Описание приложения

Приложение позволяет:

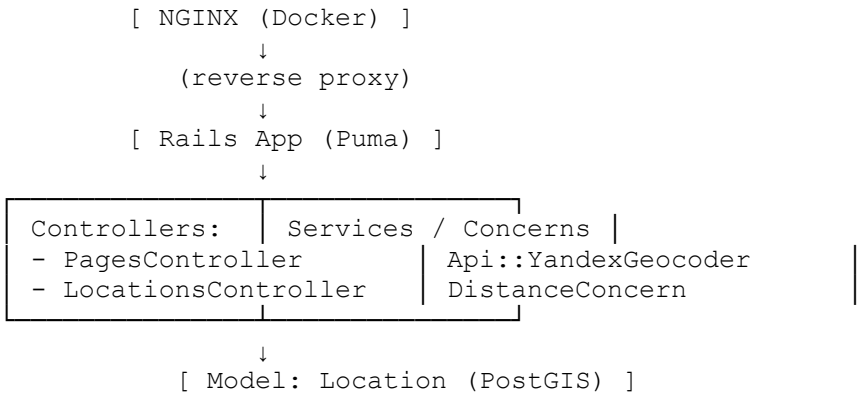
- искать города и населенные пункты через API Яндекс Геокодер;
- сохранять в БД названия и координаты локаций;
- выбирать две сохранённые локации и вычислять прямое расстояние между ними;
- пользоваться web версией (SPA с **Turbo Stream)**, и работать с REST API.

Исходный код проекта доступен в репозитории:
https://github.com/grelm21/distance_location

Используемые технологии

Компонент	**Технологии и инструменты**
Backend	Ruby on Rails (версия 7.x), PostgreSQL + PostGIS
Геокодинг	Api::YandexGeocoder (HTTParty)
Расчёт геодистанций	RGeo
SPA-интерфейс	Hotwire Turbo Stream
Контейнеризация	Docker + docker-compose
Веб-сервер	**NGINX**, проксирующий запросы на Rails

Архитектура приложения



- **NGINX** обслуживает статические файлы и проксирует запросы к Puma/Rails.
- **LocationsController** единообразно поддерживает **Turbo Stream** (если web) и **JSON** (если API).
- API геокодера и расчёты вынесены в отдельные сервисы/консерны.

Ключевые архитектурные решения

1. NGINX вместо Apache

- высокая производительности при работе с большим числом одновременных соединений,
- простота конфигурации,
- лучшая совместимости с Docker и Rails через Puma/Unix socket,
- меньшее потребление ресурсов по сравнению с Apache (особенно для SPA).

2. `Api::YandexGeocoder` — расширяемый сервис

- Обёрнут в самостоятельный сервис-класс (`lib/api/yandex_geocoder.rb`).
- Конфигурируется через переменные окружения (`YA_GEO_KEY`, `YA_GEO_BASE_URL`).
- При включении или смене логики работы API изменения не затронут контроллеры.
- Поддерживает фильтрацию по `kind` (“locality”, “province” и др.).

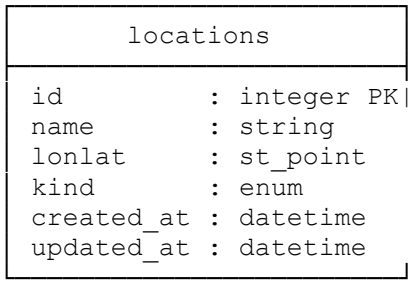
3. `DistanceConcern` — логика в отдельном модуле

- Метод `calculate_distance` реализует логику на основе RGeo/PostGIS.
- Централизованная обработка ошибок через `DistanceHandler` (билдер кастомных ошибок). При невозможности расчета дистанции приложение выбросит и обработает ошибку.
- Простота расширения: можно добавлять новые методы расчёта (например, через маршруты или внешние сервисы) и обработчики ошибок.

4. Hotwire Turbo Stream для SPA

- При действиях (поиск, добавление, удаление, расчёт) возвращаются ответы Turbo Stream и JSON.
- Обновление частей страницы (partial) без перезагрузки.
- Позволяет интерфейсу оставаться легковесным и отзывчивым.

Схема данных (ER-диаграмма)



- `lonlat`: географическая точка в формате PostGIS.
- `created_at`: фиксирует время записи (можно отображать в UI и использовать для отчетов).
- `kind` позволяет классифицировать локации по типам (например, `town`, `city`, `airport`, `station` и др.) — сейчас задан один тип `town` по умолчанию.
    - Легко расширить список типов без изменения структуры БД.
    - Возможность задавать специфичную бизнес-логику и валидации для каждого типа.
    - Улучшает читаемость и удобство работы с записями.

Масштабирование через паттерн Factory

Если в будущем появятся разные типы локаций (например, `CityLocation`, `AirportLocation`, `StationLocation`), можно масштабировать приложение с помощью внедрения паттернов, например, паттерна Factory для модели Location:

```
ruby
  def self.factory(kind, params)
    klass = case kind.to_sym
      when :town then TownLocation
      when :city then CityLocation
      when :airport then AirportLocation
        else Location
      end
```

В контроллере:

```
ruby
@location = LocationFactory.build(params[:location_kind], location_params)
```

Это позволит:

- иметь различные валидации/парсеры для разных типов;
- расширять логику без изменения базового `LocationsController` и модели `Location`.

API-интерфейс

Все основные операции доступны как через Turbo-интерфейс, так и через JSON-запросы.

Действие	HTTP Метод	URL	Параметры
Поиск городов	POST	`/locations/search`	`query`: название
Добавление локации	POST	`/locations`	`location[name]`, `raw_lonlat`
Удаление локации	DELETE	`/locations/:id`	`id`
Расчёт расстояния	POST	`/locations/distance`	`from_id`, `to_id`
Получить все локации	GET	`/locations.json`	—
Получить локацию	GET	`/locations/:id.json`	`id`

Дополнительные возможности расширения

- **Тестирование**: встроить **RSpec**, **FactoryBot**, мокирование Yandex API.
- **Кэширование**: с помощью Redis (особенно для запросов геокодирования).
- **Интеграция**: авторизация (Devise)
- **UX-функции Turbo**: lazy-loading, модальные окна, infinite scroll.
