Parakeet API: High-Level Codebase Overview

1. Project Purpose

The Parakeet API is the backend system for the Parakeet application, a platform designed to streamline email outreach and management. It's built using **PHP** and the **Laravel framework (version 10.x)**, serving a **React-based front-end**.

The core functionalities include:

- **Email Campaign Management:** Enabling users to create, schedule, send, and track outreach email campaigns using multiple integrated email accounts.
- **Workspaces:** Providing a collaborative environment where users can organize their projects, team members, and associated email accounts.
- **Mailhub:** A centralized inbox feature that allows users to read, manage, and organize emails from all their connected accounts in one place.

2. Main Components and Interactions

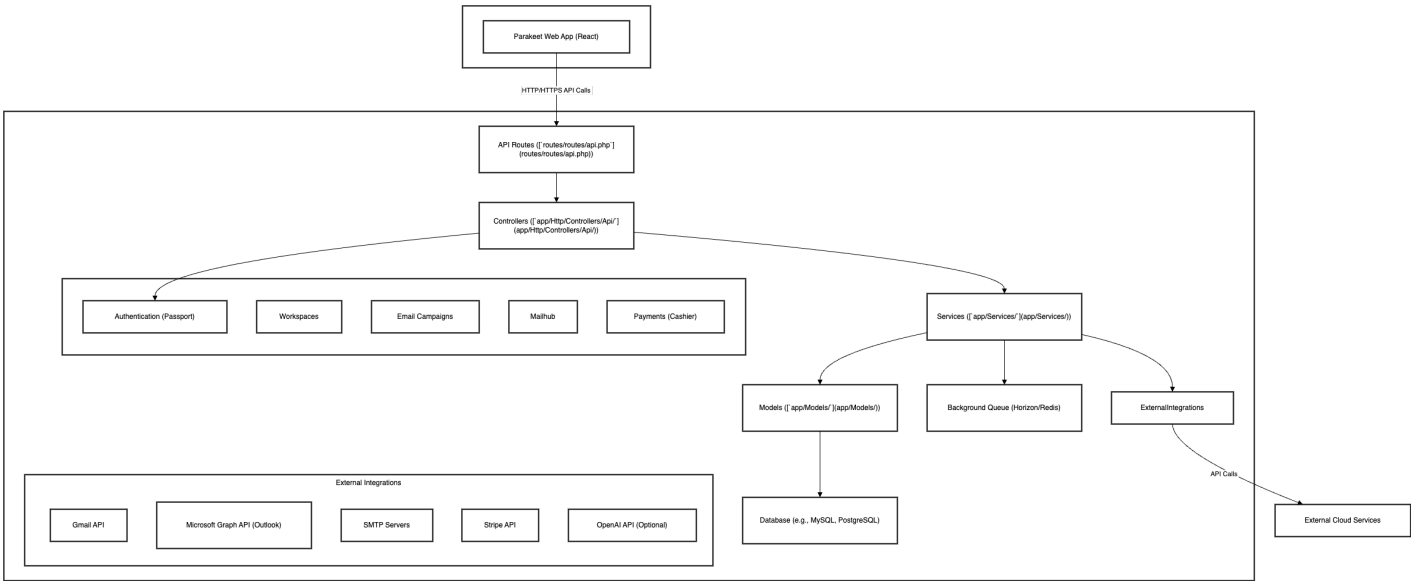
The application follows a standard Laravel structure, leveraging its Model-View-Controller (MVC) pattern, though for an API, the "View" is primarily the JSON response. Key components include:

- **Core Framework & Language:**
  - **PHP ^8.1**
  - **Laravel Framework 10.x**: Provides the foundational structure, routing, ORM (Eloquent), authentication, and other essential web development features. (`composer.json`, `config/app.php`)
- **API Routing (`routes/routes/api.php`):**
  - Defines all the accessible API endpoints. Requests from the React front-end are directed here.
  - Routes are grouped by functionality (e.g., `auth`, `workspace`, `campaign`, `mailhub`).
- **Controllers (`app/Http/Controllers/Api/`):**
  - Handle incoming API requests, validate data, and orchestrate responses.
  - Examples: `Api/Campaign/CampaignController.php`, `Api/Workspace/WorkspaceController.php`, `Api/Mailhub/MailHubController.php`.
- **Services (`app/Services/`):**
  - Contain the core business logic of the application. Controllers delegate tasks to these services.
  - Organized by domain (e.g., `app/Services/Campaign/CampaignService.php`, `app/Services/Workspace/WorkspaceService.php`, `app/Services/MailHub/MailboxService.php`).

    - Services for ESP (Email Service Provider) integrations like Gmail, Outlook, and SMTP are found in `app/Services/ESP/`.
- **Models (`app/Models/`):**
    - Eloquent models representing database tables and their relationships.
    - Examples: `Campaign.php`, `Workspace.php`, `User.php`, `Emails.php`.
- **Database:**
    - The primary data store (likely MySQL or PostgreSQL, typical for Laravel). Migrations in `database/migrations/` define the schema.
- **Authentication & Authorization:**
    - **Laravel Passport** (`laravel/passport`) is used for API token-based authentication.
    - **Spatie Laravel Permission** (`spatie/laravel-permission`) manages roles and permissions.
    - Relevant controllers: `Api/Auth/AuthController.php`, `Api/Role/RoleController.php`.
- **Email Campaign Management:**
    - This is a major module with extensive logic for creating campaigns, managing leads (`CampaignLeads.php`), sequences (`CampaignSequences.php`), scheduling (`CampaignSchedule.php`), and tracking (`CampaignTrackEmail.php`).
    - Services like `ProcessCampaignService.php` handle the execution of campaigns.
- **Mailhub & ESP Integration:**
    - Connects to external email services like Gmail (`google/apiclient`), Outlook (`dcblogdev/laravel-microsoft-graph`), and generic SMTP (`php-imap/php-imap`).
    - Services in `app/Services/ESP/` and `app/Services/MailHub/` manage fetching, sending, and organizing emails.
    - Webhooks (e.g., `routes/routes/api.php:73` for Gmail) are used for real-time email updates.
- **Workspaces:**
    - Allows users to manage distinct environments with their own accounts, campaigns, and team members (`WorkspaceUsers.php`).
- **Background Processing:**
    - **Laravel Horizon** (`laravel/horizon`) is used for managing Redis queues. This is crucial for handling time-consuming tasks like sending email blasts, email warming, and processing incoming emails without blocking user requests. **Predis** (`predis/predis`) is used as the Redis client.
- **Payment & Subscriptions:**
    - **Laravel Cashier** (`laravel/cashier`) is integrated for handling subscriptions and payments, likely with Stripe.
    - Models: `Plans.php`, `Payments.php`, `UserCredits.php`.
    - Controllers: `Api/Payment/PaymentController.php`.
- **Onboarding & User Experience:**
    - Features to guide new users, potentially using AI for campaign generation (`openai-php/client`).
    - Controllers: `Api/Boarding/OnboardingController.php`.
- **Logging, Debugging & API Documentation:**
    - **Sentry** (`sentry/sentry-laravel`) for error tracking.

- **Laravel Telescope** (`laravel/telescope`) for local debugging (though potentially disabled in `extra.laravel.dont-discover` in `composer.json`).
- **Dedoc Scramble** (`dedoc/scramble`) for API documentation generation.
- **OwenIt Laravel Auditing** (`owen-it/laravel-auditing`) for tracking model changes.

3. Interaction Flow (Simplified)



1. The **React front-end** interacts with the Parakeet API by making HTTP requests to defined endpoints.
2. **API Routes** map these requests to specific **Controller** methods.
3. **Controllers** validate input and delegate business logic to **Services**.
4. **Services** perform operations, interacting with **Models** (which represent database tables) for data persistence and retrieval.
5. For long-running tasks (e.g., sending email campaigns), Services dispatch jobs to the **Background Queue** (managed by Horizon and Redis).
6. Services also handle communication with **External Integrations** like Gmail, Outlook, Stripe, etc.
7. Finally, Controllers return JSON responses to the front-end.

This overview should provide a solid understanding of the Parakeet API's architecture, its main components, and how they interact. The use of Laravel provides a robust and scalable foundation for the application's features.

Revision #3
Created 10 June 2025 13:39:03 by Vivek Yadav
Updated 13 June 2025 06:27:46 by Vivek Yadav