Project documentation

1. **Application**
   1. **Additional Service Controller**
      1. Index() **method**

This method retrieves **additional services (type = 1)** based on the selected city. If no city is provided, a default city (id = 5) is used.

🔹 Steps & Logic

1. **Identify the City**

* If city\_name is provided → Looks up the city in both English and Arabic.
* If city\_id == 2 → Finds city by ID.
* If nothing is provided → Defaults to city\_id = 5.

1. **Validate the City**

* If the city is **not found**, return 403 - المدينة غير موجودة (City not found). Filters services

1. **Filter Services Based on service\_id (if provided)**

 Find the service by ID.

 If the service exists, retrieve:

* Services with **matching availability.**
* Services of **type = 1** (Additional Services).

1. **Retrieve All Additional Services (if no service\_id is provided)**
   * Get all **active** services in the selected city **where type = 1**.
2. **Format & Return the Services**

 Apply $this->formatServices($services, $city).

 Return the formatted services as a JSON respons

* + 1. show() Method

This method retrieves a **single additional service (type = 1)** based on the provided service ID and city.

🔹 Steps & Logic

1. **Validate the Service ID**

Ensure the provided id exists in the services table.

If validation fails, return 403 with validation errors.

**2. Identify the City**

Default city name → "بريدة".

If city\_name is provided → Search by name (English & Arabic).

If the city is not found, return 403 - المدينة غير موجودة (City not found).

**3. Retrieve the Service**

Find the service by ID.

Ensure it is active and of type = 1 (Additional Service).

4**. Format & Return the Service**

Apply $this->formatService($service, $city).

Return the formatted service in a JSON response.

1.1.3 formatServices() Method

This **private method** processes a collection of additional services and formats them for the response.

### ****🔹 Steps & Logic****

1. **Initialize an empty array** → $services\_array
2. **Loop through each service** in $services:
   * Call $this->formatService($service, $city).
   * Add the formatted service to $services\_array.
3. **Return the formatted services array.**
   * 1. **formatService() Method**

This **private method** formats a single service object before returning it in the API response.

### ****🔹 Steps & Logic****

1. **Service ID & Translations**
   * Retrieves **ID** → $service->id
   * Gets **title** and **description** in the user's locale (app()->getLocale()).
2. **Cost Calculation (City-Based)**
   * Fetches the service's **cost for the given city** from the cityservicesCost relationship.
   * If the cost is found, it assigns the value; otherwise, it defaults to 0.
3. **Tax Calculation**
   * Uses cost\_without\_tax from the service.
   * Computes **tax amount** using:  
     tax = total cost - cost without tax.
4. **Service Image Handling**
   * If an image exists, return its URL.
   * Otherwise, return a **default image** (uploads/defaults/logo.png).
   1. **Application Controller**
      1. Index() method

This method retrieves the homepage data, including application information, active sliders, available services, and packages based on the user's city.

🔹 Steps & Logic

1- City Selection & Lookup

* **Default City**: If no city is provided, it defaults to **بريدة**.
* **Lookup by Name**:
  + - Searches for the city in City where name->en or name->ar matches the provided city name.
  + **Lookup by ID**:
    - If city\_id == 2, retrieves the city with ID 2.
    - Otherwise, defaults to city ID 5.
  + **Validation**: If no city is found, the method returns a 403 error:

"المدينة غير موجودة"

2- Retrieving Sliders for the City

* + -Fetches active sliders (status = 1) associated with the city.
  + Each slider contains:
  + **Title & Description** → Translated based on app()->getLocale().
  + **Image** → Returns image URL if available, otherwise an empty string.

3-Fetching Application Details

* + Retrieves the first application record from Application::first().
  + Extracts and formats:
    - Title, Description (localized).
    - Contact Information: Emails, Phone Numbers.
    - Social Media Links (Facebook, Twitter, Instagram, etc.).
    - App Details: Versions, Download URLs.
    - Location (Latitude, Longitude).
    - Logo Handling: Returns URL if present, otherwise empty.
  + **Popup Display**:
    - If city == **بريدة**, it shows display\_pop\_up and pop\_up\_image.
    - Otherwise, it shows display\_pop\_up\_2 and pop\_up\_image\_2.

4-Checking for Pending Reviews

* + **Checks if the authenticated user has a completed reservation (**status = 3**).**
  + If found, verifies whether a review already exists for it.
    - **If not reviewed**, returns reservation\_id for frontend to prompt a review.
    - Otherwise, reservation\_id = null.

5-Fetching Client Data

* + Retrieves authenticated client.
  + Counts the number of completed reservations (status = 3).

6-Fetching & Formatting Packages

* + **Primary Package Selection**:
    - Finds the first package where for\_reservations\_from and for\_reservations\_to match the client’s completed reservations (check if there is a custom package for the client based on the reservations count ).
    - Ensures the package has services and is available for the
    - city.
  + **Additional Packages**:
    - Retrieves all packages that are available for any user (for\_reservations\_from = 0, for\_reservations\_to = 0).
    - Filters packages available for the selected city.
    - **Sorting**: Orders by cost (asc).
  + **Subscription Check**:
    - Determines if the client has an **active subscription** to the package.
    - Adds "subscribed" => 1 if the client is subscribed, otherwise 0.

7-Fetching & Formatting Services

* + Filters services that:
    - Belong to the selected city (cityservicesCost).
    - Have type = 0 and status = 1.
    - Are sorted from the latest available.
  + Calls **formatServices()** method to structure service data before returning.

### 1.2.2 application() Method

This method retrieves the application’s configuration and information details from the database and returns them as a JSON response. This includes localized titles and descriptions, contact information, social media links, version details, maintenance settings, and app review links.

#### ****🔹 Steps & Logic****

1. **Retrieve Application Data**
   * Fetch the first record from the Application table using Application::first().
   * If no record is found, an empty application info array is returned.
2. **Format Application Information**  
   If an application record exists, build the $application\_info array containing:
   * **Localized Text**:
     + **Title**: Obtained by translating the application's title based on the current locale (app()->getLocale()).
     + **Description**: Similarly, the application's description is localized.
   * **Contact Details**:
     + **Emails**: email\_1 and email\_2.
     + **Phone Numbers**: phone\_1 and phone\_2.
   * **Social Media Links**:
     + Facebook, WhatsApp, Twitter, Instagram, Snapchat, YouTube, LinkedIn, TikTok.
   * **Geographic Coordinates**:
     + **Latitude (lat)** and **Longitude (lng)**.
   * **Logo**:
     + If the application has a logo, its URL is generated; otherwise, an empty string is returned.
   * **Version & Update Information**:
     + **New Version**, **Current Version**, **Current Version for iOS**.
     + **Android URL** and **iOS URL** for downloads.
   * **Maintenance Settings**:
     + **Maintenance Mode** flag, **Maintenance Message**, and **Update Message**.
   * **App Review Links**:
     + **Android App Review** link.
     + **iOS App Review** link.

### 1.2.3 TermsAndConditions() Method

This method retrieves the Terms and Conditions from the database and returns them as a JSON response. The description is localized based on the current language.

#### ****🔹 Steps & Logic****

1. **Retrieve Terms and Conditions Data**
   * Fetch the first record from the Term table using Term::first().
   * If no record is found, an empty array is returned.
2. **Format the Response**
   * If a record exists, retrieve the description and translate it according to the current locale (app()->getLocale()).
3. **Return Response**
   * The method returns a JSON response with the key 'terms\_and\_conditions', containing the localized description, and an HTTP status code 200.

### 1.2.4 sliders() Method

This method retrieves all active sliders (status = 1), formats their data, and returns them in a JSON response.

#### ****🔹 Steps & Logic****

1. **Retrieve Active Sliders**
   * Fetch all sliders from the Slider model where status = 1.
   * Sort them in descending order (latest()).
2. **Format Each Slider Object**
   * Convert each slider into an array using map().
   * Retrieve title and description in the current locale (app()->getLocale()).
   * If an image exists, return its URL; otherwise, return an empty string.
3. **Return Response**
   * Return the formatted list of sliders in a JSON response with an HTTP status code 200.

### 1.2.5 contact() Method

This method handles user contact messages by validating input data, storing the message, and returning a response.

#### ****🔹 Steps & Logic****

1. **Validate Request Data**
   * The method ensures that the following fields are required:
     + username
     + email
     + subject
     + message
   * If validation fails, return an error response with status 403.
2. **Store the Contact Message**
   * Create a new record in the ContactMessage model using the request data.
3. **Return Success Response**
   * Return a JSON response with a success message (message\_sent from language files) and status 200.

### 1.2.6 instructions() Method

This method retrieves the application's instructions and reservation terms, handling potential errors gracefully.

#### ****🔹 Steps & Logic****

1. **Retrieve Application Data**
   * Fetches the first record from the Application model.
   * Retrieves the instructions and reservation\_terms fields.
2. **Return Success Response**
   * Returns the retrieved data in a structured JSON response.
3. **Handle Errors**
   * Catches any exceptions (\Throwable $error) and returns an error response with status 422.

### 1.2.7 getarea() Method

This method retrieves a list of active areas, including their associated city names, and returns the data in a structured response.

#### ****🔹 Steps & Logic****

1. **Retrieve Active Areas**
   * Fetches all areas where status = 1.
   * Uses map() to format each area’s details, including:
     + id, name, city\_id, and city\_name.
   * If an area has no associated city, city\_name is set to an empty string ("").
2. **Return Success Response**
   * Returns the formatted area data in a structured JSON response with a localized success message (\_\_('api.getArea')).
3. **Handle Errors**
   * Catches any exceptions (\Throwable $error) and returns an error response with status 422.
   1. Appointment Controller

### 1.3.1 dayAppointments() Method Overview

This method retrieves available appointment slots for a given date and filters them based on various conditions, such as whether the reservation system works based on areas or the nearest representative for client.

#### ****🔹 Steps & Logic****

1. **Retrieve Application Settings**
   * The function starts by fetching the application settings from the Application model to determine how reservations should be handled.
2. **Check for** from\_reservation **Flag** to know where the request coming from (1 for make reservation api)
   * If from\_reservation is set to 1, it checks whether the system sends reservations based on areas (sending\_reservations\_to\_representatives == 1).
   * Calls either workHoursUsingAreas() or workHours() accordingly.
   * Removes duplicate appointments using removeDuplicateAppointments() and returns the data.
3. **Validate Date Input & Check Working Days**
   * If the request includes a valid date, it ensures:
     + The date is in the allowed working days (workDays()).
     + The date is not in the past.
   * Based on sending\_reservations\_to\_representatives, calls either workHoursUsingAreas() or workHours().
4. **Remove Duplicates & Sort Appointments**
   * Flattens the appointments array.
   * Sorts them based on time in ascending order.
5. **Filter by Time Range (from & to)** only to companies
   * If from and to times are provided, filters the appointments within that range.
6. **Return JSON Response**
   * Returns the formatted and sorted appointment slots as JSON.

### 1.3.2 removeDuplicateAppointments() Method Overview

This method removes duplicate appointment time slots from the given array of appointments, ensuring that each time slot appears only once.

#### ****🔹 Explanation of Logic****

1. **Initialize Two Arrays**
   * $uniqueAppointments: Stores unique appointments.
   * $seenKeys: Keeps track of already processed time slots.
2. **Iterate Through Appointments**
   * Extracts the first key (time) from each appointment.
   * If the time slot is not already in $seenKeys, it adds the appointment to $uniqueAppointments.
   * Adds the time slot to $seenKeys to prevent future duplicates.

### 1.3.3 formatDate() Method Overview

This method **formats a given date** into a **lowercase** three-letter weekday abbreviation (mon, tue, wed, etc.).\*\*

#### ****🔹 Explanation of Logic****

1. **Parse the Date**
   * **Uses Carbon::createFromFormat('Y M d', $date) to** convert the date string **into a Carbon instance.**
   * **Expected input format:** 2024 Sep 23 (Year Month Day)
2. **Format the Date**
   * **Uses .format('D') to get the** three-letter weekday abbreviation **(Mon, Tue, Wed, etc.).**
3. **Convert to Lowercase**
   * Uses strtolower() to return the **abbreviation in lowercase**.

### 1.3.4 workDays() Method Overview

This method **retrieves the working days** from the Application model and returns them as an array.

#### ****🔹 Explanation of Logic****

1. **Retrieve the First Application Record**
   * Fetches the first record from the Application table.
   * Assumes that **working days are stored as a comma-separated string** (e.g., "mon, tue, wed, thu, fri").
2. **Extract Working Days**
   * If working\_days exists and is **not null**, it **splits the string** into an array using explode(', ', ...).

### 1.3.5 workHoursUsingAreas() Method Overview

This method determines available working hours based on the provided location, date, and shifts. It checks for restricted working days, retrieves shifts, and generates available time slots accordingly.

#### **Explanation of Logic**

#### 1-****Determine the Location Area****

* If locationId or location\_id is provided and not equal to 1, find the location in the Locations table and determine its area using checkLocation().
* If latitude (lat) and longitude (lng) are provided, determine the area using checkLocation().
* If no location details are given, use a default location (lat: 26.408238186969534, lng: 43.97441312670708).

#### 2-****Check if the Date is a Non-Working Day****

* Retrieve the first application from the Application table.
* Get the not\_working\_days JSON from the application settings and decode it into an array.
* Convert the given date into the format Y-m-d.
* If the reservation date exists in the non-working days list, return an empty list of hours.

#### 3-****Retrieve Work Shifts for the Given Date and Area****

* Convert the reservation date to a weekday format (e.g., Monday, Tuesday).
* Retrieve all shifts where:
  + **day** matches the reservation date’s weekday.
  + **area\_id** matches the determined area.
  + Results are sorted by the **from** time.

#### 4-****Determine the Number of Available Representatives****

* Call **getRepresentativesCount()** with the center application, reservation date, and requested time to get the number of available representatives.

#### 5-****Check App Acceptance Rules****

* If the area’s city ID is 5, use the general accepting setting.
* Otherwise, use **accepting\_of\_riyadh** from the application settings.

#### 6-****Generate Available Time Slots****

* If the application is configured to use work shifts (**use\_work\_shift** == 1):
  + Loop through the shifts and generate time slots using **generateTimeSlots().**
* If no work shifts are defined but the application has from and to times set:
  + Generate time slots using **generateTimeSlots()** with the application’s defined working hours.

#### 7-****Return the Available Hours****

* The method returns an array of available time slots for the specified date, location, and shift settings.

### 1.3.6 checkAppointmentUsingArea() Method Overview

This method checks if an appointment can be scheduled based on the given date, time, and availability of representatives. It ensures that the selected time slot is within working shifts, that there are enough representatives available, and that the appointment is not too close to the current time.

#### **Explanation of Logic**

#### 1-****Parse the Given Time****

* The time is split into hours and minutes using explode(":", $time).
* If the hour value is greater than 12, it is converted to a 12-hour format.

#### 2-****Retrieve Application Settings****

* The method retrieves the first application instance using Application::first().
* If the application uses work shifts (use\_work\_shift == 1), shift-based scheduling logic is applied.

#### 3-****Identify the Current Work Shift****

* The current weekday (Monday, Tuesday, etc.) is determined using Carbon::now()->format('l').
* The method fetches the current shift where:
  + The shift’s from time is before or equal to the given time.
  + The shift’s to time is after or equal to the given time.

#### 4-****Check Available Representatives****

* If representatives are available ($representatives\_count > 0):
  + The method retrieves representatives assigned to the current shift.
  + If the shift has assigned clients, their IDs are compared with the provided representatives list to get common IDs.

#### 5-****Count Existing Appointments****

* The method counts the number of existing reservations for the given date and time.
* It checks two conditions:
  + If work shifts are enabled, it checks Reservation records where from matches the given time.
  + If work shifts are disabled, it checks reservations where to matches the given time.

#### 6-****Verify Appointment Eligibility****

* The method calculates the difference between the current time and the requested appointment time using diffInMinutes().
* The appointment is **not allowed** if:
  + The number of available representatives is less than or equal to the number of existing appointments.
  + The requested time has already passed.
  + The requested time is within 15 minutes of the current time.

#### 7-****Validate Work Shift Rules****

* If work shifts are enabled, the method loops through all shifts for the day.
* It checks if the requested time falls within any shift’s start and end times.
* If the time falls within a shift, **the method returns** true **(appointment allowed)**.
* Otherwise, it returns false.

### 1.3.7 workHours() Method

#### ****Overview****

This method generates available working hours for a given date based on shifts, non-working days, and the center's working hours.

### 🔹 *****Explanation of Logic*****

#### ****1. Initialize Variables****

* Retrieves the application instance (Application::first()).
* Converts the input date into Y-m-d format.

#### ****2. Check for Non-Working Days****

* Decodes not\_working\_days from JSON.
* If the requested date is in the non-working days list, returns an empty array.

#### ****3. Retrieve Shifts****

* Gets all shifts for the specified day (Shift::where('day', ...)).
* Orders shifts by start time.

#### ****4. Adjust Special Date Conditions****

* If the reservation date is "2025-03-11", overrides the application’s accepting time to "90" minutes.

#### ****5. Determine Representative Availability****

* Calls getRepresentativesCount() to get the number of available representatives for the date.

#### ****6. Generate Time Slots****

* If the application uses work shifts, iterates through shifts and generates time slots for each.
* If work shifts are not used, generates time slots based on the center's general working hours.

#### ****7. Return Available Time Slots****

* Returns an array of available working hours.

### 1.3.8 getRepresentativesCount() Method

#### ****Overview****

This method determines the number of available representatives for a given date and time, taking into account work shifts and representative availability.

### 🔹 *****Explanation of Logic*****

#### ****1. Parse Date and Time****

* Converts $date to a Carbon instance and extracts the weekday (l format).
* Initializes $currentTime to the current system time (Carbon::now()->toTimeString()).
* If $time is provided:
  + Removes "AM" and "PM" from the string.
  + Converts the modified string to a Carbon time format.

#### ****2. Determine Current Work Shift****

* Retrieves the shift where the given time falls between from and to.
* If multiple queries exist, only the first valid shift is selected.

#### ****3. Count Available Representatives Based on Work Shifts****

* If the center is using work shifts (use\_work\_shift == 1) and a shift is found:
  + Retrieves the representative IDs (clients) assigned to that shift.
  + Filters representatives based on:
    - type = 1 (only active representatives).
    - status = 1 (representative is active).
    - Non-null lat and lng (representative must have location data).
  + Returns the count of available representatives.

#### ****4. Count Available Representatives (Without Work Shifts)****

* If no shift-based filtering is applied, retrieves all representatives who:
  + Have type = 1.
  + Are active (status = 1).
  + Have non-null lat and lng.
* Returns the count of all available representatives.

### 🔹 ****Return Value****

* Returns an integer representing the count of available representatives.

### 1.3.9 generateTimeSlots() Method

#### ****Overview****

This method generates available time slots between a given start (from) and end (to) time for a specific reservation date. It checks the availability of representatives before adding each time slot.

### 🔹 *****Explanation of Logic*****

#### ****1. Initialize Variables****

* $hours → Stores the list of available time slots.
* $tStart → Converts the **from** time into a Unix timestamp. If $time is provided, it uses that instead.
* $tEnd → Converts the **to** time into a Unix timestamp.
* $tNow → Keeps track of the current time while iterating through the available slots.

#### ****2. Loop Through Available Time Slots****

* A while loop iterates from $tStart to $tEnd.
* Each iteration:
  + Converts the current timestamp ($tNow) into a formatted time (H:i A).
  + Calls checkAppointment() to verify if the time slot is available based on representative availability.
  + If the time slot is valid, it is added to the $hours array in a formatted way (%I:%M %p).
  + Moves to the next time slot using the $acceptingInterval (default: 60 minutes).

#### ****3. Check for Availability****

* The checkAppointment() method is used to ensure the slot is available before adding it.

### 1.3.10 checkAppointment() Method Breakdown

#### ****🔹 Overview****

This function checks whether a given time slot is available for a reservation. It considers work shifts (if enabled), representative availability, location constraints, and scheduling rules.

## ****🔹 Step-by-Step Explanation****

### ****1-Extract Date and Time Information****

* Converts $reservation\_date into a day name (Monday, Tuesday, etc.).
* Normalizes time format by replacing "AM" or "PM" with ":00" for parsing.
* Converts $reservation\_date and the updated $timeNew into a **Carbon** object ($targetTime).
* Converts $reservation\_date alone into $targetDate.

### ****2-Work Shift Handling (If Enabled)****

* Checks if work shifts are enabled (use\_work\_shift == 1).
* Retrieves the current shift for the given day where from <= $targetTime and to >= $targetTime.
* Gets the list of representatives assigned to this shift.
* Counts the number of **existing reservations** (appointments\_count) for the given date, time, and representatives.

### ****3-General Appointment Check (If No Work Shifts)****

* If work shifts **are not enabled**, it simply counts the number of **existing reservations** for the given date and time.

### ****4-Handling Time Constraints****

* Calculates the difference between **current time** ($currentTime) and the **target appointment time** ($targetTime).
* **Invalid cases** (returns false if any condition is met):
  + **Not enough representatives** ($representatives\_count <= $appointments\_count).
  + **The time is in the past** ($currentTime->gt($targetTime) and the reservation is for **today**).
  + **The reservation is too soon** (less than 15 minutes away).

### ****5-Determine Work Area (If Work Shift is Enabled)****

* Checks the **location ID** in $request and retrieves area information using checkLocation().
* Fetches shifts based on:
  + The **day** of the reservation ($targetDate->format('l')).
  + The **area ID** (if applicable).
* Determines **accepting interval** ($appAccepting), which changes based on the city.

### ****6-Generate Available Slots for Shifts****

* Loops through **each shift** and splits it into 90-minute slots.
* Saves slots as an array indexed by shift\_id.

### ****7-Check If the Target Time Falls in a Shift****

* Compares $targetTime with available slots.
* If a **match is found**, retrieves:
  + The representatives\_count in that shift.
  + The number of **existing reservations** at that time.
* If **reservations < available representatives**, the slot is valid (return true).

### ****🔹 Return Value****

* **Returns true** if the time slot is available.
* **Returns false** if the time slot is **already full, outside shifts, or restricted by constraints**
  1. **Brand Controller**

## 1.4.1 allBrands() Method

#### ****Overview****

This method retrieves all available brands from the database and returns them in ascending order based on their ID.

### ****🔹 Explanation of Logic****

#### ****1. Fetch Brands****

* Calls Brand::orderBy('id', 'asc')->get() to retrieve all brands, sorted by their id in ascending order.
* Wraps the retrieved data in the brandsResource collection to format the response.

#### ****2. Return Response****

* If successful, returns a structured JSON response with a list of brands.
* If an error occurs, catches exceptions and returns an error response.
  1. Cancelling Reasons Controller

## 1.5.1 all() Method

#### ****Overview****

This method retrieves all cancellation reasons from the database, ordered by the most recent ones.

### ****🔹 Explanation of Logic****

#### ****1. Fetch Cancellation Reasons****

* Calls CancelReason::latest()->get() to retrieve all cancellation reasons, ordered by the most recent entries.

#### ****2. Return Response****

* If successful, returns a structured JSON response containing all cancellation reasons.
* If an error occurs, catches the exception and returns an error response.
  1. Car Controller

## 1.6.1 getCars() Method

#### **Overview**

This method retrieves all cars associated with the authenticated user that have a related car size.

### ****🔹 Explanation of Logic****

#### ****1. Retrieve Cars****

* Uses Car::whereHas('carSize') to filter cars that have a valid car size relationship.
* Filters cars that belong to the authenticated user (user\_id = request()->user()->id).
* Orders the cars by the latest entry (latest()->get()).
* Wraps the data using CarResource::collection() for consistent API formatting.

#### ****2. Return Response****

* If successful, returns a structured JSON response containing the user's cars.
* If an error occurs, catches the exception and returns an error response.

## 1.6.2 addCarNewWay() Method

#### **Overview**

This method allows authenticated users to add a new car to their profile. It validates input data, retrieves the brand image (if available), and stores the car details in the database.

### ****🔹 Explanation of Logic****

#### ****1. Log Incoming Request****

* Logs the entire request data using INFO($request->all()) for debugging.

#### ****2. Validate Input Data****

* Defines validation rules using Laravel's Validator::make() method:
* brand\_id and car\_type\_id are **required** and must exist in their respective tables (brands, car\_types), excluding soft-deleted records.
* color\_id is required.
* Other fields are optional (name, car\_plate\_number, car\_size\_id).
* If validation fails, it returns a **403 Forbidden** response with the first validation error message.

#### ****3. Retrieve Brand Image (If Applicable)****

* If brand\_id is provided, retrieves the brand image from the brands table and assigns it to the car.

#### ****4. Store the Car in the Database****

* Creates a new record in the cars table with the provided details.

#### ****5. Return Response****

* If successful, returns a **200 OK** response with a success message.
* If an error occurs, catches the exception and returns a **403 Forbidden** response.

## 1.6.3 updateCarNewWay() Method

#### **Overview**

This method allows authenticated users to update an existing car's details, including its name, plate number, size, brand, type, and color. If the brand is changed, the car’s image is updated to the new brand’s image.

### ****🔹 Explanation of Logic****

#### ****1. Validate Input Data****

* Defines validation rules:
  + car\_id is **required** and must exist in the cars table.
  + brand\_id and car\_type\_id are **required** and must exist in their respective tables (brands, car\_types), excluding soft-deleted records.
  + color\_id is required.
  + Other fields are optional (name, car\_plate\_number, car\_size\_id).
* If validation fails, it returns a **403 Forbidden** response with the first validation error message.

#### ****2. Retrieve the Car Record****

* Finds the car using Car::find($request->car\_id).
* Stores the current image name.

#### ****3. Update Car Image (If Brand Changes)****

* If a new brand\_id is provided, retrieves the **brand's image** from the brands table and assigns it to the car.

#### ****4. Update the Car in the Database****

* Updates the car’s details with the new data.

#### ****5. Return Response****

* If successful, returns a **200 OK** response with a success message.
* If an error occurs, catches the exception and returns a **403 Forbidden** response.

### 1.6.4 colors() Method Documentation

#### **Overview**

This method retrieves all available car colors from the colors table, including their ID, name, and hex code.

### ****🔹 Explanation of Logic****

#### ****1. Fetch Colors from the Database****

* Uses DB::table('colors')->get(['id', 'color\_name', 'hex\_code']) to retrieve the color ID, name, and hex code.

#### ****2. Return JSON Response****

* If successful, returns a **200 OK** response with the list of colors.
* If an error occurs, it catches the exception and returns a **403 Forbidden** response.

### 1.6.5 deleteCar() Method

#### **Overview**

This method deletes a specific car based on the provided car\_id.

### ****🔹 Explanation of Logic****

#### ****1. Validate Request****

* Ensures that car\_id is provided and exists in the cars table.

#### ****2. Find and Delete Car****

* Retrieves the car using Car::find($request->car\_id).
* Calls $car->delete() to remove the record.

#### ****3. Return JSON Response****

* Returns **200 OK** on successful deletion.
* If validation fails, returns **403 Forbidden** with the validation error.
* If an exception occurs, returns **403 Forbidden** with the error message.
  1. Car Size Controller

### ****1.7.1**** index() Method

#### **Overview**

This method retrieves a list of all available car sizes and formats them based on the application's locale.

### ****🔹 Explanation of Logic****

#### ****1. Retrieve Car Sizes****

* Fetches all records from the CarSize model.
* Orders them in descending (DESC) order by id.

#### ****2. Format Car Sizes****

* Checks if $car\_sizes is not null.
* Calls the formatCarSizes() method to format the car sizes based on the application's language.

#### ****3. Return JSON Response****

* Returns a **200 OK** response with the formatted list of car sizes.

### 1.7.2 show($id) Method

#### **Overview**

This method retrieves a specific car size by its id and returns it in a formatted response based on the application's locale.

### ****🔹 Explanation of Logic****

#### ****1. Retrieve Car Size****

* Finds the CarSize record by its id using where('id', $id)->first().

#### ****2. Format the Car Size****

* Checks if the retrieved car\_size is **not null**.
* Calls the formatCarSize() method to format the car size details based on the app's locale.

#### ****3. Return JSON Response****

* Returns a **200 OK** response containing the formatted car size details.
* If the car size does not exist, an **empty array** is returned instead of an error.

### 1.7.3 formatCarSizes($car\_sizes, $lang) Method

#### **Overview**

This private method takes a collection of CarSize objects and formats each one using the formatCarSize() function, returning an array of formatted car sizes.

### ****🔹 Explanation of Logic****

#### ****1. Initialize an Empty Array****

* $car\_sizes\_array = []: This array will store the formatted car sizes.

#### ****2. Loop Through Each**** CarSize ****Object****

* Iterates through the $car\_sizes collection.
* Calls the formatCarSize($car\_size, $lang) function to apply formatting.
* Pushes the formatted result into $car\_sizes\_array.

#### ****3. Return Formatted Car Sizes****

* Returns an array containing all formatted car sizes.

### 1.7.4 formatCarSize($car\_size, $lang) Method

#### **Overview**

This private method takes a CarSize object and formats it into an array with translated fields, returning structured data for API responses.

### ****🔹 Explanation of Logic****

#### ****1. Extracting Data from the Model****

* Retrieves the id from the CarSize model.
* Uses $car\_size->getTranslation('title', $lang) to fetch the localized title.
* Uses $car\_size->getTranslation('description', $lang) to fetch the localized description.
* Retrieves the price from the CarSize model.

#### ****2. Returns a Formatted Array****

* The returned array contains structured data with localization support.
  1. Car Type Controller

### 1.8.1 car\_types\_Of\_some\_brand($brand\_id) Method

#### **Overview**

This method retrieves all car types associated with a given brand\_id, sorts them by id in ascending order, and returns the results in a structured JSON format using a CarTypesResource collection.

### ****🔹 Breakdown of Functionality****

#### ****1-Fetching Car Types by Brand****

* Retrieves car types where brand\_id matches the given $brand\_id.
* Uses orderBy('id', 'asc') to sort the results in ascending order.
* Uses get() to fetch the data from the database.

#### ****2-Formatting Response Using API Resource****

* Wraps the retrieved CarType models in a CarTypesResource collection for structured API output.
* Calls $this->returnData() to return data with a success message.

#### ****3-Handling Errors****

* Uses try...catch to catch exceptions and return a formatted error message using $this->returnError().
* Includes \_\_('api.error\_happened') . $error->getMessage() for better debugging.

### 1.9 Client Controller

### 1.9.1 login\_using\_phone(Request $request) Method

#### **Overview**

This method allows users to log in using their phone number. If the phone number exists in the database, an activation code is sent. If the number is new, a new user is created, and an activation code is sent.

### ****🔹 Breakdown of Functionality****

#### ****1-Validate Input****

* Ensures that the phone field is present in the request.
* Optionally, device\_token validation is commented out.

#### ****2-Check if the Phone Number Exists****

* If the phone exists:
  + Generates a **4-digit activation code**.
  + If the phone is a test number (0500000000 or 0511111111), sets the activation code to 1234.
  + Saves the activation code to the client.
  + Sends an SMS with the activation code.
  + If complete\_data == 1 (it means that the user have an activated account and he filled all the data like username and email), prompts the user to enter the activation code for verification.
  + Otherwise, notifies the user that their account is not activated.

#### ****3-If the Phone Number Does Not Exist****

* Creates a new Client with:
  + phone
  + status = 0 (for unactive)
  + activation\_code = random 4-digit code
  + complete\_data = 0 (the account data is not completed yet)
* Sends an SMS with the activation code.
* Returns a response prompting the user to enter the OTP.

### 1.9.2 updateDeviceToken($client, $device\_token)Method

#### **Overview**

This method updates the device\_token of a client, ensuring the latest device information is stored for push notifications or authentication purposes.

### ****🔹 Breakdown of Functionality****

#### ****1-Update the Device Token****

* Assigns the new device\_token value to the client.
* Saves the updated client record in the database.

#### ****2-Ensure Data Persistence****

* Calls the save() method on the Client model to commit the changes.
* Returns true to indicate the update was successful.

### 1.9.3 register\_using\_phone(Request $request) Method

#### **Overview**

This method registers a user using their phone number. If the phone number exists in the database with status = 1, the user's details are updated, and an authentication token is generated. Otherwise, an error message is returned.

### ****🔹 Breakdown of Functionality****

#### ****1-Check If the Client Exists****

* Looks for a Client record where phone = $request->phone and status = 1.
* If a matching client is found, proceeds to update the user’s data.
* If no matching client is found, returns a 404 response indicating that the client does not exist.

#### ****2-Update Client Information**** (If Exists)

* Updates the username, gender, complete\_data, heard\_about\_us\_from, and comming\_by fields.
* Marks complete\_data = 1, indicating that the user's profile is fully updated.

#### ****3-Generate Authentication Token****

* Creates an API token for the user using Laravel Passport (createToken('API')->accessToken).
* Returns the updated user data along with the token.

#### ****4-Return Response****

* **If the update is successful:**  
  ✅ Returns a 200 status with a success message, formatted user details, and the generated token.
* If the client does not exist:  
  ❌ Returns a 404 error with a message stating that no client with the given phone number exists.

### 1.9.4 resendCode(Request $request) Method

#### **Overview**

This method allows users to request a new activation code (OTP) if they have not received or lost their previous one. If the phone number exists in the database, a new 4-digit activation code is generated, saved, and sent via SMS.

### ****🔹 Breakdown of Functionality****

#### ****1-Validate Input****

* Ensures that the phone field is provided in the request.
* If validation fails, returns a 403 response with the validation error messages.

#### ****2-Check If the Phone Number Exists****

* Searches for a Client record with the given phone number.
* If the client does not exist, returns a 403 response with an error message.

#### ****3-Generate & Update Activation Code****

* Generates a **random 4-digit OTP** (random\_int(1000, 9999)).
* Updates the activation\_code field for the client.

#### ****4-Send SMS with the New Activation Code****

* Constructs an SMS message containing the OTP.
* Calls sendSms($client->phone, $client\_activation\_code) to send the activation code via SMS.

#### ****5-Return Response****

* **If successful:** ✅ Returns a 200 response with a success message and the generated OTP.
* **If the client does not exist:** ❌ Returns a 403 response indicating that the user does not exist.

### 1.9.5 verifyCodeOfPhone(Request $request) Method

#### **Overview**

This method verifies the activation code entered by the user. If the code is correct and belongs to a registered phone number, the user's account is activated, their device token is saved, any pending gift balance is added to their account, and a wallet transaction is recorded. Finally, an API token is generated for authentication.

### ****🔹 Breakdown of Functionality****

#### ****1-Validate Input****

* Ensures that:
  + code (activation code) is provided.
  + device\_token is provided.
* If validation fails, returns a 403 response with the validation errors.

#### ****2-Find the Client with the Given Activation Code****

* Searches for a Client with:
  + The provided activation\_code.
  + The provided phone number.
* If no client is found, returns a 403 response with an error message.

#### ****3-Activate the User's Account****

* If a matching client exists:
  + Updates status = 1 (activates the account).
  + Saves the device\_token for future notifications.

#### ****4-Handle Pending Gift Balance****

* Retrieves and sums the total price of all **unclaimed** (status = 0, payment\_status = 1) **gifts** sent to this phone number.
* Adds the retrieved balance to the user's account.
* Updates the status of these gifts to status = 1 and associates them with the client.

#### ****5-Record Wallet Transaction if Gifts Were Received****

* If the client received a gift:
  + Creates a **wallet operation** entry with:
    - **Description:** "تم استلام هدية بقيمة {balance} ريال" (Received a gift of {balance} Riyals).
    - **Date and Time:** Current timestamp.
    - **Amount:** Gift amount.
    - **User ID:** Client's ID.
    - **Type:** 1 (Income).
    - **Status:** 1.

#### ****6-Generate Authentication Token****

* Creates an API access token for the user.

#### ****7-Return a Response Based on**** complete\_data ****Status****

* If complete\_data == 1, the user is fully registered, and a success message with user details is returned.
* Otherwise, the response prompts the user to complete their profile.

### 1.9.6 profile(Request $request) Method

#### **Overview**

This method retrieves the authenticated user's profile details, formats the user data, and returns it in a JSON response along with a hardcoded material value.

### ****🔹 Breakdown of Functionality****

#### ****1-Retrieve the Authenticated User****

* $request->user() gets the currently authenticated user based on the API token.

#### ****2-Format the User Data****

* The formatUser($request->user()) function processes and structures the user data.

#### ****3-Return the Formatted User Data****

* Returns a JSON response with:
  + user → Formatted user data.
  + material → Hardcoded value (0).

### 1.9.7 updateProfile(UpdateClientRequest $request) Method

#### **Overview**

This method allows authenticated clients to update their profile details, including their username, phone number, latitude, and longitude.

### ****🔹 Breakdown of Functionality****

#### ****1-Retrieve the Authenticated Client****

* Uses $request->user() to get the currently authenticated client.

#### ****2-Fetch the Client from Database****

* Retrieves the client record from the database using the authenticated client's ID.

#### ****3-Update Client Details****

* Uses $updated\_client->fill($request->only(...)) to update the client's **username, phone, latitude (**lat**), and longitude (**lng**)**.
* Saves the updated values with $updated\_client->update().

#### ****4-Return a JSON Response****

* Returns a success message along with the formatted updated client details.

### 1.9.8 updateProfileImage(Request $request)Method

#### **Overview**

This method allows an authenticated client to update their profile image. The image must be in **JPEG, JPG, or PNG** format.

### ****🔹 Breakdown of Functionality****

#### ****1-Retrieve the Authenticated Client****

* Uses $request->user() to get the currently authenticated client.

#### ****2-Validate the Request****

* Ensures that the image field is present and has a valid MIME type (jpeg, jpg, png).
* If validation fails, returns an error response (403 Forbidden).

#### ****3-Fetch the Client Record****

* Retrieves the client's record from the database using their ID.

#### ****4-Handle Image Upload****

* If an image is uploaded, it processes the file using handleFile($request['image']) and assigns the new path to $updated\_client->image.

#### ****5-Update the Client's Profile****

* Saves the updated image path in the database.

#### ****6-Return a JSON Response****

* Returns a success message along with the formatted updated client details.

### 1.9.9 updateLocation(Request $request) Method

#### **Overview**

This method allows an authenticated client to update their **latitude (**lat**)** and **longitude (**lng**)** (i.e., their location).

### ****🔹 Breakdown of Functionality****

#### ****1-Retrieve the Authenticated Client****

* Fetches the logged-in user's record from the Client table using $request->user()->id.

#### ****2-Update Location Data****

* Updates the lat and lng fields with the new values from the request.

#### ****3-Save Changes****

* Saves the updated location in the database.

#### ****4-Return a JSON Response****

* Sends a success message along with the updated client details.

### 1.9.11 formatUser($user) Method

#### **Overview**

This method **formats** the Client user data before returning it in API responses.

### ****🔹 Breakdown of Functionality****

#### ****1-Determine**** gender\_text

* Converts gender numeric values into a localized string (male, female, or other).

#### ****2-Construct an Array with User Information****

* Includes essential fields like id, username, phone, email, location, etc.
* Formats the **profile image** URL properly.

#### ****3-Return the Formatted User Data****

### 1.9.12 balance(Request $request) Method

#### **Overview**

This method retrieves the balance of the authenticated client and returns it as a JSON response.

**🔹 Breakdown of Functionality**

1-**Get the Authenticated User (Client)**

* Uses $request->user() to retrieve the logged-in client.

2-**Retrieve the User's Balance**

* Fetches the balance field from the Client model.

3-**Return a JSON Response**

* Converts the balance to a float to ensure proper formatting.
* Returns a 200 HTTP status (OK).

### 1.9.13 addBalanceNewWay(Request $request) Method

#### **Overview**

This method adds balance to a user's wallet either via direct payment or through a bank transfer.

**🔹 Breakdown of Functionality**

1-**Validate Input**

* bank\_id: Optional, must exist in bank\_accounts table.
* image: Optional, must be a jpeg, jpg, or png file.

2-**If Amount is Provided (**Direct Wallet Addition**)**

* Retrieves the authenticated user.
* Checks if wallet operations require verification (Application::first()->verify\_wallet\_operations).
* If verification is needed, logs the transaction in WalletOperation.
* Returns a success response.

3-**If No Amount is Provided (Bank Transfer)**

* Retrieves bank details using bank\_id.
* Creates a BankTransfer record with bank details.
  + - Saves the transfer receipt (image).
    - Returns a confirmation message

### 1.9.14 walletOperations(Request $request) Method

#### **Overview**

This method retrieves verified wallet operations for the authenticated user.

**🔹 Breakdown of Functionality**

1-**Check if Wallet Operations Need Verification**

* Retrieves the Application settings.
* Checks if verify\_wallet\_operations is enabled (1).

2-**If Verification is Enabled**

* Retrieves wallet operations for the logged-in user where status is 1 (verified transactions).
* Returns the operations sorted by the latest transaction.

3-**If Verification is Disabled**

* Returns an empty operations list.

### 1.9.15 logout(Request $request) Method

#### **Overview**

This method logs out the authenticated user by revoking their access token.

**🔹 Breakdown of Functionality**

1-**Revoke the User's Token**

* Calls $request->user()->token()->revoke();
* This invalidates the current authentication token, logging the user out.

2-**Return a Success Message**

* Returns a JSON response with a success message (trans('api.logout')).

### 1.9.16 logout(Request $request) Method

#### **Overview**

This method logs out the authenticated user by revoking their access token.

**🔹 Breakdown of Functionality**

1-**Revoke the User's Token**

* Calls $request->user()->token()->revoke();
* This invalidates the current authentication token, logging the user out.

2-**Return a Success Message**

* Returns a JSON response with a success message (trans('api.logout')).

### 1.9.17 generateRepresentativeRates($representative\_id) Method

#### **Overview**

This method fetches wallet operations for a given representative and returns a view displaying the rates.

**🔹 Breakdown of Functionality**

1-**Retrieve Wallet Operations**

* Searches for records in the WalletOperation model where:
  + user\_id matches the $representative\_id.
  + type\_notify is set to 1.
* Uses:
  + - * WalletOperation::where('user\_id', $representative\_id)->where('type\_notify', 1)->get();

2-**Return a View**

* Passes the retrieved rates data to dashboard.wallets.rates view.
* Uses:
  + - * return view('dashboard.wallets.rates', compact('rates'));

3-**Error Handling**

* Uses try...catch to handle potential errors.
* If an error occurs, it **returns a JSON response** with an error message and a 403 status code.

### 1.9.18 getRepresentativeRatesPage(Request $request) Method

#### **Overview**

This method generates a URL to fetch representative rates and returns a setting value that determines whether the rates should be displayed.

**🔹 Breakdown of Functionality**

1-**Retrieve Application Settings**

* Fetches the first record from the Application model.

2-**Generate the Rates Page URL**

* Creates a URL using the named route "representative.rates".
* Passes the authenticated user's ID to generate a dynamic URL.

3-**Return JSON Response**

* Returns an array with:
  + "rates" → The generated URL.
  + "display" → The value of verify\_representative\_rates from settings.

**2.0 Complaint Controller**

### 2.0.1 getComplaints(Request $request) Method

**🔹 Overview**

This method retrieves all complaints associated with the authenticated admin.

**🔹 Breakdown of Functionality**

1-**Validate User Type**

* Fetches the authenticated user.
* If the user is **not an supervisor (type != 3)**, returns a **403 Forbidden** response.

2-**Retrieve Complaints**

* Fetches complaints where the admin\_id matches the authenticated user's admin\_id.
* Uses **Eloquent Resources (ComplaintResource)** to format the response.

3-**Return Response**

* Returns the **list of complaints** in a JSON response with a **200 status**.

4-**Error Handling**

* Catches unexpected errors and returns a **403 Forbidden** response.

### 2.0.2 send(Request $request) Method

**🔹 Overview**

This method allows **representatives** to send complaints to their assigned admin.

**🔹 Breakdown of Functionality**

1-**Validate User Type**

* If the authenticated user is **not a representative (type != 1)**, returns a **403 Forbidden** response.

2-**Validate Request Data**

* Ensures the request contains a **valid message**.
* If validation fails, returns a **422 Unprocessable Entity** response.

3-**Create a New Complaint**

* Saves the **complaint** in the database.

4-**Send Notification**

* Dispatches a **delayed notification** to the admin.

5-**Return Response**

* Returns a **success message** indicating the complaint was sent successfully.

6-**Error Handling**

* Catches unexpected errors and returns a **403 Forbidden** response.

### 2.0.3 changeStatus(Request $request) Method

**🔹 Overview**

This method **updates the status of a complaint** to "solved".

**🔹 Breakdown of Functionality**

1-**Validate User Type**

* If the authenticated user is **not an admin (type != 3)**, returns a **403 Forbidden** response.

2-**Find the Complaint**

* Searches for a complaint using the **provided complaint\_id**.
* If not found, returns a **404 Not Found** response.

3-**Update Complaint Status**

* Sets the **status to 1 (solved)** and saves the changes.

4-**Return Success Response**

* Returns a **success message** after updating the status.

5-**Error Handling**

* Catches unexpected errors and returns a **422 Unprocessable Entity** response.

**2.1 Coupon Controller**

### 2.1.1 checkNew(Request $request) Method

**🔹 Overview**

This method validates a coupon code, ensuring it is active, has not expired, and meets usage restrictions before applying it.

**🔹 Breakdown of Functionality**

1-**Validate Request Parameters**

* Ensures the request contains the required fields: code, date, and time.
* If validation fails, returns a **403 Forbidden** response.

2-**Check if the Coupon Exists & Is Active**

* Retrieves the coupon using the provided code and ensures:
  + The coupon exists.
  + The coupon status is active (status = 1).
  + The coupon has remaining usage (times != used).

3-**Check if the User Has Already Used the Coupon**

* If the coupon is limited to one-time use per user (valid\_for\_one\_user == 1):
  + Checks if the user has already used the coupon.
  + Returns **403 Forbidden** if the coupon was used before or if times == 0 (fully consumed).

4-**Determine the Date for Coupon Validation**

* If type\_use\_of\_coupon == 0, (check the way we use from setting use date only or date and time) uses the **current date and time**.
* Otherwise, uses the provided date and time from the request.

5-**Validate Coupon Based on Location (if applicable)**

* If the coupon type (coupon\_type == 0), checks if the user is **within a valid location range (0.5 km)** from the company's location.
* If within range, calls check\_date\_of\_coupon\_company() to further validate the coupon.

6-**Return Response**

* If the coupon is valid, returns the formatted coupon details with a **200 OK** response.
* Otherwise, returns **403 Forbidden** with an error message.

### 2.1.2 checkCompany(Request $request) Method

🔹 **Overview**  
This method verifies a coupon's validity for a company based on the provided latitude and longitude.

🔹 **Breakdown of Functionality**

1-**Validate Request Parameters**

* Ensures that code, lat, and long are provided in the request.
* If validation fails, returns a 403 response with validation errors.

2-**Retrieve and Validate the Coupon**

* Finds the coupon by code and checks if it is active (status = 1).
* If the coupon doesn't exist or has been fully used, returns 403 with an error message.

3-**Check If the User Has Used the Coupon Before**

* If the coupon is restricted to a single use per user (valid\_for\_one\_user = 1), verifies whether the user has already used it.
* If the user has used the coupon, returns a 403 response.

4-**Check Coupon Eligibility Based on Location**

* Retrieves the company associated with the coupon.
* Checks if the user has a location entry matching the company’s coordinates. If not, creates one.
* Calculates the distance between the user’s provided lat and long and the company’s location.

5-**Determine Coupon Validity**

* If the distance between the user and the company is ≤ 0.5, checks whether the coupon is valid using check\_date\_of\_coupon\_company().
* If valid, returns a 200 response with the formatted coupon details.
* Otherwise, returns a 403 response indicating that the coupon is not valid.

🔹 **HTTP Responses**

* 200: Returns the formatted coupon if valid.
* 403: Returns an error message if validation fails, the coupon is expired, or the location conditions are not met.

### 2.1.3 formatCouponCompany($coupon, $location\_id) Method

🔹 **Overview**  
This private method formats the coupon data specifically for company-related coupons before returning it in the API response.

🔹 **Functionality**

1-**Extracts Coupon Details**

* Retrieves essential coupon attributes such as code, discount, use\_type, and applicable service.

2-**Formats Service Details**

* Calls $this->formatMainService($coupon->service) to structure service information.

3-**Includes Validity Period**

* Returns the time (time\_from, time\_to) and date (date\_from, date\_to) restrictions for the coupon.

4-**Includes Company and Location Details**

* Attaches the company\_id and location\_id where the coupon is valid.
* Formats company details using CompanyResource::make($coupon->company).

### 2.2.4 formatMainService($service) Method

🔹 **Overview**  
This private method formats the main service details associated with a coupon before returning it in the API response.

🔹 **Functionality**

1-**Extracts Service Details**

* Retrieves the id and cost of the service.

2-**Handles Multilingual Support**

* Uses $service->getTranslation('title', app()->getLocale()) to return the service title in the current application language.

### 2.1.5 formatCoupon($coupon) Method

🔹 **Overview**  
This private method formats the coupon details before returning it in the API response.

🔹 **Functionality**

1-**Extracts Coupon Details**

* Retrieves code, discount, and use\_type of the coupon.

2-**Formats Associated Services**

* Calls $this->formatService($coupon->car\_sizes) to structure the service details.

**Usage**

* This method is called inside checkNew() when validating a coupon.

### 2.1.6 formatService($services) Method

🔹 **Overview**  
This private method structures the service details for a given coupon before returning them in the API response.

🔹 **Functionality**

1-**Initializes an Empty Array**

* $services\_array = [] to store formatted service details.

2-**Loops Through Each Service**

* Extracts id and title (translated based on the current locale).
* Pushes formatted data into $services\_array.

🔹 **Return Format**

🔹 **Usage**

* This method is called inside formatCoupon($coupon), ensuring services are returned in a structured way.

### 2.1.7 check\_date\_of\_coupon\_company($coupon, $date, $request) Method

🔹 **Overview**  
This method checks whether a given coupon is valid based on its date, time, and activation period.

🔹 **Functionality**

1-**Checks if the Coupon is Unrestricted**

* If date\_from, date\_to, time\_from, and time\_to are null, the coupon is **always valid**.

2-**Validates the Date Range**

* Converts date\_from and date\_to into Carbon instances.
* Ensures that the current date falls within the defined range.

3-**Validates the Time Range (if applicable)**

* Checks if time\_from and time\_to are set.
* Ensures that the current time is within this period.

4-**Handles Coupons with Activation Periods**

* If period\_type != 1, the method checks active\_from and active\_to.
* If both values are null, the coupon is **always valid**.
* Otherwise, the coupon is valid if the current date falls within the activation period.

🔹 **Return Values**

* true → The coupon is valid.
* false → The coupon is **expired** or **not yet active**.

### 2.1.8 check\_date\_of\_coupon($coupon, $date) Method

🔹 **Overview**  
This method checks whether a given coupon is valid based on its activation period (active\_from and active\_to).

🔹 **Functionality**

1-**Checks if the Coupon is Unrestricted**

* If both active\_from and active\_to are null, the coupon is **always valid**.

2-**Validates the Activation Period**

* If active\_from **exists** and is **less than or equal to** the given $date, and active\_to **exists** and is **greater than or equal to** the $date, the coupon is **valid**.
* If active\_from is null, but active\_to **exists** and is **greater than or equal to** the $date, the coupon is **valid**.
* If active\_from **exists** and is **less than or equal to** the $date, but active\_to is null, the coupon is **valid**.

3-**Default Case: Coupon is Expired or Not Yet Active**

* If none of the above conditions match, the coupon is **invalid**.

🔹 **Return Values**

* true → The coupon is **valid**.
* false → The coupon is **expired** or **not yet active**.

### 2.1.9 myCoupons(Request $request) Method

🔹 **Overview**  
This method retrieves all coupons associated with the authenticated user and formats them for response. It also calculates the number of remaining days before expiration and determines if the coupon has been used.

**Functionality**

1-**Fetches Coupons for the Authenticated User**

* Filters coupons by user\_id matching the current authenticated user.
* Orders the results by the latest (latest()).
* Retrieves all matching coupons (get()).

2-**Formats Each Coupon**

* Calculates **remaining days** (reminder\_days) until the coupon expires.
  + Uses Carbon::now()->diffInDays(Carbon::parse($coupon->active\_to), false).
  + If the expiration date has passed, it sets reminder\_days to 0.
* Retrieves translations for **title** and **description** in Arabic.
* Determines if the coupon is **used** (used == 1) or **expired** (reminderDays <= 0).

3-**Returns the Coupons as a JSON Response**

**2.2 Location Controller**

**2.2.1 getLocations() Method**

### ****Overview****

This method retrieves all locations associated with the authenticated user and formats them for response.

### ****🔹 Functionality****

#### ****1-Fetches Locations for the Authenticated User****

* Filters locations where user\_id matches the currently authenticated user.
* Ensures only locations where outside = 0 are retrieved.
* Orders results by the latest (latest()).
* Retrieves all matching locations (get()).

#### ****2-Formats Each Location****

* Wraps each location in the LocationResource for structured formatting.
* Provides details such as name, coordinates, and address.

#### ****3-Returns the Locations as a JSON Response****

* If successful, returns a 200 OK response with the list of locations.
* If an error occurs, returns a 403 Forbidden response with an error message.

2.2.2 **addAddress() Method**

## ****Overview****

This method allows an authenticated user to add a new address, validating the input, checking if the location exists within the app's defined areas, and handling cases where the location is outside the supported areas.

## ****🔹 Functionality****

### ****1-Validates the Request Data****

* Ensures type, name, lat, and long are required.
* name should be a string with a maximum of 255 characters.
* lat and long should be numeric and greater than or equal to 0.
* address is optional but should be a string with a maximum of 255 characters.

### ****2-Checks If the Location Exists in the App's Defined Areas****

* Calls checkLocationExist($lat, $long) to determine if the given latitude and longitude are within supported areas.
* Calls checkLocation($lat, $long) to retrieve the area information.

### ****3-Handles Locations Outside the App’s Defined Areas****

* If the location does not exist within the app's supported areas (checkLocationExist returns false), it is added as an "outside" location (outside = 1).
* Returns an error response (404) with the message "العنوان غير موجود ضمن مناطق التطبيق" (Address is not within the app’s supported areas).

### ****4-Adds the Location If It Exists in the Defined Areas****

* Saves the location with the area\_id retrieved from checkLocation.
* Returns a success message (200 OK) with the translated "api.addLocation" response.

### ****5-Handles Errors Gracefully****

* Catches any unexpected errors and returns a 403 response with the error message.

2.2.3 **updateLocation() Method**

## ****Overview****

This method allows an authenticated user to update an existing location, ensuring data validation, checking if the location is within the app’s defined areas, and handling locations outside supported areas.

## ****🔹 Functionality****

### ****1-Validates the Request Data****

* Ensures the required fields: type, name, lat, long, address, and location\_id.
* location\_id must exist in the locations table.

### ****2-Checks If the Location Exists in the App's Defined Areas****

* Calls checkLocationExist($lat, $long) to determine if the given latitude and longitude are within supported areas.
* Retrieves the location by location\_id from the database.
* Calls checkLocation($lat, $long) to fetch the area information.

### ****3-Handles Locations Outside the App’s Defined Areas****

* If checkLocationExist returns false, it updates the location by setting outside = 1.
* Returns an error response (404) with the message "العنوان غير موجود ضمن مناطق التطبيق" (Address is not within the app’s supported areas).

### ****4-Updates the Location If It Exists in the Defined Areas****

* Updates the location with the new details, including the area\_id retrieved from checkLocation.
* Returns a success message (200 OK) with the translated "api.updateLocation" response.

### ****5-Handles Errors Gracefully****

* Catches any unexpected errors and returns a 403 response with the error message.

2.2.4 **deleteLocation() Method**

## ****Overview****

The deleteLocation method allows an authenticated user to delete a saved location. However, it ensures that the location is not associated with any ongoing reservations before proceeding with the deletion.

## ****🔹 Functionality****

### ****1-Validates the Request Data****

* Ensures the location\_id is provided and exists in the locations table.

### ****2-Checks If the Location Has Active Reservations****

* Retrieves the location from the database using the given location\_id.
* Counts reservations that are linked to the location and are in specific statuses (2, 4, 5), which indicate ongoing or uncompleted reservations.

### ****3-Restricts Deletion if the Location Has Active Reservations****

* If the location has any active reservations, it returns an error (403) with the message "عفوا لا يمكنك حذف عنوان لطلب قيد التنفيذ" (Sorry, you cannot delete an address associated with an ongoing request).

### ****4-Deletes the Location If No Active Reservations Exist****

* If the location is not associated with any active reservations, it is deleted from the database.
* Returns a success message (200 OK) with the translated "api.deleteLocation" response.

### ****5-Handles Errors Gracefully****

* Catches any unexpected errors and returns a 403 response with the error message.

2.2.5 **addressTypes() Method**

## ****Overview****

The addressTypes method returns a predefined list of address types available for selection by users. These address types are translated dynamically based on the application’s localization settings.

## ****🔹 Functionality****

### ****1-Defines Address Types****

* The method defines an array of address types:
  + 0 → **Home** (\_\_('api.home'))
  + 1 → **Work** (\_\_('api.work'))
  + 2 → **Other** (\_\_('api.other'))

### ****2-Returns the Address Types in JSON Format****

* The response contains an array of address types formatted as:

{

"addressTypes": [

{ "id": 0, "name": "المنزل" },

{ "id": 1, "name": "العمل" },

{ "id": 2, "name": "أخرى" }

]

}

* The names are localized based on the application's language settings.

### ****3-Handles Errors Gracefully****

* If an error occurs, the method catches it and returns a 403 response with the error message.

2.2.6 **checkLocationExist() Method**

## ****Overview****

The checkLocationExist() method determines whether a given latitude and longitude fall within the boundaries of predefined areas. It does this by checking if the coordinates are inside any polygon formed by the area's points.

## ****🔹 Functionality****

### ****1-Retrieve All Areas****

* Fetches all areas stored in the Area model.

### ****2-Convert Area Boundaries into Polygons****

* Each area has a set of boundary points stored in JSON format.
* These points are decoded and used to construct a polygon.

### ****3-Check if the Given Coordinates Are Inside Any Polygon****

* The method iterates through all area polygons.
* Uses the contains() method to verify if the given coordinates fall within any polygon.
* If found inside a polygon, it sets $exist = true and exits the loop.

### ****4-Handles Errors Gracefully****

* If an error occurs, it returns a 403 response with the error message

**2.3 Gift Controller**

### 2.3.1 storeNew(Request $request) Method

🔹 **Overview**  
This method handles the process of sending a gift within the application. It validates the request, checks for the recipient, processes payment, and sends notifications.

🔹 **Functionality**

### 1-****Validates the Incoming Request****

* Ensures phone is required and has at least 10 characters.
* Ensures total\_price and payment\_method are required.
* Allows an optional recieve\_name.
* Restricts payment\_method to either 0 (Balance) or 1 (Credit Card).
* Captures an optional transaction\_id.

### 2-****Formats the Phone Number****

* Removes non-numeric characters.
* Ensures the number is formatted correctly, adding a leading 0 if necessary.

### 3-****Determines the City****

* Defaults to **"بريدة"** if no city is provided.
* Looks up the city based on the provided city\_name or city\_id.
* Returns an error if the city is not found.

### 4-****Checks for the Recipient****

* Finds the recipient (Client) using the formatted phone number.
* Prevents sending a gift to oneself.
* If the recipient exists, proceeds with gift creation.
* If the recipient does not exist, generates a verification code.

### 5-****Processes the Gift Transaction****

#### If the recipient exists:

* Creates a Gift record.
* If payment method is **balance (0)**:
  + Checks if the sender has sufficient balance.
  + Deducts the amount from the sender’s balance.
  + Credits the amount to the recipient’s balance.
  + Records the transaction in WalletOperation.
  + Sends an SMS notification.
* If payment method is **credit card (1)**:
  + Marks the gift as pending.
  + Stores the payment transaction details.

#### If the recipient does not exist:

* Creates a Gift record with a unique verification code.
* If payment method is **balance (0)**:
  + Deducts the amount from the sender’s balance.
* If payment method is **credit card (1)**:
  + Marks the gift as pending.
  + Stores the payment transaction details.
* Sends an SMS notification with a download link and gift details.

### 6-****Sends a Gift Notification****

* Sends an SMS message to the recipient with gift details, depending on the payment method.

### 2.3.2 received(Request $request) Method

🔹 **Overview**  
This method retrieves all the gifts received by the authenticated user, orders them by the latest received, and returns them in a formatted JSON response.

🔹 **Functionality**

**1-Retrieves the Received Gifts**

* Fetches all gifts where the receiver\_id matches the authenticated user’s ID.
* Orders the results by the most recent gifts (latest() method).

**2-Formats the Gifts Data**

* Calls the formatGifts method to structure the gift data before returning it.

**3-Returns a JSON Response**

* Wraps the formatted gift data in a JSON response with a 200 status code.

### 2.3.3 sent(Request $request) Method

🔹 **Overview**  
This method retrieves all the gifts sent by the authenticated user that have been successfully received by a recipient. The gifts are ordered from the most recent to the oldest and returned as a formatted JSON response.

🔹 **Functionality**

**1-Retrieves the Sent Gifts**

* Fetches all gifts where:
  + sender\_id matches the authenticated user’s ID.
  + receiver\_id is **not null**, ensuring that the gift has been received by someone.
* Orders the results by the most recent gifts (latest() method).

**2-Formats the Gifts Data**

* Calls the formatGifts method to structure the gift data before returning it.

**3-Returns a JSON Response**

* Wraps the formatted gift data in a JSON response with a 200 status code.

### 2.3.4 formatGifts($gifts) Method

🔹 **Overview**  
This private method processes a collection of Gift objects and formats them into an array with specific attributes. It ensures that sender and receiver details are correctly assigned before returning the structured data.

🔹 **Functionality**

**1-Initializes an Empty Array**

* Creates $gifts\_array to store formatted gift data.

**2-Loops Through Each Gift**

For each Gift object in the $gifts collection:

* Extracts the id, sender\_id, receiver\_id, car\_size\_id, total\_price, status, and created\_at.
* Determines the **sender's name**:
  + If the gift ID is greater than 233, it assigns $gift->sender\_name.
  + Otherwise, it uses $gift->sender->username, falling back to an empty string if unavailable.
* Assigns the **receiver's username** from $gift->receiver->username.

**3-Pushes the Formatted Data into the Array**

Each processed gift object is appended to $gifts\_array.

**4-Returns the Formatted Gift Data**

The method returns an array containing the formatted gift details.

### 2.3.5 use(Request $request) Method

🔹 **Overview**  
This method allows a user to redeem a gift and create a reservation. It validates the request, checks gift eligibility, finds the nearest representative, and creates a reservation. If successful, it updates related data and sends notifications.

🔹 **Functionality**

**1-Validates the Request Data**

* Ensures gift\_id exists in the gifts table.
* Requires lat, lng, date, and time for location and scheduling.
* Optionally includes services\_ids and a code from the coupons table.
* Returns validation errors if any exist.

**2-Checks Gift Eligibility**

* Ensures the user is the receiver\_id of the gift.
* Ensures the gift has not been used (status == 0).
* Returns an error message if the gift is invalid.

**3-Finds the Nearest Representative**

* Calls get\_nearest\_representative() to find an available representative.
* Returns an error if no representative is found.

**4-Creates a New Reservation**

* Fills Reservation model fields with request and gift details.
* Assigns car\_size\_id, total\_price, and time range (from → to).
* Sets gift\_id, payment\_type = 1, and status = 1.
* Saves the reservation.

**5-Handles Reservation Services (if any)**

* Calls saveReservationServices() to store service details.
* Updates the client’s balance using updateClientBalance().

**6-Updates Gift Status**

* Marks the gift as used via updateGiftStatus().

**7-Assigns Representative & Sends Notifications**

* Updates representative\_id and changes status to 2 if a representative is assigned.
* Sends a representative notification using representativeReservationNotification().
* Notifies the client via newReservationNotification().

**8-Returns a Response**

* **Success**: Returns a success message with formatted reservation details.
* **Failure**: Returns an error message if the reservation fails.

### 2.3.6 useCode(Request $request) Method

🔹 **Overview**  
This method allows a user to redeem a gift using a gift code and their phone number. It validates the input, verifies the gift's status, and assigns it to the correct client.

🔹 **Functionality**

**1-Validates the Request Data**

* Ensures gift\_code exists in the gifts table.
* Requires a valid phone number.
* Returns validation errors if any exist.

**2-Retrieves Gift and Client**

* Fetches the gift using gift\_code and ensures it has not been used (status == 0).
* Finds the client using the provided phone number.

**3-Verifies Phone Number**

* Ensures the provided phone number matches the one stored in the database.
* Returns an error if the phone numbers do not match.

**4-Assigns the Gift to the Client**

* If the gift is valid and the client exists, updates the gift’s receiver\_id with the client’s ID.

**5-Returns a Response**

* **Success**: Confirms that the gift has been assigned.
* **Failure**: Returns an error message if validation fails or the gift is invalid.

### 2.3.7 saveReservationServices($reservation, $services) Method

🔹 **Overview**  
This method assigns services to a reservation by attaching them to the reservation and calculating the total services cost.

🔹 **Functionality**

**1-Extracts Service IDs**

* Splits the $services string by , to get an array of service IDs.

**2-Iterates Through the Services**

* Loops through each service ID and retrieves the corresponding Service model.
* If the service exists:
  + Adds its cost to the total services cost.
  + Attaches the service to the reservation with its cost.

**3-Returns Total Services Cost**

* Returns the sum of all attached service costs.

### 2.3.8 updateClientBalance($client, $price) Method Documentation

🔹 **Overview**  
This method deducts a specified amount from a client's balance and updates their record in the database.

🔹 **Functionality**

**1-Deducts the Price from the Client's Balance**

* Decreases the balance attribute of the $client by the given $price.

**2-Saves the Updated Client Record**

* Persists the changes to the database by calling $client->save().

**3-Returns a Success Indicator**

* Always returns true after successfully updating the balance.

### 2.3.9 updateGiftStatus($gift\_id) Method

🔹 **Overview**  
This method updates the status of a gift to indicate that it has been used.

🔹 **Functionality**

**1-Retrieves the Gift Record**

* Finds the gift using the provided $gift\_id via Gift::find($gift\_id).

**2-Updates the Gift Status**

* Sets the status attribute of the gift to 1, marking it as used.

**3-Saves the Updated Gift Record**

* Calls $gift->save() to persist the changes in the database.

**4-Returns a Success Indicator**

* Always returns true after successfully updating the gift status.

**2.4 Package Controller**

### 2.4.1 index(Request $request) Method

🔹 **Overview**  
This method retrieves a list of active packages that contain at least one service of type 0. The packages are returned in descending order (latest first).

🔹 **Functionality**

**1-Retrieves Active Packages with Specific Services**

* Calls Package::active() to get only active packages.
* Uses whereHas('services', ...) to filter packages that have services of type 0 and are not soft deleted (deleted\_at = null).

**2-Formats the Retrieved Data**

* Wraps the collection of packages in PackageResource::collection() to structure the API response.

**3-Handles Exceptions**

* If any exception occurs, it catches the error and returns a JSON response with the error message and a 403 status code.

### 2.4.2 show(Package $package) Method

🔹 **Overview**  
This method retrieves and returns the details of a specific package using Laravel’s route model binding.

🔹 **Functionality**

**1-Retrieve the Package**

* Laravel automatically fetches the Package model instance from the database based on the provided id.
* If the package does not exist, Laravel will automatically return a 404 Not Found response.

**2-Transform the Response**

* Wraps the package in a PackageResource for consistent API formatting.

### 2.4.3 hasPackage(Request $request) Method

🔹 **Overview**  
This method checks if the authenticated user has an **active package subscription** (i.e., one where is\_finish is 0).

🔹 **Functionality**

**1-Retrieve User's Active Package**

* Fetches the **first active package subscription** where is\_finish = 0.

**2-Determine Subscription Status**

* If a package is found, has\_package is set to true; otherwise, it's false.

**3-Return JSON Response**

* Returns the package status in a structured JSON format.

### 2.4.4 clientPackage(Request $request) Method

#### ****🔹 Overview****

This method retrieves the **active package subscriptions** of a client, checks if they include any **services of type 0**, and returns the subscription details in a structured JSON response.

### ****🔹 Key Functionality****

1-**Fetch Active Subscriptions**

* Retrieves all **non-finished (is\_finish = 0) and active (status = 1)** subscriptions for the authenticated client.

2-**Validate Services Existence**

* If the package **does not contain services of type 0**, the method returns an early response asking the user to subscribe first.

3-**Loop Through Subscriptions & Format Response**

* Extracts **package details** such as:
  + ID, title, description, cost (with and without tax), tax amount.
  + Period of subscription.
  + Remaining days.
  + Image URL or auto-generated avatar.
  + Terms and conditions.
  + Alert message regarding location restrictions.

4-**Extract & Categorize Services**

* Loops through **subscription services** and categorizes them as:
  + **Main Services (type 0)**
  + **Additional Services (other types)**
* Includes service-specific details like:
  + ID, title, description, usage count, remaining count, and image.

5-**Return Response**

* **Only includes subscriptions that contain at least one valid service.**
* If no valid subscriptions exist, it returns an empty data array.

### 2.4.5 remainingDays($subscribtion) Method

#### ****🔹 Overview****

This method calculates the remaining days for a subscription based on its start date and total period.

#### ****🔹 Key Functionality****

### 1-Extract Subscription Start Date

* Retrieves the subscription's created\_at timestamp.
* Converts it into a Carbon instance for date calculations.

### 2-Determine the Current Date

* Gets today's date using Carbon::now().
* Calculates the difference in days between the start date and today.

### 3-Compute Remaining Days

* Subtracts the elapsed days from the subscription's total period.
* Ensures the remaining days do not drop below 0.

### 4-Return the Computed Value

* Returns the calculated remaining days as an integer.

### 2.4.6 subscriptionNew(PackageSubscriptionRequest $request) Method

🔹 **Overview**  
Handles new client subscriptions for a package, ensuring proper validation, checking payment methods, and recording wallet transactions.

**🔹 Key Functionality**

**1-City Selection & Validation**

* **Determines the city**: Defaults to "بريدة", unless specified by city\_name or city\_id.
* **Finds the city** in the database based on English or Arabic names.
* **Returns an error if the city is not found**.

**2-Check for Existing Subscription**

* **If the client already has an active subscription** (is\_finish = 0, status = 1), it returns a message:  
  ✅ "You are already subscribed to this package."
* **If the client started a subscription but didn't complete the payment**, it updates the subscription details and returns:  
  ✅ "Subscription completed successfully after payment."

**3-Handle Payment Methods**

💰 **Wallet Payment**

* Checks if the payment method is "wallet".
* **Verifies if the client has enough balance**:
  + Deducts the package cost from the client's balance.
  + Creates the package subscription.
  + Assigns package services to the client.
  + Logs the wallet operation (WalletOperation table).

📌 **Returns an error if the wallet balance is insufficient.**

📌 **Tabby Payment (Buy Now, Pay Later)**

* Checks if the package allows **Tabby** payments (verify\_tabby = 1).
* Creates the subscription and assigns services.
* Returns a success response.

📌 **Returns an error if Tabby payment is not allowed.**

**4-Final Subscription Creation (For Other Payment Methods)**

* **Creates a new package subscription** with the selected city and payment details.
* **Assigns all package services** to the client.
* **Returns success message**: "Package subscribed successfully."

**✅ Final Outcome**

* If the subscription is **valid**, it's successfully created.
* If **wallet balance is insufficient** or **Tabby is not available**, it returns an error.
* If a **subscription already exists**, it prevents duplication.

**2.5 Reservation Controller**

# 2.5.1 makeReservation(Request $request) Method

## 🔹 Overview

This method handles the complete reservation process for a client, including validation, availability checks, pricing calculations, and reservation creation with all related business logic.

## 🔹 Key Functionality

### 1-Rate Limiting Check

* Checks if client made a reservation within last 60 seconds
* Prevents spam by enforcing 1-minute cooldown between reservations
* Uses Saudi Arabia timezone (Asia/Riyadh) for accurate time comparison

### 2-Request Validation

* Validates required fields:
  + Car ID (must exist in database)
  + Location ID (must exist)
  + Date and time
  + Main service ID
  + Payment type (1,2,3, or 5)
  + Optional transaction ID
* Returns validation errors if any checks fail

### 3-Location Processing

* Determines city based on request parameters (city\_name or city\_id)
* Falls back to default city (ID 5) if not specified
* Validates city existence

### 4-Appointment Availability Check

* Creates internal request to appointment.check route
* Processes response to determine available time slots
* Converts time formats between English/Arabic as needed

### 5-Shift and Representative Assignment

* Finds available shifts based on:
  + Target date and time
  + Geographic area (using location coordinates)
  + Application settings for representative assignment
* Calculates time slots based on business rules (90-minute intervals)
* Assigns nearest available representative using geolocation

### 6-Pricing Calculations

* Calculates base service prices
* Applies discounts if:
  + Valid coupon provided
  + Online payment discount enabled in settings
* Verifies client balance for wallet payments
* Handles special cases for different payment types (1,2,3,5)

### 7-Reservation Creation

* Builds complete reservation data structure
* Sets proper status based on payment type
* Handles coupon usage tracking
* Saves additional services if provided
* Updates client balance for wallet payments

### 8-Notification & Response

* Sends notification to assigned representative
* Returns formatted reservation details
* Provides appropriate success/error messages

## 🔹 Special Considerations

* Handles complex timezone conversions (Saudi Arabia time)
* Supports multi-language (English/Arabic) for time displays
* Extensive logging of key decision points
* Flexible handling of different payment scenarios
* Geographic-based representative assignment logic
* Coupon validation with multiple usage rules

## 🔹 Return Values

* Success: Returns reservation details with success message (HTTP 200)
* Errors: Returns appropriate error messages for:
  + Rate limiting (HTTP 403)
  + Validation failures (HTTP 403)
  + No available representatives (HTTP 403)
  + Insufficient balance (HTTP 403)
  + Invalid coupon (HTTP 403)
  + General errors (HTTP 403)

# 2.5.2 makeReservationUsingPackage(Request $request) Method

## 🔹 Overview

This method handles package-based reservations, allowing clients to use their subscribed service packages for appointments. It validates package eligibility, checks service availability, and creates reservations without additional charges.

## 🔹 Key Functionality

### 1-Request Validation

* Validates required fields:
  + Car ID (must exist)
  + Location ID (must exist)
  + Date and time
  + Main service ID
  + Package ID (must exist)
* Returns validation errors if checks fail

### 2-Package Eligibility Check

* Verifies client's active subscription to the specified package
* Validates package is usable in the requested city
* Checks remaining service counts in the package

### 3-Location Processing

* Determines city based on:
  + Explicit city\_name parameter
  + city\_id parameter (with fallback to ID 5)
  + Validates city existence
* Retrieves Arabic city name for messaging

### 4- Appointment Availability Check

* Creates internal request to appointment.check route
* Processes time slot availability
* Converts between English/Arabic time formats

### 5-Representative Assignment

* Finds nearest available representative based on:
  + Location coordinates
  + Requested date/time
  + Shift availability
* Uses geographic area matching when enabled in settings

### 6-Package Service Deduction

* Reduces count for main service in package
* Reduces counts for any additional services used
* Validates sufficient service counts remain

### 7- Reservation Creation

* Creates zero-cost reservation (package-covered)
* Sets special package flags:
  + use\_package = 1
  + payment\_type = 5
  + status = 1 (pending)
* Stores all location/car details

### 8-Notification & Response

* Assigns representative if available
* Sends notifications
* Returns formatted reservation details
* Handles success/error cases

## 🔹 Special Considerations

* Strict geographic restrictions for package usage
* No payment processing (all covered by package)
* Detailed service count tracking
* Time slot calculations based on business rules
* Multi-language support (English/Arabic)
* Representative assignment logic matches standard reservations

## 🔹 Error Cases Handled

* Invalid package subscription (404)
* Service not included in package (404)
* Insufficient service counts (404)
* No available representatives (403)
* Validation failures (403)
* General errors (403)

## 🔹 Return Values

* Success: Returns reservation details with success message (HTTP 200)
* Errors: Returns appropriate error messages for:
  + Package/subscription issues (HTTP 404)
  + Availability/validation issues (HTTP 403)
  + General failures (HTTP 403)

## 🔹 Business Rules

* Packages are city-specific (cannot change cities)
* Each service type has limited uses
* Time slots follow same rules as regular reservations
* Representative assignment uses same logic as standard reservations

# 2.5.3 updateReservationWithInvoiceId(Request $request) Method

## 🔹 Overview

Updates a reservation record with payment transaction details after successful payment processing.

## 🔹 Key Functionality

### 1-Reservation Lookup

* Finds reservation by ID from request
* Returns 404 error if reservation doesn't exist

### 2-Payment Data Update

* Updates three payment-related fields:
  + transaction\_id - Payment gateway reference
  + payment\_method\_id - Payment method identifier
  + payment\_method\_name - Human-readable payment method

### 3-Response

* Returns success message (200) on update
* Returns error message (404) if reservation not found

# 2.5.4 handle\_webhook(Request $request) Method

## 🔹 Overview

Processes payment webhook notifications from myFatoraah payment gateway and updates corresponding system records based on transaction status.

## 🔹 Key Functionality

### 1-Webhook Processing

* Handles three distinct transaction types identified by CustomerReference:
  1. **Package Subscriptions**
  2. **Wallet Top-ups**
  3. **Gift Cards**
  4. **Default Case**: Regular reservations

### 2-Transaction Type Handling

#### 📦 Package Subscriptions (CustomerReference == "package")

* Updates package subscription status to active (1) on successful payment
* Stores payment reference IDs:
  + transaction\_id → InvoiceReference
  + paymentId → PaymentId

#### 💰 Wallet Operations (CustomerReference == "wallet")

* Credits user's balance with the transaction amount
* Updates wallet operation record:
  + Sets status to successful (1)
  + Stores payment reference IDs
  + Maintains audit trail

#### 🎁 Gift Cards (CustomerReference == "gift")

* Delegates to sendGift() method (separate gift card processing)

#### 🚗 Default Case (Reservations)

* Updates reservation status to confirmed (2) on success
* Stores payment reference IDs
* Triggers new reservation notification

### 3-Error Handling

* Wrapped in try-catch block
* Returns 422 error with message if processing fails
* Detailed logging of all operations

## 🔹 Return Values

* No explicit return on success (processing complete)
* Returns 422 error with message on failure

# 2.5.5 sendGift($request) Method

## 🔹 Overview

Handles successful gift card payments by updating records, crediting recipient balances, and sending notification messages. This private method is called from the payment webhook handler.

## 🔹 Key Functionality

### 1-Gift Transaction Processing

* Locates gift record using transaction ID
* Updates payment status to successful (1) on "SUCCESS" status
* Stores payment reference IDs:
  + transaction\_id → InvoiceReference
  + paymentId → PaymentId

### 2-Recipient Handling

#### For Registered Users (receiver\_id exists):

* Credits recipient's wallet balance
* Creates wallet operation record (if enabled in settings)
* Generates personalized message including recipient name

#### For Non-Registered Users:

* Formats phone number for SMS delivery
* Creates account activation message with redemption instructions
* Handles both named and anonymous gift cases

### 3-SMS Notification

* Constructs culturally-appropriate Arabic message templates
* Includes:
  + Personalized sender/recipient names (when available)
  + Gift value information
  + App download link
  + Brand tagline
* Sends via SMS gateway

## 🔹 Special Features

### 📱 Phone Number Processing

* Normalizes phone numbers:
  + Removes non-digit characters
  + Handles long numbers (keeps last 9 digits)
  + Adds leading zero when needed

### 💳 Wallet Integration

* Automatic balance crediting for registered users
* Detailed audit trail through WalletOperation records
* Configurable via application settings

### ✉️ Message Templates

Three distinct message variants:

1. For registered users with recipient name
2. For registered users without recipient name
3. For non-registered users (with phone number instructions)

## 🔹 Error Handling

* Implicit through null checks
* Extensive INFO logging at each stage
* No explicit error returns (assumes caller handles)

# 2.5.6 totalPriceAfterDiscount(request,request,services\_price) Method

## 🔹 Overview

Calculates the final price after applying coupon discounts to the main service (excluding additional services).

## 🔹 Parameters

* $request: HTTP request containing coupon code
* $services\_price: Total price of additional services

## 🔹 Key Functionality

### 1-Base Price Calculation

* Extracts main service price by subtracting additional services cost from total:
  + - * $total\_price = $request->total\_price - $services\_price;

### 2-Coupon Handling

* Checks for coupon in request
* Validates coupon:
  + Exists in database
  + Has active status (1)
* Retrieves discount percentage if valid

### 3-Discount Application

* Calculates discount amount:

($value / 100 \* $total\_price)

* Returns final price after discount:
  + - * $total\_price - discount\_amount

## 🔹 Business Rules

* Discounts only apply to main service cost
* Additional services are always full price
* Only active coupons (status=1) are considered

## 🔹 Return Value

* Returns float: Final price after discount
* Returns original main service price if:
  + No coupon provided
  + Coupon invalid
  + Coupon inactive

# 2.5.7 calServicesPrice($request) Method

## 🔹 Overview

Calculates the total cost of additional services selected during a reservation.

## 🔹 Parameters

* $request: HTTP request containing additional services IDs

## 🔹 Key Functionality

### 1-Initialization

* Sets initial services price to 0

$services\_price = 0;

### 2-Additional Services Processing

* Checks for additional services in request:

$request->has('additional\_services')

* Splits comma-separated service IDs into array:

explode(',', $request->additional\_services)

### 3-Price Calculation

* For each service ID:
  + Retrieves service record from database
  + Validates service exists
  + Accumulates service costs:

$services\_price += $service->cost;

## 🔹 Business Rules

* Only processes if additional services exist in request
* Silently skips invalid service IDs (no error thrown)
* Returns 0 if:
  + No additional services selected
  + All service IDs invalid

## 🔹 Return Value

* Returns float: Total cost of all valid additional services

# 2.5.8 service\_price\_of\_reservation($reservation) Metho

## 🔹 Overview

Calculates the total cost of all services associated with a reservation by summing individual service costs.

## 🔹 Parameters

* $reservation: Reservation model instance with loaded services relationship

## 🔹 Key Functionality

### 1-Initialization

* Initializes total price accumulator:

$services\_price = 0;

### 2-Service Cost Aggregation

* Retrieves all services attached to reservation:

$reservation->services

* Iterates through each service:

foreach ($services as $service)

* Sums the cost of each service:

$services\_price += $service->cost;

## 🔹 Business Rules

* Expects pre-loaded services relationship on reservation
* Returns 0 if no services are attached
* Handles both main and additional services together

## 🔹 Return Value

* Returns float: Total combined cost of all reservation services

# 2.5.9 saveReservationServices(reservation,reservation,services) Method

## 🔹 Overview

Attaches additional services to a reservation record while storing each service's cost at the time of reservation.

## 🔹 Parameters

* $reservation: The Reservation model instance
* $services: Comma-separated string of service IDs (e.g., "3,7,12")

## 🔹 Key Functionality

### 1-Service ID Processing

* Splits comma-separated string into individual IDs:

explode(',', $services)

### 2-Service Attachment

* For each valid service ID:
  + Retrieves the service record
  + Creates a pivot relationship with cost preservation:

attach($service\_id, ['cost' => $service->cost])

## 🔹 Business Rules

* Silently skips invalid service IDs
* Maintains historical pricing through pivot table
* Only processes if at least one valid ID exists

## 🔹 Database Impact

* Creates records in reservation\_services pivot table with:
  + reservation\_id
  + service\_id
  + cost (snapshot of price at time of booking)

## 🔹 Return Value

* Always returns true (consider changing to boolean result in future versions)

# 2.5.10 updateClientBalance(client,client,total\_price) Method

## 🔹 Overview

Deducts a specified amount from a client's wallet balance during reservation payment.

## 🔹 Parameters

* $client: Client model instance
* $total\_price: Amount to deduct (float)

## 🔹 Key Functionality

1. **Balance Update**:

$client->balance = $client->balance - $total\_price;

1. **Persists Change**

$client->save();

## 🔹 Business Rules

* Performs simple arithmetic deduction
* No validation of sufficient balance (caller must verify)
* Always saves regardless of balance result

## 🔹 Return Value

* Always returns true (consider returning new balance in future versions)

# 2.5.11 newReservationMail($reservation) Method

## 🔹 Overview

This method handles sending email notifications to administrators when a new car wash reservation is created. It retrieves system configuration and dispatches the reservation details to the configured email address.

## 🔹 Key Functionality

### 1-Retrieve System Configuration

* Fetches the primary application settings record
* Verifies email notification is configured (email\_1 exists)

### 2- Prepare Email Notification

* Uses the NewReservationMail Mailable class
* Packages all reservation details for email content
* Maintains professional email formatting standards

### 3- Dispatch Notification

* Sends to primary admin email from configuration
* Utilizes Laravel's mail queue system
* Handles both immediate and queued delivery

### 4- Silent Error Handling

* Gracefully fails if no application config exists
* Continues without interruption if email sending fails
* No explicit return value (fire-and-forget design)

# 2.5.12 newReservationNotification($reservation) Method

## 🔹 Overview

This method sends a push notification to the client's mobile device when a new car wash reservation is successfully created, using Firebase Cloud Messaging (FCM).

## 🔹 Key Functionality

### 1- Notification Initialization

* Creates a new instance of FirebasePushNotification controller
* Prepares multilingual notification content using translation files

### 2- Client Device Targeting

* Uses the client's stored device\_token for precise delivery
* Requires valid device registration token for successful delivery

### 3- Notification Content

* **Title**: "New Reservation" (localized via trans())
* **Body**: "Complete Reservation" (localized via trans())
* Follows mobile notification best practices

### 4- Silent Error Handling

* No explicit error handling (failures logged by Firebase service)
* Assumes valid device token exists
* No return value confirmation

# 2.5.13 representativeReservationNotification(reservation,request) Method

## 🔹 Overview

This method handles dual-notification functionality for reservation assignments, sending:

1. A payment-specific alert to the client
2. An assignment notification to the representative
3. Persists both notifications in the database

## 🔹 Key Functionality

### 1- Client Notification

* **Payment Type Detection**:
  + Online payments (payment\_type == 1) get special messaging
  + Other payments use standard reservation confirmation
* **Content Features**:
  + Localized titles/bodies via trans()
  + Payment-method contextual messaging

### 2- Representative Notification

* Standard assignment message
* Always uses representative-specific translations
* Contains no payment details

### 3- Notification Persistence

* Saves both notifications via saveNotification()
* Maintains complete audit trail with:
  + Timestamps
  + User references
  + Exact message content

# 2.5.14 saveNotification(clientid,clienti​d,message) Method

🔹 Overview  
This method creates persistent records of all notifications sent within the system, maintaining a complete historical record for auditing and user notification history purposes.

🔹 Key Functionality  
1- Notification Storage

* Creates new database entries for every notification
* Captures exact message content at time of sending
* Maintains recipient association through client\_id

2- Data Organization

* Stores structured notification components:
  + Title (short summary)
  + Body (full message content)
* Automatically records timestamp of creation

3- System Integration

* Supports notification history features
* Enables message auditing capabilities
* Provides data for user notification centers

# 2.5.15 formatDate($date) Method

🔹 Overview  
This utility method converts date strings between different formats, specifically designed to handle the system's standard date representation and convert it to database-friendly format.

🔹 Key Functionality  
1- Date Parsing

* Accepts dates in "Y M d" format (e.g., "2023 Nov 20")
* Uses Carbon for reliable date manipulation
* Strictly enforces expected input format

2- Format Conversion

* Transforms dates to ISO-8601 format ("Y-m-d")
* Ensures database compatibility (e.g., "2023-11-20")
* Maintains date accuracy during conversion

# 2.5.16 show($id) Method

🔹 Overview  
This API endpoint retrieves and formats detailed information about a specific reservation.

🔹 Key Functionality

1- Reservation Retrieval

* Fetches reservation by ID using Eloquent
* Includes all related model data through formatReservation()
* Verifies existence before processing

2- Localized Formatting

* Uses application's current locale
* Formats all dates/times appropriately
* Structures data for mobile app consumption

3- Response Handling

* Returns HTTP 200 even for empty results
* Standardizes JSON output structure
* Protects against null reference errors

# 2.5.17 clientCurrentReservation(Request $request) Method

🔹 Overview  
This endpoint retrieves active reservations for the authenticated user, with different logic for clients versus representatives. It filters out completed/canceled reservations and formats results for the application UI.

🔹 Key Functionality

### 1- User Type Handling

**For Clients (type == 0):**

* Fetches all non-terminal status reservations
* No date filtering applied
* Ordered by most recent

**For Representatives (type != 0):**

* Excludes additional statuses (7)
* Restricts to past 24-hour window
* Prioritizes newest reservations

### 2- Status Filtering

**Excluded Status Codes:**

* 0: Canceled
* 3: Completed
* 6: Rejected
* 7: (For reps) Payment Failed

### 3- Data Formatting

* Applies localization through formatReservation()
* Maintains consistent output structure
* Includes count metadata

# 2.5.18 clientFinishedtReservation(Request $request) Method

🔹 Overview  
Retrieves completed or canceled reservations for authenticated users, with separate logic for clients and representatives. Filters out active reservations and formats results for display.

### 🔹 Key Functionality

#### 1- User-Specific Filtering

**For Clients (type == 0):**

* Excludes active reservations (status != 2)
* Includes all past dates regardless of status
* Shows today's canceled/completed reservations

**For Representatives (type != 0):**

* Excludes rejected/failed payments (status != 6,7)
* Includes all terminal statuses
* No date restrictions

#### 2- Date/Status Logic

Complex filtering handles:

* Past-dated reservations
* Today's canceled (0)
* Today's completed (3)
* Today's rejected (6 for reps)

#### 3-Response Formatting

* Localized via formatReservation()
* Returns array + count metadata
* Standardized JSON structure

# 2.5.19 formatReservation(reservation,reservation,lang) Method

🔹 Overview  
Transforms raw reservation data into a standardized, localized format for API responses, including all related entity data and calculated values. Handles complex relationships and special formatting rules.

### 🔹 Key Functionality

#### 1-Vehicle Data Formatting

* **Color Processing**:
  + Converts color\_id to name/hex values
  + Falls back to "N/A" if missing
* **Vehicle Attributes**:
  + Combines car size with main service
  + Preserves original car number
  + Includes brand/type information

#### 2- Relationship Handling

* **Representative Data**: Safe null checks with fallbacks
* **Service Processing**: Delegates to formatServices()
* **Location Data**: Uses LocationResource when present

#### 3- Financial Formatting

* Consistent 2-decimal formatting
* Payment type text conversion
* Discount calculations display
* Invoice URL generation

#### 4- Special Logic

* Status code normalization (7→1)
* Review score processing
* Terms/conditions inclusion
* Cancellation time windo

2.5.20 formatServices($services) Method

🔹 Overview  
Transforms service records into a standardized array format with localized titles and historical pricing, ensuring consistent API responses across all reservation endpoints.

🔹 Key Functionality

1-Service Data Structuring

* **ID Preservation**:  
  Maintains original service identifiers
* **Title Localization**:  
  Uses current application locale  
  Falls back to default language
* **Cost Accuracy**:  
  Preserves historical pricing from pivot table

2-Array Formatting

* Strict output structure:

{

id: number,

title: string,

cost: number

}[]

* Consistent decimal handling

3-Relationship Processing

* Handles Eloquent pivot data
* Maintains service-reservation link
* No database re-queries

# 2.5.21 formatReservationDate($reservation) Method

🔹 Overview  
Converts reservation date strings into standardized ISO-8601 format (YYYY-MM-DD) for consistent date handling across the application.

🔹 Key Functionality

1-Date Normalization

* Parses raw date string using Carbon
* Formats to universal date standard
* Ensures database compatibility

2-Input Handling

* Accepts Reservation model
* Extracts date attribute automatically
* Processes any valid date format

3-Output Guarantees

* Always returns:

"YYYY-MM-DD" // e.g. "2023-11-20"

* Consistent string length (10 chars)

# 2.5.22 completeReservation ($reservation) Method

🔹 Overview  
Handles the end-to-end completion workflow for reservations including image uploads, status updates, commission calculations, and notifications.

🔹 Key Functionality

1-Completion Validation

* Requires completion image upload
* Validates image integrity
* Processes only existing reservations

2-Image Handling

* File upload to server
* Automatic renaming (space to underscore)
* Image resizing (75% quality)
* Error logging for processing failures

3️⃣ Financial Processing

* Investor wallet updates
* Dual commission calculation:
  + Arrival time bonus
  + Completion time bonus
* Management ratio deductions

4-Notification System

* Client completion alert
* Commission notification
* Audit trail creation

**2.5.23 resizeImage ($source, $destination, $quality) Method**

🔹 Overview  
Handles image resizing and optimization for uploaded reservation completion photos, supporting multiple image formats while maintaining quality standards.

🔹 Key Functionality

1-Image Type Detection

* Uses getimagesize() for MIME identification
* Supports 8 formats:
  + JPEG/JPG
  + PNG
  + GIF
  + WebP
  + BMP
  + WBMP
  + AVIF

2-Format-Specific Processing

* Special quality handling per format:
  + JPEG: Direct quality parameter
  + PNG: Converts to 9-point scale
  + Others: Format-native saving

3-Memory Management

* Proper resource cleanup with imagedestroy()
* Null checks for failed operations

**2.5.24 resizeImage ($source, $destination, $quality) Method**

🔹 Overview  
Handles image resizing and optimization for uploaded reservation completion photos, supporting multiple image formats while maintaining quality standards.

🔹 Key Functionality

1-Image Type Detection

* Uses getimagesize() for MIME identification
* Supports 8 formats:
  + JPEG/JPG
  + PNG
  + GIF
  + WebP
  + BMP
  + WBMP
  + AVIF

2-Format-Specific Processing

* Special quality handling per format:
  + JPEG: Direct quality parameter
  + PNG: Converts to 9-point scale
  + Others: Format-native saving

3-Memory Management

* Proper resource cleanup with imagedestroy()
* Null checks for failed operations

# 2.5.25 completeCommissionNotification ($source, $destination, $quality) Method

🔹 Overview  
Handles dual notification delivery (database + push) for commission earnings related to reservation completion, ensuring representatives are informed of their earnings.

🔹 Key Functionality

1-Notification Creation

* Formats commission details into structured message:

[

"title" => "Commissions of Reservation #{id}",

"body" => $description

]

2-Dual Delivery System

* **Persistent Storage**:
  + Saves to database via saveNotification()
  + Maintains audit trail
* **Real-Time Alert**:
  + Firebase push to representative's device
  + Uses stored device token

3-Contextual Messaging

* Includes reservation ID in title
* Preserves full commission breakdown in body

**2.5.26 addReward(Request $request) Method**

🔹 Overview  
Creates a new reward record linked to specific reservations, allowing the system to track and manage client incentives or bonuses programmatically.

🔹 Key Functionality

1-Reward Creation

* Links to existing reservation via:

'order\_id' => $request->reservation\_id

* Marks as type 0 (default reward type)
* Stores financial amount:

'total' => $request->total

2-Reservation Verification

* Implicitly validates reservation exists
* Uses reservation's client\_id for ownership

3-Financial Tracking

* Associates with invoice when provided:

'invoice\_id' => $request->invoice\_id

🔹 Business Rules

* Mandatory fields:
  + reservation\_id
  + total amount
* Default type (0) for standard rewards
* Client association via reservation

**2.5.27 changeReservationStatus (Request $request) Method**

🔹 Overview  
Handles comprehensive reservation status transitions with location verification, image processing, and audit logging for the complete reservation lifecycle management.

🔹 Key Functionality

1-Status Transition Logic

* **Arrival (4)**:
  + Records timestamp
  + Triggers notifications
* **Work Start (5)**:
  + Requires image upload
  + Processes/resizes images
  + Updates timestamps
* **Cancellation (6)**:
  + Blocks if already canceled
  + Special case handling

2-Location Verification *(Currently Disabled)*

* Calculates distance:

distance($reservation->lat, $reservation->lng, $user->lat, $user->lng)

* 200m threshold commented out

3-Image Processing

* File validation and sanitization
* Server-side resizing (75% quality)
* Error-handled workflow

**2.5.28 completeReservationNotification($reservation) Method**

🔹 Overview  
Sends real-time push notifications to clients when their reservation is marked as completed, using localized message content for clear communication.

🔹 Key Functionality

1-Notification Dispatch

* Instantiates Firebase controller
* Targets client's specific device token
* Sends bilingual alert with:

trans('notifications.complete\_reservation\_body') // Message content

trans('notifications.complete\_reservation') // Title

2-Localization Support

* Leverages Laravel's translation system
* Requires translation keys:

// resources/lang/xx/notifications.php

'complete\_reservation' => "Reservation Complete",

'complete\_reservation\_body' => "Your wash #:id is finished"

3-Minimalist Design

* Single responsibility principle
* No status checks or fallbacks
* Fire-and-forget implementation

**2.5.29 review($reservation) Method**

🔹 Overview  
Sends real-time push notifications to clients when their reservation is marked as completed, using localized message content for clear communication.

🔹 Key Functionality

1-Notification Dispatch

* Instantiates Firebase controller
* Targets client's specific device token
* Sends bilingual alert with:

trans('notifications.complete\_reservation\_body') // Message content

trans('notifications.complete\_reservation') // Title

2-Localization Support

* Leverages Laravel's translation system
* Requires translation keys:

// resources/lang/xx/notifications.php

'complete\_reservation' => "Reservation Complete",

'complete\_reservation\_body' => "Your wash #:id is finished"

**2.5.30 review($reservation) Method**

🔹 Overview  
Handles the complete review submission workflow for reservations, including validation, authorization checks, and review persistence with proper error handling.

🔹 Key Functionality

### 1- Input Validation

* **Score Validation**:

'required|integer|min:1|max:5' // 1-5 rating scale

* **Error Response**:  
  Returns 403 with validation messages

### 2- Business Rule Enforcement

* **Duplicate Review Check**:

$this->checkReview($id)

* **Reservation Verification**:
  + Existence check (404 if missing)
  + Ownership validation (403 if mismatch)

### 3- Review Creation

* **Data Persistence**:

$review->reservation\_id = $id;

$review->review\_text = $request->review\_text;

$review->score = $request->score;

* **Representative Association**:  
  Automatically links to servicing representative

### 4- Localized Responses

Uses translation keys:

* api.review\_already\_exist
* api.reservation\_not\_found
* api.review\_added\_successfully

**2.5.31 checkReview($reservation\_id) Method**

🔹 Overview  
Verifies whether a review already exists for a specific reservation to prevent duplicate submissions and maintain data integrity in the review system.

🔹 Key Functionality

### 1- Review Existence Check

* **Database Query**:

Review::where('reservation\_id', $reservation\_id)->first()

* **Null Verification**:  
  Double-check with isset() and != null

### 2- Boolean Response

* **True**: Review exists
* **False**: No review found

**2.5.32 changePaymentStatus (Request $request) Method**

🔹 Overview  
Handles payment status updates for reservations, triggering notifications and status transitions while ensuring data validity through comprehensive validation checks.

🔹 Key Functionality

### 1- Input Validation

* **Reservation Verification**:

'required|exists:reservations,id'

* **Transaction ID Requirement**:

'transaction\_id' => 'required'

* **Structured Error Response**:

{

"code": 400,

"data": "",

"message": "[validation message]"

}

### 2- Payment Processing

* **Status Update**: Changes to 2 (Payment Confirmed)
* **Dual Notification System**:

$this->newReservationNotification($reservation);

$this->representativeReservationNotification(...);

* **Payment Type Verification**: Ensures valid payment method exists

### 3- Response Handling

* **Success**:

{

"code": 200,

"message": "Status updated"

}

* **Failure**:

{

"code": 400,

"message": "Payment type not set"

}

**2.5.33 cancelRequest($reservation\_id, Request $request) Method**

🔹 Overview  
Handles reservation cancellation requests with package service restoration and comprehensive status updates, ensuring proper resource management and cancellation auditing.

🔹 Key Functionality

### 1- Package Service Restoration

* **Main Service Rollback**:

$service->count += 1;

$service->save();

* **Additional Services Handling**:  
  Iterates through all services to restore counts

### 2- Cancellation Processing

* **Status Update**: Sets to 6 (Canceled)
* **Reason Tracking**: Stores cancel\_reason
* **Error Handling**: Full try-catch wrapping

### 3- Response System

* **Success**:

{"status": 200, "message": "تم الغاء الطلب بنجاح"}

* **Errors**:

{"status": 422, "message": "[error details]"}

**2.5.34 postponeRequest($reservation\_id, Request $request) Method**

🔹 Overview  
Handles complex reservation postponement requests with time slot validation, representative reassignment, and historical tracking while maintaining business rules and availability constraints.

🔹 Key Functionality

### 1- Pre-Validation Checks

* **Reservation Status**: Must be status 2 (Confirmed)
* **Duplicate Time Prevention**:

if ($reservation->date == $request->date && $reservation->from == $request->time)

* **Date/Time Parsing**:  
  Handles both AM/PM and Arabic time formats

### 2- Availability Verification

* **Internal API Call**:

Request::create(route('appointment.check'), 'POST', $request->all())

* **Shift Processing**:
  + Area-based representative assignment
  + 90-minute slot calculations
  + Time format conversions

### 3- Representative Reassignment

* **Nearest Available Rep**:

get\_nearest\_representative()

* **Fallback Handling**: Returns 403 if no reps available

### 4- Historical Tracking

* **Original Values Preservation**:

'old\_representative' => $reservation->representative\_id,

'old\_date' => $reservation->date

* **Time Adjustment**:

$newTo->addMinutes($setting->accepting)

**2.5.35 updateLocation(Request $request) Method**

🔹 Overview  
Handles secure location updates for reservations with comprehensive validation, ensuring geographic consistency and business rule compliance.

🔹 Key Functionality

### 1- Validation Chain

* **Reservation Existence**:

Reservation::find($request->reservation\_id)

* **Location Verification**:

Locations::find($request->location\_id)

* **Geographic Checks**:

$area = $this->checkLocation(...)

if($area->id != $reservation->area\_id)

### 2- Business Rule Enforcement

* **Same Location Prevention**:

if ($reservation->location\_id == $request->location\_id)

* **Area Boundary Compliance**:

"تغيير الموقع مسموح داخل المدينة المختارة اثناء الحجز فقط"

### 3- Secure Update Process

* **Atomic Updates**:

$reservation->lat = $location->lat;

$reservation->lng = $location->long;

* **Full Transaction**: Wrapped in try-catch

**2.5.35 convertTime($timeString) Method**

🔹 Overview  
Transforms time strings between 12-hour and 24-hour formats while handling both English (AM/PM) and Arabic (ص/م) time indicators with comprehensive format validation.

🔹 Key Functionality

### 1- Bilingual Time Processing

* **Indicator Recognition**:

/(PM|م)/iu // Case-insensitive matching

* **Format Cleaning**:

preg\_replace('/\s?(AM|PM|ص|م)\s?/iu', '', $timeString)

### 2- Smart Conversion Logic

* **PM/م Handling**:

if ($hour < 12) {

return Carbon::parse($cleanedTime)->addHours(12)

}

* **24h Format Detection**:

/^([01]?\d|2[0-3]):[0-5]\d$/

* **Fallback Conversion**:

Carbon::parse($timeString)->format('H:i')

**2.5.36 generateInvoice($reservation\_id) Method**

🔹 Overview  
Generates printable invoices for reservations by compiling service details and pricing information into a structured view, with comprehensive error handling.

🔹 Key Functionality

### 1- Reservation Validation

* **Existence Check**:

Reservation::find($reservation\_id)

* **Error Response**:

returnError(404, \_\_('api.reservation\_not\_found'))

### 2- Financial Calculation

* **Service Pricing**:

$this->service\_price\_of\_reservation($reservation)

* **View Data Preparation**:

compact('reservation', 'services\_price')

### 3- Invoice Generation

* **Template Rendering**:

view('dashboard.reservation.invoice', ...)

* **Structured Output**: Returns HTML/PDF-ready content

**2.5.37 get\_nearest\_representative\_based\_on\_client\_area($lat, $lng, $date, $from) Method**

🔹 Overview  
Identifies and assigns the most suitable service representative based on geographic proximity, shift availability, and real-time scheduling constraints.

🔹 Key Functionality

### 1- Geographic Filtering

* **Area Determination**:

$area = $this->checkLocation($lat, $lng)

* **Representative Pooling**:

$area->representatives()->get()

### 2- Temporal Processing

* **Time Format Conversion**:  
  Handles Arabic/English time indicators

str\_replace(['م', 'ص'], ['PM', 'AM'], $from)

* **Shift Matching**:

Shift::where('day', $day->format('l'))

->whereTime('from', '<=', $time)

->whereTime('to', '>=', $time)

### 3- Availability Verification

* **Conflict Checking**:

Reservation::where(['representative\_id' => $representative->id, ...])->first()

* **Distance Calculation**:  
  Implements proximity-based selection

**2.5.38 getNextReservation(Request $request) Method**

🔹 Overview  
Retrieves and formats the next upcoming reservation for representatives, with intelligent time handling and prioritization of today's confirmed bookings.

🔹 Key Functionality

### 1- Authentication & Authorization

* **Representative Verification**:

if (! $user || $user->type != 1)

* **Error Response**:

returnError(401, \_\_('api.unauthorized'))

### 2- Time-Smart Filtering

* **Current Time Baseline**:

$currentTime = Carbon::now()

* **Date Constraint**:

whereDate('date', Carbon::today())

* **Status Filter**: Only status 2 (Confirmed)

### 3- Advanced Time Processing

* **Format Correction**:

if (strpos($time,'PM') && intval(explode(':',$time)[0])>12)

* **Comparative Sorting**:

->sortBy(function($reservation){...})

### 4- Structured Response

* **Single Reservation Formatting**:

$this->formatReservation($next\_reservation, "ar")

* **Empty State Handling**:

\_\_('api.no\_next\_reservation\_found')

**2.5.39**  **nameOfMainService(Reservation $reservation)Method**

🔹 Overview  
Provides localized naming for a reservation's main service, with fallback to default values when no specific service is assigned or when dealing with the base wash service.

🔹 Key Functionality

### 1- Default Service Handling

* **Base Case Detection**:

$reservation->main\_service\_id == 1 || == null

* **Bilingual Fallbacks**:

app()->getLocale() == 'ar' ? "الغسيل الاساسي" : "main service"

### 2- Localized Service Names

* **Dynamic Translation**:

$reservation->mainService->getTranslation('title', app()->getLocale())

* **Relationship Access**:  
  Requires mainService relationship

**2.6 Reward Controller**

**2.6.1** **addReward(storeRewardRequest $request) Method**

🔹 Overview  
Handles reward transactions between clients and representatives with dual wallet operations, supporting both wallet-based and invoice-based payments while maintaining comprehensive financial auditing.

🔹 Key Functionality

### 1- Payment Type Handling

* **Wallet Transactions**:

if (ctype\_alpha($request->invoice\_id) && strtolower($request->invoice\_id) == "wallet"

* + Balance verification
  + Real-time fund transfer
* **Invoice Payments**:

Reward::create([...]) // Standard reward creation

### 2- Financial Safeguards

* **Balance Check**:

$client->balance >= $request->total

* **Atomic Operations**:

$client->balance -= $request->total;

$representative->balance += $request->total;

### 3- Audit Trail

* **Dual Wallet Records**:

WalletOperation::create([...]) // Both client and rep entries

* **Detailed Descriptions**:  
  Includes timestamps, amounts, and references

**2.7 Service Controller**

**2.7.1** **index(Request $request) Method**

🔹 Overview  
Retrieves and formats active services for a specific city, with flexible city identification through either name or ID, falling back to a default city (Buraidah) when unspecified.

🔹 Key Functionality

### 1- City Identification

* **Multi-Method Lookup**:
  + By city name (Arabic/English):

City::where('name->en','LIKE',...)->orWhere('name->ar','LIKE',...)

* + By city ID:

City::where('id', $request->city\_id)

* + Default fallback (Buraidah - ID 5):

City::where('id', 5)

### 2- Service Filtering

* **City-Specific Pricing**:

whereHas('cityservicesCost', fn($q) => $q->where('city\_id', $city\_id))

* **Active Main Services**:

where(['type' => 0, 'status' => 1])

### 3- Response Formatting

* **Structured Output**:

$this->formatServices($services, $city)

* **Error Handling**: Returns 403 for invalid cities

**2.7.2** **show(Request $request, $id) Method**

🔹 Overview  
Retrieves and formats a single service's details with validation and status checking, ensuring only active services are returned to clients.

🔹 Key Functionality

### 1- Request Validation

* **Service Existence Check**:

'id' => 'exists:services,id'

* **Error Response**:

return response()->json($Validated->messages(), 403)

### 2- Service Retrieval

* **Active Status Filter**:

where('status', 1)

* **Single Result Formatting**:

$this->formatService()

### 3- Structured Response

* **Success**: Returns 200 with service object
* **Failure**: Returns 403 with validation errors

**2.8 Tabby Controller**

**2.8.1** **checkout(Request $request) Method**

🔹 Overview  
Processes package subscription payments through Tabby's payment gateway, handling package validation, subscription management, and payment session creation with comprehensive error handling.

🔹 Key Functionality

### 1- Request Validation

* **Payment Details**:

'payment.amount' => 'required|numeric'

* **Buyer Information**:

'payment.buyer.email' => 'required|email'

* **Merchant URLs**:

'merchant\_urls.success' => 'required|url'

### 2- Subscription Management

* **Existing Subscription Check**:

PackageSubscription::where(...)->latest()->first()

* **New Subscription Creation**:

PackageSubscription::create([...])

### 3- Tabby Integration

* **Order Payload Construction**:

$validated['payment']['order'] = [...]

* **API Request Handling**:

Http::withHeaders(...)->post('https://api.tabby.ai/api/v2/checkout',...)

**2.8.2** **handle (Request $request) Method**

🔹 Overview  
Processes Tabby payment gateway webhook events, handling payment authorization, subscription activation, and loyalty program updates with comprehensive status tracking and error management.

🔹 Key Functionality

### 1- Payment Status Handling

* **Authorized Payments**:

case 'authorized':

$this->capturePayment(...)

* **Rejected/Failed**: Returns localized error messages
* **Canceled**: User-initiated cancellation feedback

### 2- Subscription Management

* **Status Update**:

$subscription->update(['status' => 1])

* **Service Allocation**:

ClientSubscriptionService::create([...])

### 3- Loyalty Program

* **Points Calculation**:

$loyalty = $amount / 10

* **Client Update**:

$client->loyalty\_level += $loyalty

**2.8.3** **capturePayment($paymentId, $data) Method**

🔹 Overview  
Executes payment capture requests to Tabby's API after successful authorization, finalizing transactions and ensuring fund settlement while maintaining detailed error logging.

🔹 Key Functionality

### 1- API Request Configuration

* **Authentication**:

'Authorization' => 'Bearer ' . env('TABBY\_WEBHOOK\_SECRET')

* **Endpoint Construction**:

"https://api.tabby.ai/api/v2/payments/{$paymentId}/captures"

### 2- Payment Capture Execution

* **Amount Specification**:

'amount' => $data['amount']

* **Description Forwarding**:

'description' => $data['description']

### 3- Response Handling

* **Error Logging**:

Log::error('Failed to capture payment', [...])

* **Simplified Return**: Always returns true

**2.8.4** **capturePayment($paymentId, $data) Method**

🔹 Overview  
Executes payment capture requests to Tabby's API after successful authorization, finalizing transactions and ensuring fund settlement while maintaining detailed error logging.

🔹 Key Functionality

### 1- API Request Configuration

* **Authentication**:

'Authorization' => 'Bearer ' . env('TABBY\_WEBHOOK\_SECRET')

* **Endpoint Construction**:

"https://api.tabby.ai/api/v2/payments/{$paymentId}/captures"

### 2- Payment Capture Execution

* **Amount Specification**:

'amount' => $data['amount']

* **Description Forwarding**:

'description' => $data['description']

### 3- Response Handling

* **Error Logging**:

Log::error('Failed to capture payment', [...])

* **Simplified Return**: Always returns true

**2.8.4** **getPaymentDetailsFromTabby($paymentId) Method**

🔹 Overview  
Retrieves comprehensive payment details from Tabby's API using a payment reference ID, serving as a critical verification step in the payment processing workflow.

🔹 Key Functionality

### 1- API Request Configuration

* **Secure Authentication**:

'Authorization' => 'Bearer ' . env('TABBY\_WEBHOOK\_SECRET')

* **Endpoint Construction**:

"https://api.tabby.ai/api/v2/payments/{$paymentId}"

### 2- Response Handling

* **Success Case**: Returns full JSON payment details
* **Error Case**:

Log::error('Failed to retrieve payment details', [...])

Returns null with detailed logging

### 3- Data Verification

* **Status Validation**: Used to confirm AUTHORIZED state
* **Amount Verification**: Ensures correct capture amount
* **Order Reference**: Links to local subscription records

**2.8.4** **registerWebhook($webhookUrl) Method**

🔹 Overview  
Registers a system webhook URL with Tabby's payment gateway to receive real-time payment notifications, enabling automated payment status updates and subscription management.

🔹 Key Functionality

### 1- Webhook Registration

* **API Endpoint**:

'https://api.tabby.ai/api/v1/webhooks'

* **Payload Structure**:

[

'url' => $webhookUrl,

'is\_test' => true // Toggle for production

]

### 2- Secure Authentication

* **Bearer Token**:

env('TABBY\_WEBHOOK\_SECRET')

* **HTTPS Enforcement**:  
  Requires valid SSL certificate

### 3- Response Handling

* **Success**: Returns true
* **Failure**: Returns false with silent logging

**2.9 Material Requests Controller**

**2.9.1** **getMaterials() Method**

🔹 Overview  
Retrieves and formats all available materials with positive inventory counts, providing a standardized API response for client applications.

🔹 Key Functionality

### 1- Data Retrieval

* **Inventory Filter**:

Materials::where('count', '>', 0)

* **Collection Formatting**:

MaterialResource::collection()

### 2- Response Handling

* **Success**: Returns 200 with materials array
* **Failure**: Returns 403 with error messa

**2.9.3 getRequests(Request $request) Method**

🔹 Overview  
Retrieves and formats material requests based on user role, providing different data views for administrators and representatives with proper access control.

🔹 Key Functionality

### 1- Role-Based Filtering

* **Admin View (type=3)**:

whereHas('client', fn($q) => $q->where('admin\_id', $user->admin\_id))

* **Representative View**:

where('represetive\_id', $user->id)

### 2 Data Organization

* **Chronological Order**:

latest()

* **API Resource Formatting**:

MaterialRequestResource::collection()

### 3 Response Handling

* **Success**: 200 with material requests
* **Failure**: 403 with error message

**2.9.3 addRequest(Request $request) Method**

🔹 Overview  
Handles material request submissions from representatives, validating inventory availability and updating stock levels while enforcing strict access controls.

🔹 Key Functionality

### 1- Request Validation

* **Mandatory Fields**:

'count' => 'required|numeric|integer',

'material\_id' => 'required|exists:materials,id'

* **Error Response**:

return response()->json(['message' => $validator->errors()->first()], 422)

### 2- Authorization Check

* **Representative Only**:

if ($user->type != 1) return 403

* **Arabic Message**: "غير مصرح لك بالدخول"

### 3- Inventory Management

* **Stock Verification**:

if ($material->count < $request->count) return 422

* **Atomic Update**:

$material->decrement('count', $request->count)

**2.9.4 accept(Request $request) Method**

🔹 Overview  
Processes material request approvals by administrators, updating request status and triggering notifications while enforcing strict authorization checks.

🔹 Key Functionality

### 1- Request Validation

* **Mandatory ID Check**:

'material\_request\_id' =>'required|exists:material\_requests,id'

* **Validation Response**:

return response()->json(['message' => $validator->errors()->first()], 422)

### 2 Authorization Layer

* **Admin-Only Access**:

if ($user->type != 3) return 403

* **Localized Message**: "غير مصرح لك بالدخول"

### 3- Request Processing

* **Status Update**:

$materialRequest->update(['status' => 1])

* **Notification Dispatch**:

SendNotification::dispatch(...)->delay(now()->addSeconds(2))

**2.9.5 reject(Request $request) Method**

🔹 Overview  
Handles material request rejections by administrators, restoring inventory counts and notifying representatives while maintaining strict access controls.

🔹 Key Functionality

### 1- Request Validation

* **Mandatory Fields**:

'material\_request\_id' =>'required|exists:material\_requests,id'

* **Validation Response**:

return response()->json(['message' => $validator->errors()->first()], 422)

### 2- Authorization Enforcement

* **Admin-Only Access**:

if ($user->type != 3) return 403

* **Localized Message**: "غير مصرح لك بالدخول"

### 3- Rejection Processing

* **Status Update**:

$materialRequest->update(['status' => 2]) // Status 2 = Rejected

* **Inventory Restoration**:

$Material->update(['count' => $Material->count + $materialRequest->count])

* **Async Notification**:

SendNotification::dispatch(...)->delay(now()->addSeconds(2))

**3.0 Neighborhood Controller**

**3.0.1 check(Request $request) Method**

🔹 Overview  
Validates geographic coordinates and checks for existing neighborhood coverage, providing clear API responses about service availability in specific locations.

🔹 Key Functionality

### 1- Coordinate Validation

* **Mandatory Fields**:

'lat' => 'required', 'lng' => 'required'

* **Error Response**:

return response()->json($Validated->messages(), 403)

### 2- Location Processing

* **Neighborhood Identification**:

get\_neighborhood\_name\_multi\_language\_check\_api()

* **Slug Generation**:

Str::slug($neighborhood\_name['en'])

### 3- Coverage Verification

* **Database Check**:

Neighborhood::where('slug', 'like', '%'.$neighborhood\_slug.'%')

* **Response Logic**:
  + 200: Neighborhood exists
  + 403: No coverage

**3.1 Notification Controller**

**3.1.1 index(Request $request) Method**

🔹 Overview  
Retrieves and combines all notifications for the authenticated client from multiple sources, presenting them in reverse chronological order through a unified API response.

🔹 Key Functionality

### 1- Multi-Source Aggregation

* **Primary Notifications**:

Notification::where('client\_id', $client\_id)->get()

* **Detailed Notifications**:

NotificationDetail::where('client\_id', $client\_id)->get()

### 2- Data Processing

* **Formatting**:

$this->formatNotifications()

* **Chronological Sorting**:

array\_reverse($all\_notifications)

### 3- Response Structure

* **Unified Output**:

array\_merge($notifications, $notifications\_details)

* **Consistent Format**:

['notifications' => [...]]

**3.1.2 formatNotifications($notifications) Method**

🔹 Overview  
Transforms raw notification records into a standardized array format for API responses, ensuring consistent structure across all notification types and sources.

🔹 Key Functionality

### 1- Data Transformation

* **Field Selection**:

'id', 'title', 'body', 'created\_at'

* **Array Conversion**:

array\_push($notifications\_array, [...])

### 2- Structure Standardization

* **Uniform Output**:

[

id: int,

title: string,

body: string,

created\_at: timestamp

]

* **Type Preservation**: Maintains original data types

### 3- Bulk Processing

* **Collection Handling**: Processes multiple notifications
* **Empty Input Safety**: Returns empty array if null

**3.1.2 delete(Request $request) Method**

🔹 Overview  
Handles secure deletion of client notifications with ownership verification, ensuring users can only remove their own notifications through validated requests.

🔹 Key Functionality

### 1- Request Validation

* **Mandatory Field**:

'notification\_id' => 'required'

* **Error Response**:

return response()->json($Validated->messages(), 403)

### 2- Ownership Verification

* **Client-Specific Query**:

where('client\_id', $client\_id)

* **Existence Check**:

if (isset($notification) && $notification != null)

### 3- Secure Deletion

* **Database Operation**:

$notification->delete()

* **Localized Responses**

trans('api.notification\_deleted\_successfully')