pagination details are calculated, including:
    - total: The total number of clients.
    - per\_page: The number of clients per page.
    - current\_page: The current page number.
    - total\_pages: The total number of pages.
* **If pagination is not required**: All matching clients are fetched using .get().

#### 4-****Returning Data****

* **If paginated**: Returns the clients along with pagination details.
* **If not paginated**: Returns just the list of clients.

### 🔹 Return Response

#### 🔹 ****Success Response****

* **Data**:
  + **clients**: A collection of clients who have not made any reservations in the past week and meet the filtering criteria (i.e., type 0 and status 1).
  + **pagination**: If pagination is applied, the response includes the pagination details:
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: Total number of pages.

#### 🔹 ****Error Response****

* If any error occurs (such as a database issue or invalid data), the method will catch the error and return:
  + **Error message**: The exception message from the catch block.

**10.31 getClientsWithoutReservationsForMonthReport() Method**

🔹 **Overview**

* + The getClientsWithoutReservationsForMonthReport() method is similar to the previous one, with the difference being that it checks for clients who have not made any reservations in the past **month** instead of the past week.

### 🔹 Key Functionality

1. **Pagination Check**:
   * Similar to previous methods, it checks if the paginate parameter is present in the request and if it's set to 1. If true, pagination is applied. Otherwise, all results are fetched without pagination.
2. **Fetching Clients without Reservations in the Past Month**:
   * **Using whereDoesntHave('reservations')**: This filters clients who do not have any reservations.
   * **Date Condition**:
     + The query checks if the reservation date is **greater than or equal** to the date one month ago (Carbon::now()->subMonth()->toDateString()).
     + This ensures the clients who have made reservations within the last month are excluded.
3. **Client Filters**:
   * **Client type**: Filters clients with a type of 0 (you can adjust based on your specific needs).
   * **Client status**: Filters active clients (status = 1).
4. **Pagination**:
   * **With Pagination**: If pagination is enabled, the results are limited to 200 clients per page. The latest() method orders the results by the most recent clients.
   * **Without Pagination**: If no pagination is needed, all the matching clients are fetched with the .get() method.
5. **Returning the Data**:
   * If **pagination** is used, it returns the clients along with the pagination details (total, per\_page, current\_page, total\_pages).
   * If **pagination** is not used, it returns only the clients.

### 🔹 Return Structure

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

* **Message**: Returns a success message, such as "Clients without reservations for the past month fetched successfully."
* **Data**:
  + **clients**: A collection of clients who have not made any reservations in the last month.
  + **pagination** (only if pagination is applied):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: Total number of pages.

**Error Response**:

* If any error occurs (e.g., database issues), the method will return an error message in the response.

**10.32 getClientsWithoutReservationsForThreeMonthsReport() Method**

🔹 **Overview**

* The getClientsWithoutReservationsForThreeMonthsReport() method is almost identical to the previous methods, but it checks for clients who have not made any reservations in the last **three months** instead of a week or a month.

### 🔹 Key Functionality

1. **Pagination Check**:
   * Checks if the paginate parameter is set in the request and if it's equal to 1. If true, pagination will be applied. Otherwise, all the results are fetched without pagination.
2. **Fetching Clients without Reservations in the Past Three Months**:
   * **Using whereDoesntHave('reservations')**: This filters clients who do not have any reservations.
   * **Date Condition**:
     + The query checks if the reservation date is **greater than or equal** to the date three months ago (Carbon::now()->subMonth(3)->toDateString()).
     + This ensures the clients who have made reservations in the last three months are excluded.
3. **Client Filters**:
   * **Client type**: Filters clients with a type of 0.
   * **Client status**: Filters only active clients (status = 1).
4. **Pagination**:
   * **With Pagination**: If pagination is enabled, the results will be limited to 200 clients per page. The latest() method orders the results by the most recent clients.
   * **Without Pagination**: If pagination is not needed, it fetches all the matching clients with the .get() method.
5. **Returning the Data**:
   * If **pagination** is used, it returns the clients along with pagination details like total, per\_page, current\_page, and total\_pages.
   * If **pagination** is not used, it returns just the clients.

### 🔹 Return Structure

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

* **Message**: A success message indicating that the clients without reservations for the past three months are fetched successfully.
* **Data**:
  + **clients**: A collection of clients who haven't made any reservations in the last three months.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: Total number of pages.

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

* If any error occurs (such as a database issue), the method will return an error response with a code and the error message.

**12.33 getClientsWithoutReservationsForSixMonthsReport() Method**

🔹 **Overview**

The getClientsWithoutReservationsForSixMonthsReport() method is very similar to the previous ones, with the key difference being that this version targets clients who have **not** made any reservations in the **last six months**.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method checks if the paginate parameter is set in the request and if it's equal to 1. If true, pagination will be applied. Otherwise, the results are fetched without pagination.
2. **Fetching Clients without Reservations in the Past Six Months**:
   * **Using whereDoesntHave('reservations')**: Filters clients who do not have any reservations.
   * **Date Condition**:
     + The query checks if the reservation date is **greater than or equal** to six months ago (Carbon::now()->subMonth(6)->toDateString()).
     + This ensures that clients who have made reservations in the last six months are excluded.
3. **Client Filters**:
   * **Client type**: Filters clients with a type of 0.
   * **Client status**: Filters only active clients (status = 1).
4. **Pagination**:
   * **With Pagination**: If pagination is enabled, the results will be limited to 200 clients per page. The latest() method orders the results by the most recent clients.
   * **Without Pagination**: If pagination is not requested, the query will fetch all matching clients.
5. **Returning the Data**:
   * If **pagination** is enabled, it returns the clients along with pagination details such as total, per\_page, current\_page, and total\_pages.
   * If **pagination** is not enabled, it simply returns the list of clients.

### 🔹 Return Structure

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

* **Message**: A success message indicating that the clients without reservations for the past six months are fetched successfully.
* **Data**:
  + **clients**: A collection of clients who haven't made any reservations in the last six months.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: The current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., database issues), the method will return an error response with a code and the error message.

**10.34 getClientsWithoutReservationsForNineMonthsReport() Method**

🔹 **Overview**

* The getClientsWithoutReservationsForNineMonthsReport() method is designed to retrieve clients who have not made any reservations in the past **nine months**. It is similar to the other methods, but specifically targets a **nine-month** period.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is provided in the request and is set to 1. If pagination is enabled, it will return a paginated response; otherwise, it will fetch all the matching clients without pagination.
2. **Fetching Clients without Reservations in the Last Nine Months**:
   * **Using whereDoesntHave('reservations')**: Filters clients who do **not** have any reservations.
   * **Date Condition**:
     + The query checks if the reservation date is **greater than or equal** to **nine months ago** using Carbon::now()->subMonth(9)->toDateString().
3. **Client Filters**:
   * **Client Type**: Filters clients whose type is 0.
   * **Client Status**: Filters clients that are **active** (status = 1).
4. **Pagination**:
   * **With Pagination**: If pagination is enabled, the method limits the results to 200 clients per page.
   * **Without Pagination**: If pagination is not requested, it fetches all the matching clients.
5. **Returning the Data**:
   * If pagination is enabled, it returns the clients along with pagination details such as total, per\_page, current\_page, and total\_pages.
   * If pagination is not enabled, it simply returns the full list of clients.

### 🔹 Return Structure

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

* **Message**: A success message indicating that clients without reservations in the last nine months have been fetched.
* **Data**:
  + **clients**: A collection of clients who haven't made any reservations in the last nine months.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., a database issue), the method returns an error response with an error code and message.

**10.35 getClientsWithoutReservationsForYearReport() Method**

🔹 **Overview**

* The getClientsWithoutReservationsForYearReport() method is designed to retrieve clients who haven't made any reservations in the last **year**. It follows the same structure as the other methods, with a specific focus on a **12-month** period.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is present in the request and is set to 1. If pagination is enabled, it will return the data in pages; otherwise, it will fetch all clients without pagination.
2. **Fetching Clients without Reservations in the Last Year**:
   * **Using whereDoesntHave('reservations')**: Filters clients who do **not** have any reservations.
   * **Date Condition**:
     + The query filters clients where the reservation date is **greater than or equal** to **one year ago** using Carbon::now()->subMonth(12)->toDateString().
3. **Client Filters**:
   * **Client Type**: Filters clients with type equal to 0.
   * **Client Status**: Filters only active clients (status = 1).
4. **Pagination**:
   * **With Pagination**: If pagination is enabled, the method limits the results to 200 clients per page.
   * **Without Pagination**: If pagination is not requested, it fetches all matching clients.
5. **Returning the Data**:
   * If pagination is enabled, it returns the clients along with pagination details such as total, per\_page, current\_page, and total\_pages.
   * If pagination is not enabled, it returns the list of clients without pagination.

### 🔹 Return Structure

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

* **Message**: A success message indicating that clients without reservations in the last year have been fetched.
* **Data**:
  + **clients**: A collection of clients who have not made any reservations in the last year.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., a database issue), the method returns an error response with an error code and message.

**10.36 getClientsWithoutReservationsAtBuraidah() Method**

🔹 **Overview**

* The getClientsWithoutReservationsAtBuraidah() method is designed to fetch clients who have not made any reservations and are located in **Buraidah** (a city in Saudi Arabia). This method supports both paginated and non-paginated responses and filters clients based on their location and reservation status.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is present and set to 1. If pagination is enabled, it will return the data in pages; otherwise, it fetches all clients without pagination.
2. **Fetching Clients without Reservations in Buraidah**:
   * **Using whereHas('locations')**: Filters clients who have a location with a name like "بريدة" (the Arabic name for Buraidah).
   * **Using whereDoesntHave('reservations')**: Filters clients who have **no reservations**.
   * **Client Type and Status**: The method filters clients where:
     + type = 0 (assuming this is the type of client you're interested in).
     + status = 1 (active clients).
3. **Pagination**:
   * **With Pagination**: If pagination is enabled, it limits the results to 200 clients per page.
   * **Without Pagination**: If pagination is not requested, it fetches all matching clients.
4. **Returning the Data**:
   * If pagination is enabled, the method returns the clients along with pagination details, such as total, per\_page, current\_page, and total\_pages.
   * If pagination is not enabled, it returns the list of clients without pagination.

### 🔹 Return Structure

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

* **Message**: A success message indicating that clients without reservations in Buraidah have been fetched.
* **Data**:
  + **clients**: A collection of clients located in Buraidah who have not made any reservations.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.

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

* If an error occurs (e.g., a database issue), the method returns an error response with an error code and message.

**10.37 getClientsSendGift() Method**

🔹 **Overview**

* The getClientsSendGift() method is designed to fetch clients who have sent gifts. It allows for both paginated and non-paginated responses and filters clients based on their type, status, and whether they have associated gift-sending information.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is set to 1. If so, it will return a paginated response with up to 200 clients per page. If pagination is not requested, it fetches all matching clients without pagination.
2. **Fetching Clients Who Have Sent Gifts**:
   * **Using whereHas('giftsSender')**: Filters clients who have an associated gift-sender relationship. This means the clients must have sent at least one gift.
   * **Client Type and Status**: Filters clients where:
     + type = 0 (presumably active clients).
     + status = 1 (active clients).
3. **Pagination**:
   * **With Pagination**: When pagination is requested, the query limits the number of clients to 200 per page.
   * **Without Pagination**: If pagination is not enabled, all matching clients are retrieved.
4. **Returning the Data**:
   * If pagination is enabled, the method returns the clients along with pagination details like total, per\_page, current\_page, and total\_pages.
   * If pagination is not enabled, it returns the list of clients without pagination.

### 🔹 Return Structure

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

* **Message**: A success message indicating that clients who sent gifts have been fetched.
* **Data**:
  + **clients**: A collection of clients who have sent gifts.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., a database issue), the method returns an error response with an error code and message.

**10.38 getClientsReceiveGift() Method**

🔹 **Overview**

* The getClientsReceiveGift() method is designed to fetch clients who have received gifts. It supports both paginated and non-paginated responses, and filters clients based on specific conditions such as type, status, and whether they have received gifts.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is set to 1. If so, it will return a paginated response with a maximum of 200 clients per page. If pagination is not requested, it fetches all matching clients without pagination.
2. **Fetching Clients Who Have Received Gifts**:
   * **Using whereHas('giftsReceive')**: Filters clients who have an associated giftsReceive relationship, meaning these clients have received at least one gift.
   * **Client Type and Status**: Filters clients where:
     + type = 0 (active clients).
     + status = 1 (active clients).
3. **Pagination**:
   * **With Pagination**: When pagination is requested, the query limits the number of clients to 200 per page.
   * **Without Pagination**: If pagination is not enabled, all matching clients are retrieved.
4. **Returning the Data**:
   * If pagination is enabled, the method returns the clients along with pagination details like total, per\_page, current\_page, and total\_pages.
   * If pagination is not enabled, it returns the list of clients without pagination.

### 🔹 Return Structure

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

* **Message**: A success message indicating that clients who have received gifts have been fetched.
* **Data**:
  + **clients**: A collection of clients who have received gifts.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., a database issue), the method returns an error response with an error code and message.

**10.39 getClientsOrderedOnce() Method**

🔹 **Overview**

* The getClientsOrderedOnce() method is designed to retrieve clients who have placed exactly one reservation. This function allows for both paginated and non-paginated results depending on the request parameters.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is set to 1. If pagination is requested, it will return a paginated response with up to 200 clients per page.
   * If pagination is not requested, it retrieves all matching clients without pagination.
2. **Fetching Clients with One Reservation**:
   * **Using whereHas('reservations')**: This filters clients who have associated reservations.
   * **Query inside whereHas**: The query groups reservations by client\_id and filters for clients who have exactly **one reservation**:
     + groupBy('client\_id'): Groups reservations by the client.
     + havingRaw('COUNT(id) = 1'): Ensures only clients with exactly one reservation are returned.
   * **Client Type and Status**: Filters clients based on:
     + type = 0 (active clients).
     + status = 1 (active clients).
3. **Pagination**:
   * **With Pagination**: If the paginate parameter is present and set to 1, it limits the results to 200 clients per page.
   * **Without Pagination**: If pagination is not enabled, it returns all matching clients in a single response.
4. **Returning the Data**:
   * If pagination is enabled, the response will include details of the pagination (total, per\_page, current\_page, total\_pages).
   * If pagination is not enabled, the response will simply return the list of clients.

### 🔹 Return Structure

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

* **Message**: A success message indicating that the clients who have ordered once have been fetched.
* **Data**:
  + **clients**: A collection of clients who have placed exactly one reservation.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.

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

* If an error occurs (e.g., a database issue), the method will return an error response with an error code and message.

**10.40 getClientsOrderedFive() Method**

🔹 **Overview**

* The getClientsOrderedFive() method is designed to retrieve clients who have placed **at least five reservations**. Like the previous methods, this function supports both paginated and non-paginated results depending on the request parameters.

### 🔹 Key Functionality

1. **Pagination Check**:
   * The method first checks if the paginate parameter is set to 1. If pagination is requested, it returns a paginated response with up to 200 clients per page.
   * If pagination is not requested, it retrieves all matching clients without pagination.
2. **Fetching Clients with at Least Five Reservations**:
   * **Using whereHas('reservations')**: Filters clients who have associated reservations.
   * **Query inside whereHas**: The query groups reservations by client\_id and filters for clients who have **at least five reservations**:
     + groupBy('client\_id'): Groups reservations by the client.
     + havingRaw('COUNT(id) >= 5'): Ensures only clients with five or more reservations are returned.
   * **Client Type and Status**: Filters clients based on:
     + type = 0 (active clients).
     + status = 1 (active clients).
3. **Pagination**:
   * **With Pagination**: If the paginate parameter is present and set to 1, it limits the results to 200 clients per page.
   * **Without Pagination**: If pagination is not enabled, it returns all matching clients in a single response.
4. **Returning the Data**:
   * If pagination is enabled, the response will include details of the pagination (total, per\_page, current\_page, total\_pages).
   * If pagination is not enabled, the response will simply return the list of clients.

### 🔹 Return Structure

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

* **Message**: A success message indicating that the clients who have ordered at least five times have been fetched.
* **Data**:
  + **clients**: A collection of clients who have placed five or more reservations.
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., a database issue), the method will return an error response with an error code and message.

**10.41 targeting\_report() Method**

🔹 **Overview**

* The targeting\_report() method is designed to generate a detailed report of clients based on various criteria, such as completed reservations, pending reservations, subscription count, gift sending/receiving, wallet balance, and registration date. It supports both paginated and non-paginated results.

### 🔹 Key Functionality

1. **Eager Loading Counts**:
   * The method uses the withCount() method to load the counts of related records:
     + completed\_reservations\_count: Counts reservations with status 3 (completed).
     + completed\_reservations\_with\_coupon: Counts completed reservations that have a coupon.
     + pending\_reservations\_count: Counts reservations that are not completed (status != 3).
     + subscription\_count: Counts active package subscriptions (status = 1).
     + sent\_gifts\_count: Counts the number of gifts sent by the client.
     + received\_gifts\_count: Counts the number of gifts received by the client.
2. **Filtering Based on Request Parameters**:
   * **Completed Reservations Count**: Filters clients based on the count of completed reservations.
   * **Completed Reservations with Coupon**: Filters clients based on the count of completed reservations that have a coupon.
   * **Pending Reservations Count**: Filters clients based on the count of pending reservations.
   * **Subscription Count**: Filters clients based on the count of active subscriptions.
   * **Wallet Balance**: Filters clients based on wallet balance range (wallet\_balance\_from and wallet\_balance\_to).
   * **Sent and Received Gifts Count**: Filters clients based on the count of gifts they have sent or received.
   * **Date Range**: Filters clients based on the registration date range (start\_date and end\_date).
3. **Pagination**:
   * The method checks if pagination is enabled (paginate = 1). If so, it returns a paginated list of clients with a maximum of 200 clients per page.
   * It includes pagination details (total, per\_page, current\_page, total\_pages) in the response.
   * If pagination is not enabled, it fetches all the matching clients without pagination.
4. **Returning the Data**:
   * If pagination is enabled, the response includes the pagination data along with the client targeting report.
   * If pagination is not enabled, the response includes just the client targeting report.

### 🔹 Return Structure

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

* **Message**: A success message indicating the targeting report has been fetched.
* **Data**:
  + **targeting\_report**: A collection of clients with the calculated counts for each category (e.g., completed reservations, sent gifts, etc.).
  + **pagination** (only if pagination is enabled):
    - total: Total number of clients.
    - per\_page: Number of clients per page.
    - current\_page: Current page number.
    - total\_pages: Total number of pages.

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

* If an error occurs (e.g., database issue, invalid parameters), the method will return an error response with an error code and message.

**10.42 allReservayionsDataReport() Method**

🔹 **Overview**

* The allReservayionsDataReport() method is designed to generate a report of **reservations** based on specific date ranges, and it can be customized to filter by a date range specified in the request or default to the current month's start and end.

### 🔹 Key Functionality

1. **Date Range Handling**:
   * The method defaults to fetching reservations for the **current month** by determining the **start** (now()->startOfMonth()) and **end** (now()->endOfMonth()) of the current month.
   * If the user provides custom date ranges in the request (from and to), the method adjusts the start and end dates accordingly using **Carbon** to parse and format the date strings.
2. **Filtering Reservations**:
   * The method fetches **reservations** with the status 3 (likely indicating completed reservations).
   * It then filters the results using the **whereBetween()** method to select records where the date field falls between the computed start and end dates (either the current month or the ones provided by the user).
3. **Returning the Data**:
   * The method returns the **reservations** in a collection, wrapped in a resource format (ReservationResource::collection($reservations)).
   * The response includes the data along with a success message in the language of the API (using the \_\_('api.allReservayionsDataReport') translation).

### 🔹 Return Structure

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

* **Message**: A success message indicating the report has been successfully fetched.
* **Data**:
  + **reservations**: A collection of reservations filtered by the date range and status.

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

* If there is an issue (e.g., invalid dates, database query failure), the method will return an error response with an error code and message.

1. **Company Controller**

**11.1 all(Request $request) Method**

### 🔹 Overview

Retrieves all companies from the database, with optional support for pagination. The response is returned as a structured collection using the CompanyResource.

### 🔹 Key Functionality

#### 1-Check for Pagination Request

The method checks if the request includes a paginate parameter:

* If paginate = 1, it retrieves paginated results (10 per page).
* If paginate = 0, it retrieves **all companies** without pagination.
* If paginate is not provided, it **defaults** to pagination (10 per page).

#### 2-Query Companies

The companies are fetched using Company::latest(), ordering them by created\_at in **descending** order (most recent first).

#### 3-Format the Response

The companies are wrapped in a CompanyResource collection for consistent API structure.  
When paginated, a pagination object is added containing:

* total: total number of companies
* per\_page: results per page
* current\_page: current page number
* total\_pages: total number of pages

### 🔹 Return Response

🔹 **Success:**  
Returns a structured response containing the list of companies and, if paginated, pagination metadata.  
The data is wrapped using CompanyResource::collection($companies).  
The response also includes a localized message from \_\_('api.comapny\_all').

🔹 **Error Handling:**  
If an exception occurs (e.g. database error), it is caught by the catch block.  
Returns a 422 error response using returnError() along with the exception message.

**11.2 get($company\_id) Method**

### 🔹 Overview

Retrieves a single company record based on the provided $company\_id.

### 🔹 Key Functionality

#### 1-Find Company by ID

Uses Company::find($company\_id) to attempt to retrieve the company record from the database.

#### 2-Check If Company Exists

* If the company **exists**, it is returned wrapped in a CompanyResource.
* If the company **does not exist**, it returns a 404 error with a localized message from \_\_('api.company\_not\_found').

### 🔹 Return Response

🔹 **Success:**  
If the company is found:

* Returns the data wrapped in a data key.
* Uses CompanyResource to format the returned company.
* Includes a localized success message from \_\_('api.company\_found').

🔹 **Error Handling:**

* If the company is **not found**, returns a 404 error with a relevant message.
* If any **exception** is thrown during execution (e.g. database connection issue), it catches the error and returns a 422 error with the exception message.

**11.3 add(storeCompanyRequest $request) Method**

### 🔹 Overview

Adds a new company record to the database using the data provided in the validated storeCompanyRequest.

### 🔹 Key Functionality

#### 1-Create New Company

* Uses Company::create([...]) to insert a new company into the database.
* Fields included: name, email, phone, lat, and long.
* All inputs are expected to be validated by the storeCompanyRequest.

#### 2-Wrap in Resource

* Once created, the new company is formatted using CompanyResource for a structured API response.

### 🔹 Return Response

🔹 **Success:**

* Returns the created company wrapped inside a data key.
* Uses CompanyResource to structure the response.
* Includes a localized success message from \_\_('api.company\_added').

🔹 **Error Handling:**

* If any exception occurs (e.g. database issue, unexpected failure), it is caught and returned with:
  + HTTP status code 422
  + Exception message as the error content.

**11.4 update(updateCompanyRequest $request) Method**

### 🔹 Overview

Updates an existing company's details based on the provided request data.

### 🔹 Key Functionality

#### 1-Find Company by ID

* The method attempts to locate the company using the id provided in the request.
* If the company does not exist, a 404 error is returned with a localized message from \_\_('api.company\_not\_found').

#### 2-Update Company Data

* If the company exists, it updates the following fields:
  + name
  + email
  + phone
  + lat
  + long
* The data is updated using Laravel’s update() method.

#### 3-Resource Wrapping

* The updated company data is returned using CompanyResource for a structured API response format.

### 🔹 Return Response

🔹 **Success:**

* Returns the updated company wrapped in a data key using CompanyResource.
* Includes a localized success message from \_\_('api.company\_updated').

🔹 **Error Handling:**

* If the company is not found, returns a 404 error with a localized message.
* If an exception occurs during the update process, returns a 422 error with the exception message.

**11.5 destroy($company\_id) Method**

### 🔹 Overview

Deletes a specific company based on the provided company\_id.

### 🔹 Key Functionality

#### 1-Find Company by ID

* The method looks up the company using Company::find($company\_id).
* If the company does not exist, it returns a 404 error with a localized message from \_\_('api.company\_not\_found').

#### 2-Delete Company

* If the company is found, it is deleted using Laravel’s delete() method.

### 🔹 Return Response

🔹 **Success:**

* Returns a success message using returnSuccess() with the localized message from \_\_('api.company\_deleted').

🔹 **Error Handling:**

* If the company is not found, a 404 error response is returned.
* If any exception occurs during the deletion process, it is caught and a 422 error is returned with the exception message.

1. **Complaint Controller**
   1. **getComplaints(Request $request) Method**

### 🔹 Overview

Retrieves complaints based on the role of the authenticated user and whether pagination is requested or not.

### 🔹 Key Functionality

1-**Filter by Admin Role**  
If the authenticated user has a role ID of 7, the method filters the complaints to only include those associated with that specific admin.

2-**Handle Pagination**

* If pagination is requested (paginate = 1), the method retrieves the complaints in a paginated format, including pagination metadata such as total items, items per page, current page, and total pages.
* If pagination is not requested (paginate = 0), the method retrieves all complaints without pagination.

### 🔹 Return Response

🔹 **Success:**

* Returns the complaints wrapped inside a data key using a structured resource collection.
* If paginated, it includes pagination details in the response.
* Includes a success message indicating that the data was retrieved successfully.

🔹 **Error Handling:**

* If any error occurs, the method catches the exception and returns a JSON response with a 403 status code and the error message.
  1. **changeStatus(Request $request) Method**

### 🔹 Overview

Updates the status of a specific complaint to indicate it has been resolved.

### 🔹 Key Functionality

1-**Find Complaint by ID**  
The method attempts to locate a complaint using the provided complaint\_id. If it does not exist, it returns an error message indicating that the complaint was not found.

2-**Update Complaint Status**  
If the complaint is found, the method sets its status to 1, representing that the complaint has been marked as solved.

### 🔹 Return Response

🔹 **Success:**

* Returns a success response using a localized message to indicate that the complaint status has been successfully changed.

🔹 **Error Handling:**

* If the complaint is not found, returns a 404 error with a relevant localized message.
* If any other error occurs, it returns a 422 error with a message describing what went wrong.
  1. **destroy(Request $request) Method**

### 🔹 Overview

Deletes a specific complaint from the system based on the provided complaint\_id.

### 🔹 Key Functionality

1-**Find Complaint by ID**  
The method checks for the existence of the complaint using the complaint\_id. If it does not exist, an error response is returned indicating that the complaint was not found.

2-**Delete Complaint**  
If the complaint exists, the method proceeds to delete it from the database.

### 🔹 Return Response

🔹 **Success:**

* Returns a success response using a localized message to confirm the complaint was successfully deleted.

🔹 **Error Handling:**

* Returns a 404 error if the complaint is not found.
* If any exception occurs during the process, a 422 error is returned with a descriptive message of the error.

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

### 🔹 Overview

Retrieves a list of contact messages, with the option to filter by status and paginate the results.

### 🔹 Key Functionality

1-**Filter by Status**  
The method checks if a status filter is provided in the request and applies it to filter the contacts by their type (status).

2-**Pagination**

* If the paginate parameter is set to 1, the results are paginated, returning a paginated list of contacts with metadata (total, per page, current page, and total pages).
* If paginate is set to 0, all the contact messages are returned without pagination.
* If no paginate parameter is provided, the default behavior is pagination with 10 items per page.

### 🔹 Return Response

🔹 **Success:**

* The success response includes the list of contacts formatted using ContactResource, along with pagination information (if applicable), and a localized success message.

🔹 **Error Handling:**

* If an exception occurs during the process, a 422 error is returned with a descriptive error message.

**13.2 get($id) Method**

### 🔹 Overview

Retrieves a specific contact message by its unique identifier (id).

**🔹 Key Functionality**

1-**Find Contact by ID**  
The method attempts to find a contact message using the provided id with ContactMessage::find($id).

2-**Check if Contact Exists**  
If the contact is found, it is formatted using ContactResource. If the contact does not exist, it returns a 404 error indicating that the contact was not found.

**🔹 Return Response**

**🔹 Success:**  
When the contact is found, it returns the contact data wrapped in a data key, formatted using the ContactResource, and a localized success message.

**🔹** **Error Handling:**  
If an error occurs or the contact is not found, it returns a 422 error with the exception message or a 404 error when the contact is not found.

**13.3 changeStatus($contact\_id, Request $request) Method**

### 🔹 Overview

Changes the status (or type) of a specific contact message based on the provided contact\_id and the new status from the request.

**🔹 Key Functionality**

1-**Find Contact by ID**  
The method attempts to find the contact message using the provided contact\_id with ContactMessage::find($contact\_id).

2-**Check if Contact Exists**  
If the contact is found, it updates the type field with the new status value from the request. If the contact does not exist, it returns a 404 error indicating that the contact was not found.

3-**Update Contact Status**  
If the contact is found and successfully updated, the new status is saved, and a success message is returned.

**🔹 Return Response**

**🔹 Success:**  
When the contact status is successfully updated, it returns a success response with a localized message indicating the status change.

**🔹** **Error Handling:**  
If an error occurs or the contact is not found, it returns a 422 error with the exception message or a 404 error when the contact is not found.

**13.4 destroy($id) Method**

### 🔹 Overview

Deletes a specific contact message based on the provided id.

**🔹 Key Functionality**

1-**Find Contact by ID**  
The method attempts to find the contact message using the provided id with ContactMessage::find($id).

2-**Check if Contact Exists**  
If the contact is found, it proceeds to delete the contact from the database. If the contact does not exist, it returns a 404 error indicating that the contact was not found.

3-**Delete Contact**  
If the contact is found and successfully deleted, a success message is returned indicating that the contact has been deleted.

**🔹 Return Response**

**🔹 Success:**  
When the contact is successfully deleted, it returns a success response with a localized message confirming the deletion.

**🔹** **Error Handling:**  
If an error occurs or the contact is not found, it returns a 422 error with the exception message or a 404 error when the contact is not found.