Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Северо-Восточный федеральный университет имени М.К.Аммосова»

Институт математики и информатики

Кафедра прикладной математики

Черновой вариант архитектуры проекта:

**Веб-приложение "Кто ты из айдолов"**

Выполнили: студенты 1-го курса СВФУ

гр. ИМИ М-НОД-21

Алексеев Станислав

Ву Ба Шанг

Имеев Леонид

Проверил: доцент научно-исследовательской кафедры ВТ

Григорьев Александр Виссарионович

Якутск, 2021

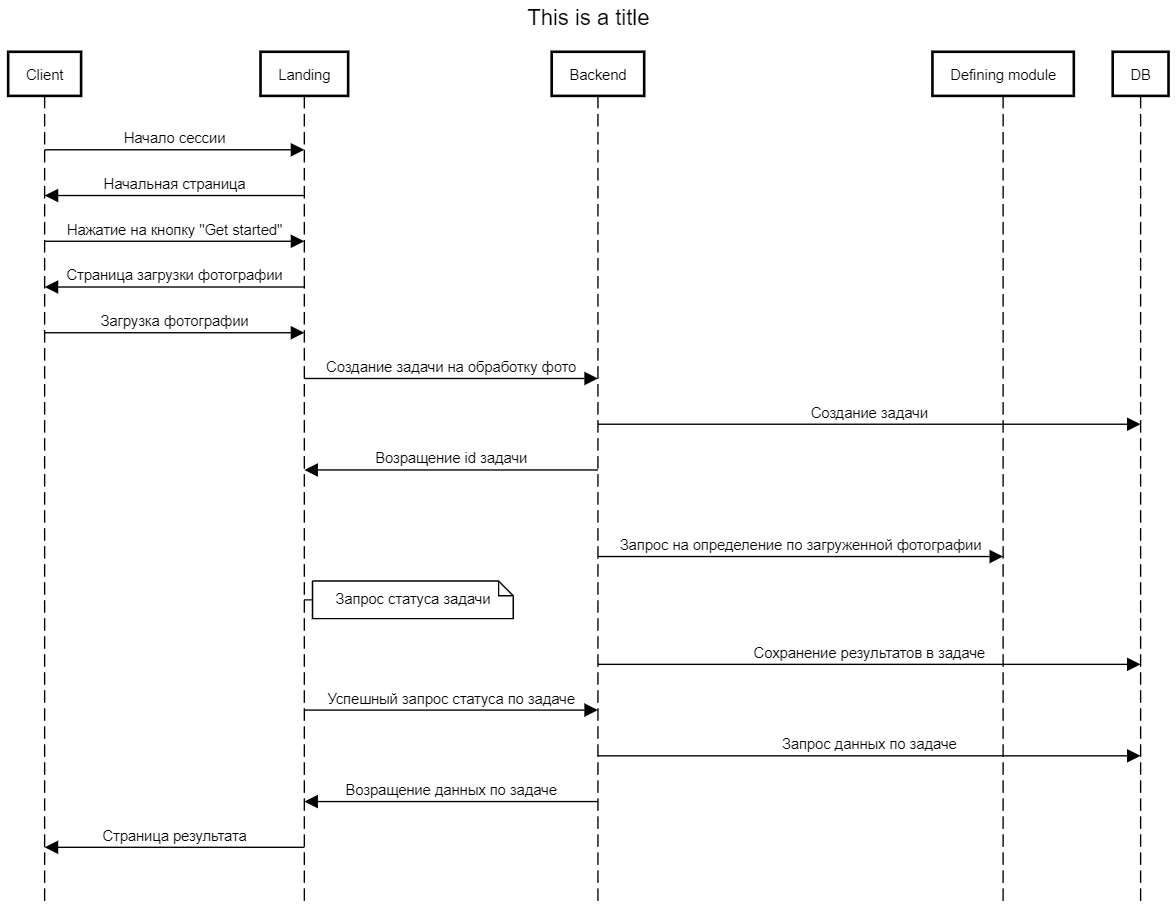
# Описание предлагаемой архитектуры web приложения

Предполагается использовать структуру из двух сервисов:  
- лендинг сайта (frontend)  
- api приложение (backend)

Взаимодействие между сервисами осуществляется с помощью REST запросов между ними.

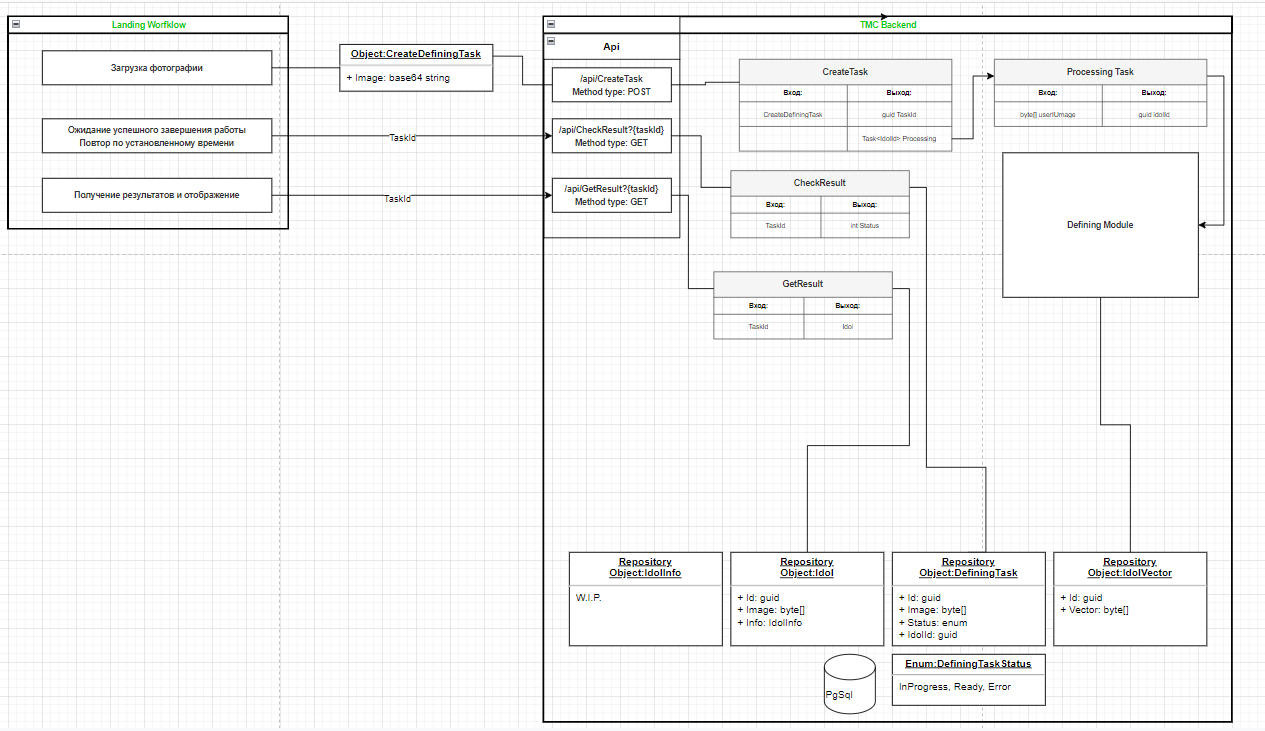
Сессии пользователей сохраняются в виде куки приложения.

# Сценарий использования приложения



1. Начало сессии: вход на сайт клиента
2. Начальная страница: вывод на экран клиента начальной страницы лендинга
3. Нажатие на кнопку “Get started”
4. Страница загрузки фотографии
5. Загрузка фотографии: всплывающее окно загрузки стандартное для ОС с которого работает пользователь
6. Создание задачи на обработку фото: создание сущности “DefiningTask” с новым id и фотографией
7. Возращение id задачи: запуск экрана загрузки на лендинге, ожидание перехода статуса задачи на “ready” или “error”
8. Запрос на определение по загруженной фотографии: запуск функции определения максимального сходства векторов загруженной фотографии и векторов для айдолов
9. Сохранение результатов в задаче: Перевод статуса задачи на “ready” или “error”, сохранение результата работы функции как id айдола
10. Успешный запрос статуса по задаче: возникает при успешной проверки статуса задачи
11. Запрос данных по задаче: определение айдола, генерация страницы резултата с информацией об айдоле и (опционально) информации о задаче (например: фотография)
12. Возращение данных по задаче
13. Страница результата

# Workflow



Объекты передающиеся между лендингом и api

* <guid> TaskId
* Object CreateDefiningTask:  
  <string> Image (base64)
* Object IdolInfo:  
  <byte[]> Image  
  <IdolInfo> IdolInfo w.i.p.

Методы API TMC Backend:

* POST /api/CreateTask   
  Создает новую задачу в базе, записывает в нее загруженную картинку и генерирует для нее id  
  Вход: <string> Image (base64)  
  Выход: <guid> TaskId
* GET /api/CheckResult?{taskId}  
  Проверяет статус задачи определения  
  Вход: <guid> TaskId  
  Выход: <int> Status
* GET /api/GetResult?{taskId}  
  При успешной проверке статуса задачи (возращение “Ready”) запрашивает результаты по проверке   
  Вход: <guid> TaskId  
  Выход: <IdolInfo> IdolInfo w.i.p.

Воркер API

* Берет задачи из базы в работу и запускает по ней модуль определения (скрипт, определяющий айдола), результат записывает как айди айдола и статус задачи  
  Вход: DefiningTask  
  Выход: Idol

Модуль определения

* Запускается воркером, сравнивает фотографию, записанную в DefiningTask с хранящимися в базе векторами айдолов, анализирует их и результат возращает воркеру  
  Вход: DefiningTask, IdolVector  
  Выход: IdolVector
* Наша задача делится на 2 основные задачи: Обнаружение лица (Face Detection) и оценка того, какое лицо айдола из БД больше всего похоже на обнаруженное лицо.

Для обучения веб-приложения необходимо собрать базу данных изображений. Все изображения помещаются в папку (your\_face) со следующей структурой:

your\_face/

├── LeeAhin

│   ├── LeeAhin\_0001.jpg

│   ├── LeeAhin\_0002.jpg

│   ├── ...

│   ├── LeeAhin\_0014.jpg

│   └── LeeAhin\_0015.jpg

├── Ahyoung

│   ├── Ahyoung \_0001.jpg

│   ├── Ahyoung \_0002.jpg

│   ├── ...

│   ├── Ahyoung \_0009.jpg

│   └── Ahyoung \_0010.jpg

├── Lisa

│   ├── Lisa \_0001.jpg

│   ├── ...

│   └── Lisa \_0006.jpg

Каждая папка содержит изображения одного айдола (чем больше изображений, тем лучше), и каждое изображение содержит только одно лицо айдола.

Лицо будет обнаружено с помощью алгоритма MTCNN. Затем, благодаря модели распознавания лиц FaceNet, лицо преобразуется в вектор лица в 512-мерном пространстве. Затем веб-приложение вычисляет скалярное произведение этого вектора лица с векторами лиц в базе данных, которые были обучены ранее. Максимально близкий результат(i) выводится в виде изображения и имени айдола под i-тым вектором.