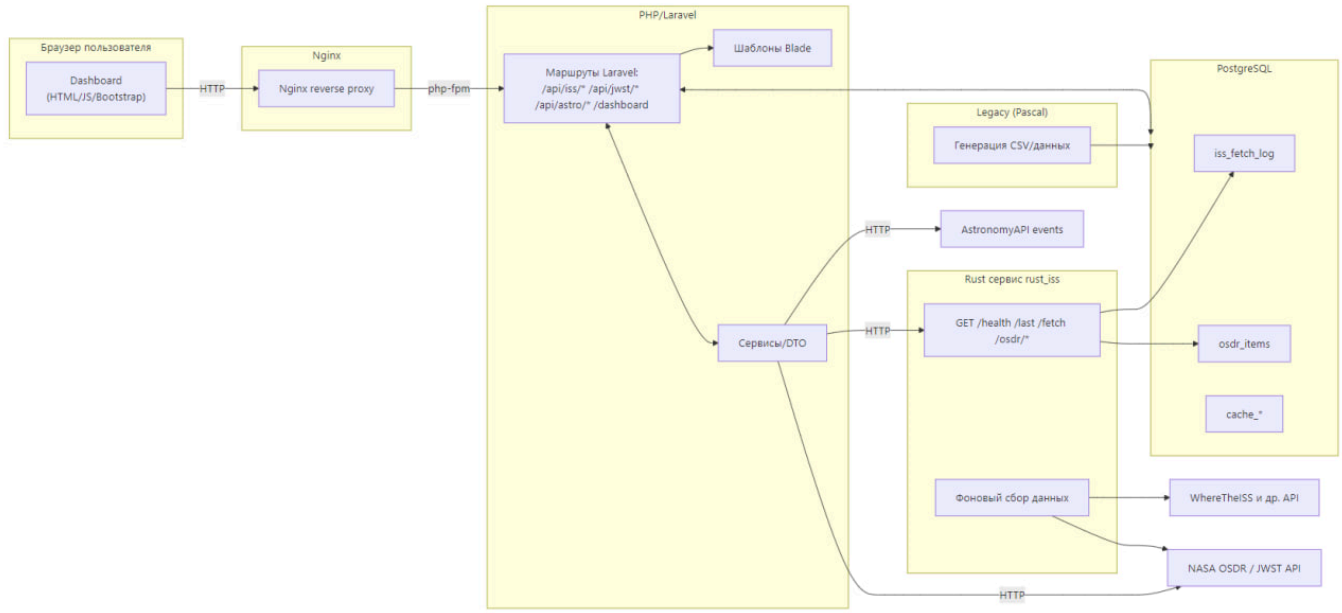
Введение в проект

⚓ Ссылка на gitverse.

Мы представители компании "Кассиопея" и мы просим вас помочь в оптимизации проекта. На данный момент он выглядит как «распределённый монолит», его основная задача - реализация и сбор данных для космических данных. Наш сервис на Rust собирает внешние данные из открытых API (мы не помним из каких, но оставляем вам все ключи по ним в коде), там же мы пишем их в PostgreSQL и отдаёт REST-эндпойнты для веб-приложения. На фронтенде у нас Laravel, это дашборд с картами, графиками, галереями, где основная часть контента приходит из БД и/или внешних API. У нас есть старая часть, которая используется нами с 2008 года (легаси-модуль), мы его написали не знаем на каком языке, но он периодически генерирует CSV и/или промежуточные данные для БД. Основная БД у нас PostgreSQL. Nginx в качестве прокси.

Схема взаимодействий



🔔 Легенда карты

- **rust_iss** — rust-сервис: опрос внешних космических API (ISS, NASA OSDR и др.), периодическая запись сырых данных/логов в PostgreSQL, собственные REST-ручки для выборок/триггеров.
- **php_web** — веб-сайт на Laravel + Bootstrap с Dashboard’ами и API-прокси-ручкам
- **iss_db** — PostgreSQL (хранение логов, кэшей и производных данных).
- **pascal_legacy** — легаси-утилита (Pascal), периодически генерирует CSV и/или записи для БД.

- **nginx** — фронтовой reverse-proxy (HTTP 80 → php-fpm).

Цель доработки информационной системы

Мы просим вас:

1. Доработать на фронтенд (Laravel-решение):
    1. Убрать все лишние модули, мешающие работе;
    2. Добавить анимацию;
    3. Разделили логику на контексты - каждой бизнес-функции должна быть отдельная страница;
    4. Добавить CSS-визуализацию;
    5. Добавить гибкие дашборды с фильтрацией данных: по возрастанию, по убыванию - с учетом даты и выбором столбца в таблице;
    6. Добавить поиск совмещенный с фильтрами по ключевым словам;
2. Доработать Pascal-Legacy:
    1. Переписать генерацию CSV файла в следующий вид:
        1. Время и даты timestamp;
        2. Логические блоки: ИСТИНА и ЛОЖЬ;
        3. Числа - числовой формат;
        4. Строки - текст;
    2. Визуализация CSV в виде полноценной таблицы;
    3. Реализация алгоритма в формат *.xlsx с подстановкой значений (дата и время);
3. Доработка бэкенд-части в Docker:
    1. Добавить Rate-Limit;
    2. Добавить в распределенный монолит Redis;
    3. Добавить валидацию данных в виде отдельных классов (для удобства проверки);

> **Важно:** функциональный охват не меняем. Любые визуальные улучшения допустимы, но **поведение API и страниц остаётся совместимым**.

Мы часто общаемся с командами и точно знаем, что существуют какие-то паттерны, которые повышают производительность и есть алгоритмы которые ускоряют обработку. Можете, пожалуйста, добавить их? Просто хотелось бы увидеть зрелое и полное контекстом решение в бизнесе.

Вам необходимо проверить как работает наш набор сервисов и дать обратную связь в виде диаграмм, блок-схем, тестов. Давайте про каждый поговорим rust_iss (Axum + SQLx).

В нём у нас должен был быть следующий подход в плане работы. Не знаем точно, но вроде программист всё сделал, а он: ввести слои `routes/`, `handlers/`, `services/`, `clients/`, `repo/`, `domain/`, `config/`. Организовал вроде даже какую-то DI через `AppState`: `PgPool`, но это не точно и мы до конца не знаем. Пожалуйста, проверьте? Так же проблема может быть в настройке внешних URL (ключей) и таймауты - какие моменты там могут быть, какие сложности, поделитесь вашим экспертным мнением? Хендлеры у нас принимаются через `State<AppState>` и возвращаются в `Result<Json<T>, ApiError>`, так ли это? Может стоит переехать на другой стек?. Наш клиент состоит из модулей ISS, OSDR, JWST, AstronomyAPI и чтобы нас их сервисы не забанили мы установили таймауты, какими-то ретраями и обработку в юзер-агенте. Мы сторонники чистой архитектуры и поэтому вся работа с БД состоит из: работы с репозиториями в `IssRepo`, `OsdrRepo`, `CacheRepo`. Никаких SQL в хендлерах. У нас очень чёткие типы для `fetched_at`, `updated_at` или нет, нужно проверить, напишите отзыв для TIMESTAMPTZ с `DateTime<Utc>`. Кстати, расскажите чем Upsert по бизнес-ключам отличается от (вместо) слепых INSERT? Проект выполняется в фоновом режиме и наш планировщик обрабатывает все интервалы из `env`, сделали защиту от наложения (mutex/pg advisory lock), настроили rate-limit для внешних API.

Удачи вам в разработке!