SOLVERS.HUB

# Проект веб-приложения для компании “Solvers.lv”

#### Подготовил: Александр Баринов-Випулис

#### Рига, 2025

# Введение

Solvers.Hub– проект единой веб-платформы компании Solvers.lv, который позволит клиентам и сотрудникам компании возможность работать с различными продуктами и решениями, предоставляемыми компанией.

Solvers.lv – Латвийская компания, специализирующаяся на анализе, моделировании и планировании транспортных потоков. Предлагаемые компанией услуги включают:

* Анализ транспортной системы: нагрузка на систему, изменения нагрузки в течение дня, узкие места, классификация пользователей (частные авто, общественный, грузовой транспорт и т.п.)
* Анализ парковок: продолжительность парковки, классификация, изменение нагрузки на парковку по часам
* Моделирование транспортных потоков с помощью дигитальной визуализации (Aimsun)
* Решения по улучшению работы транспортной системы.

Платформа должна предоставить клиентам и сотрудникам компании унифицированный доступ к этим сервисам, проектам и сопутствующим данным.

MVP – Minimal Viable Product – предполагает создание основы веб-платформы Solvers.Hub и размещение на ней одного сервиса – Solvers.ANPR: распознавание номеров на видео.

Solvers.Hub задумывается как масштабируемая платформа: со временем планируется подключение новых модулей анализа и интеграция с внешними городскими и инженерными системами.

# **Цели проекта**

Цель проекта Solvers.hub — создать единую платформу, объединяющую инструменты компьютерного зрения для анализа транспортных видеоданных. Платформа должна быть простой в использовании, гибкой в расширении и применимой как в исследовательских, так и в прикладных задачах.

## Основные цели:

Централизация решений

Собрать в одном месте все сервисы, предоставляемые компанией Solvers.lv

Модульность

Обеспечить возможность добавления новых разделов в виде независимых модулей: распознавание номеров, подсчет потока, построение траекторий и т.д.

Доступность

Предоставить веб-интерфейс, понятный как инженерам, так и внешним пользователям (городским службам, исследователям, бизнесу).

Автоматизация

Обеспечить автоматическую обработку загруженного видео без необходимости ручной настройки и вмешательства.

Масштабируемость

Предусмотреть возможность масштабирования как по функциональности (модули), так и по объёму обрабатываемых данных.

Интеграция

Возможность будущей интеграции с внешними транспортными моделями, геоинформационными системами (QGIS, Aimsun и др.).

# Задачи проекта

Для реализации платформы Solvers.hub необходимо выполнить следующий набор технических, архитектурных и продуктовых задач. Они разбиты по направлениям и приоритетности:

## Базовая архитектура платформы (MVP)

* Разработать структуру бэкенда с поддержкой API и авторизации (например, Flask/FastAPI)
* Реализовать систему регистрации и личного кабинета пользователя
* Организовать хранилище данных: загрузка пользовательскич видео и файлов, выгрузка результатов анализа
* Разработать систему очередей задач (Celery/Redis)
* Построить модульную архитектуру для подключения отдельных сервисов (анализ, визуализация, обработка)

## Solvers.anpr (первый модуль)

* Имплементировать загрузку и обработку видео
* Интегрировать модель для детекции и распознавания автомобильных номеров
* Обеспечить высокую скорость обработки и однопроходную логику (без трекинга)
* Сохранять результаты в CSV и визуализировать аннотированное видео
* Отображать статус задачи в личном кабинете пользователя

## Frontend / UI

* Страница загрузки видео и выбора типа анализа
* Дашборд с историей задач
* Прогресс выполнения, сообщения об ошибках
* Страница с результатами и возможностью скачивания

## Безопасность и авторизация

* Реализация OAuth (Google) или почтовой авторизации
* Разграничение доступа к пользовательским данным
* Хеширование паролей, защита видеофайлов от публичного доступа

## Развитие и масштабирование

* Документация по добавлению новых модулей
* План по подключению других модулей машинного зрения: solvers.track, solvers.flow, solvers.classify
* Подключение модуля для работы с картографическими данными solvers.gis
* Интеграция с внешними транспортными моделями и системами
* Возможность API-доступа к функциям платформы

# Используемые технологии

## Среда разработки

Проект будет разрабатываться локально в среде VScode с последующим выводом репозитория в GitHub.

## Структура проекта

## Backend: Nginx, Gunicorn, Flask.

Эти три компонента необходимы нам для организации работы backend’а. Каждый компонент выполняет свою задачу в иерархии взаимодействия между клиентом и сервером. Для начала, вкратце опишем тип и назначение каждого из них:

NGINX

**Тип**: обратный прокси-сервер (reverse proxy), веб-сервер (web server), балансировщик нагрузки (load balancer)

**Назначение**: принимает внешние HTTP/HTTPS-запросы от клиента, передаёт их дальше (на Gunicorn) через Unix socket или HTTP, затем принимает результат обработки от Gunicorn и возвращает HTTP ответ клиенту

Кроме того, может кешировать, обслуживать статические файлы, шифровать трафик (SSL/TLS)

Gunicorn

**Тип:** WSGI-сервер (WSGI HTTP server), Application Server

**Назначение:** принимает HTTP-запросы, вызывает приложение через WSGI-протокол[[1]](#footnote-1), возвращает ответ

Flask

**Тип:** веб-фреймворк (web framework), Application Framework, Web Application Backend

**Назначение:** обрабатывает HTTP-запросы, маршрутизирует URL, генерирует ответы (HTML, JSON и т.д.)

## Взаимодействие компонентов backend

Полная цепочка взаимодействия всех компонентов выглядит следующим образом:

1. Браузер клиента отправляет HTTP запрос на NGINX.
2. NGINX неспособен сам обработать и выполнить логику запросов – HTTP для этого слабоват. поэтому NGINX передаёт запрос запрос на Gunicorn.
3. Gunicorn принимает HTTP запрос, “переводит” его по протоколу WSGI в пригодную для питона форму, запускает необходимое количество веб-приложений (воркеров)
4. Flask обрабатывает логику запроса и возвращает Gunicorn’у ответ по тому же WSGI протоколу
5. Gunicorn формирует из этого ответа HTTP и передает NGINXу
6. NGINX возвращает ответ клиенту

Таким образом, для корректной работы необходимы все эти компоненты – ни один из них не может быть исключён из системы:

* NGINX принимает запросы от клиента, но *не может обработать логику* в них и *не способен сам запустить Flask*.
* Gunicorn *не может эффективно справляться с функцией веб-сервера*, например, работать с большим объёмом входящих запросов, защищать соединение, поэтому мы не можем обойтись без NGINX
* Flask выполняет логику запроса, но *не может сам интерпретировать HTTP-запросы* и *не может выполнять несколько задач одновременно* без помощи Gunicorn.

*Как видим, выполнение логики и создание ответов происходит во Flask. Gunicorn и NGINX, в свою очередь, помогают Flask взаимодействовать с клиентом в сети. Во время разработки, пока сетевое взаимодействие не так важно, мы начнём с построения логики во Flask и уже затем перейдём к деплою проекта в сеть.*

# Установка компонентов backend’a: FLASK

## Структура Flask - пакета

Flask будем устанавливать как пакет. Структура его будет следующей:

1. WSGI (Web Server Gateway Interface) — это стандарт интерфейса между веб-серверами и Python-приложениями или фреймворками (такими, как Flask). Запуская питон-приложение, Gunicorn фактически выполняет питон-код, особым образом передавая в него аргументы (например, запрос представлен в виде питон-словаря). Этот стандарт также описывает, как должно ответить приложение (все на том же питоне), чтобы Gunicorn мог принять и обработать эти данные. [↑](#footnote-ref-1)