Открытая библиотека методов объяснимого искусственного интеллекта для выявления суицидальных состояний на основе анализа содержания профилей пользователей социальных сетей «Китобой»

Руководство разработчика

Ответственный за документацию _____Буянов И. О.

СОДЕРЖАНИЕ

Введение

Данный документ представляет собой руководство для разработчиков по открытой библиотеке методов объяснимого искусственного интеллекта для выявления суицидальных состояний на основе анализа содержания профилей пользователей социальных сетей. Руководство включает себя описания настройки окружение, верхнеуровневое представление библиотеки, словарь терминов и описание классов, функций и моделей библиотеки.

Платформа является открытым программным обеспечением, код которого выложен в публичный репозиторий (https://github.com/psytechlab/kitoboy). Из-за продолжающейся разработки, необходимо проверять актуальность данного руководства по ссылке https://github.com/psytechlab/kitoboy/tree/main/docs/developer_guide.pdf.

Настройка окружения для разработки

Предвариательные требования

Для разработки понадобится вычислительная машина, работающая на операционной системе Linux или MacOS. Должны быть установлены следующие программные средства:

- Docker
- Docker-compose
- Менеджер пакетов NPM версии не ниже 3.12
- Интерпретатор Python не ниже версии 3.10
- Менеджер пакетов pip
- Программа git

Установка необходимых зависимостей

Для развертывания системы в режиме **Development** необходимо выполнить следующие действия:

1) **Скачать репозиторий.** Для этого выполните команду

2) $ git clone https://github.com/psytechlab/kitoboy/

3) **Подготовить окружение**. Для этого в корне репозитория создайте файл окружения *.env* на основе шаблона *.env.example*. Переменная *NODE_ENV* должна иметь значение *development*.

4) **Установить зависимости для Frontend**. Для этого перейдите в соответствующую директорию и установите необходимые зависимости:

5) *$ cd frontend*

6) *$ npm install*

7) **Установить зависимости для API.** Для этого перейдите соответствующую директорию и установите необходимые зависимости:

8) *$ cd ../api*

9) *$ npm install*

10) *$ cd ..*

11) **Установить зависимости для Python.** Для этого выполните команду

12) $ pip install –r zoo/requirements.txt

Сборка докер-образов

1) **Собрать докер-образы** . Для этого выполните следующую команду:

   a. *$ docker-compose build --no-cache*

2) **Запустить контейнеры в фоновом режиме**. Для этого выполните команду:

3) *$ docker-compose up --build -d*

4) **Запустить Frontend в контейнере.** Для запуска Frontend с Hot Reload выполните следующие команды:

$ docker-compose exec frontend sh

$ npm run dev

Для выхода из контейнера выполните следующие команды:

$ Ctrl + C

$ Exit

Словарь библиотеки

1) **Суицидальный статус (степень кризисности)** — категориальная градация, показывающая степень выраженности суицидального поведения. Чем выше степень, тем вероятнее, что человек совершит суицид. Определяется и устанавливается только пользователем системы (на данный момент). Может быть присвоен цифровому и реальному пользователям. Имеет следующие градации: отсутствует, низкая, средняя, высокая.

2) **Цифровой аватар (аватар)** — сущность, характеризующая наблюдаемого пользователя социальной сети. Состоит из следующего набора данных:
   a. - имени пользователя (user name),
   b. - названия социальной сети,
   c. - опционально никнейма (в Твиттере имя пользователя и никнейм могут различаться),
   d. - уровня приватности (публичный/закрытый)
   e. - статус активности (действует/удален, заблокирован)
   f. - опционально (последнее время активности)
   g. - суицидальный статус

3) **Реальная персона (персона)** — сущность, характеризующего реальную личность человека, стоящую за аватаром (если такая имеется). Одина персона может иметь несколько аватаров. Состоит из следующего набора данных:
   a. - ФИО
   b. - возраст
   c. - адрес проживания
   d. - телефон
   e. - место учебы
   f. - и т.д.
   g. - список ассоциированных цифровых аватаров,
   h. - суицидальный статус (с точки зрения БД — это полностью отдельная от суицидального статуса цифрового аватара сущность)

4) **Пост** — текст, размещенный в социальной сети от имени аватара. Состоит из следующего набора данных:
   a. - дата создания,
   b. - ассоциированный цифровой аватар,
   c. - тест сообщения,
   d. - множество атрибутов.

5) **Страница аватара** — виртуальная сущность для серии постов, размещенных одним аватаром на личной странице в социальной сети.

6) **Атрибут поста** — информационная характеристика, полученная от модели машинного обучения. Атрибут поста может быть определен или исправлен пользователем системы.

7) **Важность атрибута** — бинарный признак, показывающий распространяется ли на признак индикация в пользовательском интерфейсе. Если атрибут важный, то если будет найден хотя бы один пост с этим признаком, плашка со страницей аватара будет окрашена.

8) **Набор атрибутов модели** — множество атрибутов, которое может предсказать одна модель машинного обучение.

9) **Системный набор атрибутов** — определяется совокупностью наборов зарегистрированных в системе атрибутов.

10) **Динамика атрибута** — временное распределение проявления атрибута при фиксированной скважности для цифрового аватара.

11) **Статус обработки постов** — индикатор, показывающий статус получения атрибутов от всех подключенных моделей.

Верхнеуровневое описание библиотеки

Библиотека состоит из четырех основных компонентов:

1) Фронтэнд (frontend) — графический интерфейс, через который пользователь общается с системой.
2) Бекэнд (api) модуль, который реализует всю логику и является связующим звеном.
3) Зоопарк моделей (zoo) — модуль, который управляет моделями машинного обучения на базе Triton Inference Server. Его главная задача — собирать предсказания моделей на запросный текст.
4) База данных, где будет храниться вводимая информация пользователя.

Схема взаимодействия компонентов в формате модели C4 показана на Рисунке 1.
Основной функционал системы — автоматическая категоризация текстов по пресуицидальным и антисуицидальным сигналам. На Рисунке 2 показана диаграмма последовательносит, которая отражает взаимодейсвтие всех компонентов для реализации указанного функционала:

1) Пользователь через фронтэнд отправляет CSV-файл без заголовков, в котором первый столбец — это дата и время постинга, второй столбец — текст поста.
2) Модуль api производит парсинг CSV-файла, регистрирует данные в базе данных и получает ID добавленных текстов.
3) Модуль api отправляет пары тестов и ID в модуль зоопарка моделей.
4) Зоопарк собирает предсказания от подключенных моделей.
5) Зоопарк на основе словаря, задаваемого в конфигурационном файле, производит отображение предсказаний от моделей в пары пользовательских названий категорий и их цветовым кодом в hex формате.
6) Зоопарк отправляет в api модуль отображенные предсказания с кодом цвета, ассоциированные с ID теста.
7) Модуль api регистрирует предсказания в базе данных.

Основные пользовательские сценарии взаимодействия с системой:

1) Добавление нового пользователя социальной сети, включающее:
    a. Загрузку CSV-файла, содержащего посты и даты их публикации;
    b. Ввод логина пользователя в этой социальной сети;
    c. Ввод ссылки на данную страницу в социальной сети;
    d. Либо прикрепление этого пользователя к одному из ранее созданных в системе владельцев, включающее:
        i. Указание идентификатора владельца

    e. Либо создание нового владельца, если его ранее не добавляли в систему, и в последующее прикрепление страницы к этому владельцу, включающее:
        i. Указание фамилии и имени владельца, а также дополнительных данных, в виде возраста, связанной организации и т.д.
2) Просмотр данных о конкретной странице пользователя социальной сети:
    a. Просмотр информации и пользователе (его логин, ссылка на страницу, ссылку на информацию о владельце)
    b. Просмотр и изменение суицидального статуса страницы
    c. Просмотр текста и дат публикации постов со страницы
    d. Просмотр статуса обработки постов со страницы
    e. Просмотр, удаление и добавление атрибутов (классов), присвоенных постам со страницы
3) Просмотр данных о владельце страниц в социальных сетях:
    a. Просмотр и изменение личных данных владельца
    b. Просмотр и изменение суицидального статуса владельца
    c. Просмотр списка страниц в социальных сетях, связанных с владельцем, а также статуса обработки постов с этих страниц
4) Просмотр динамики страницы в социальной сети:
    a. Просмотр графика динамики определенных атрибутов (классов) в постах пользователя социальной сети с настройкой временного интервала и порядка агрегации, согласно интерфейсу модуля отслеживания динамики, описанному в п.4.
5) Авторизация пользователей приложения:
    a. Ввод логина и пароля
6) Просмотр списка всех добавленных страниц пользователей в социальных сетях:
    a. Просмотр статуса обработки постов со страниц
    b. Просмотр адресов страниц
    c. Возможность перейти к просмотру полной информации о каждой странице и к просмотру ее динамики

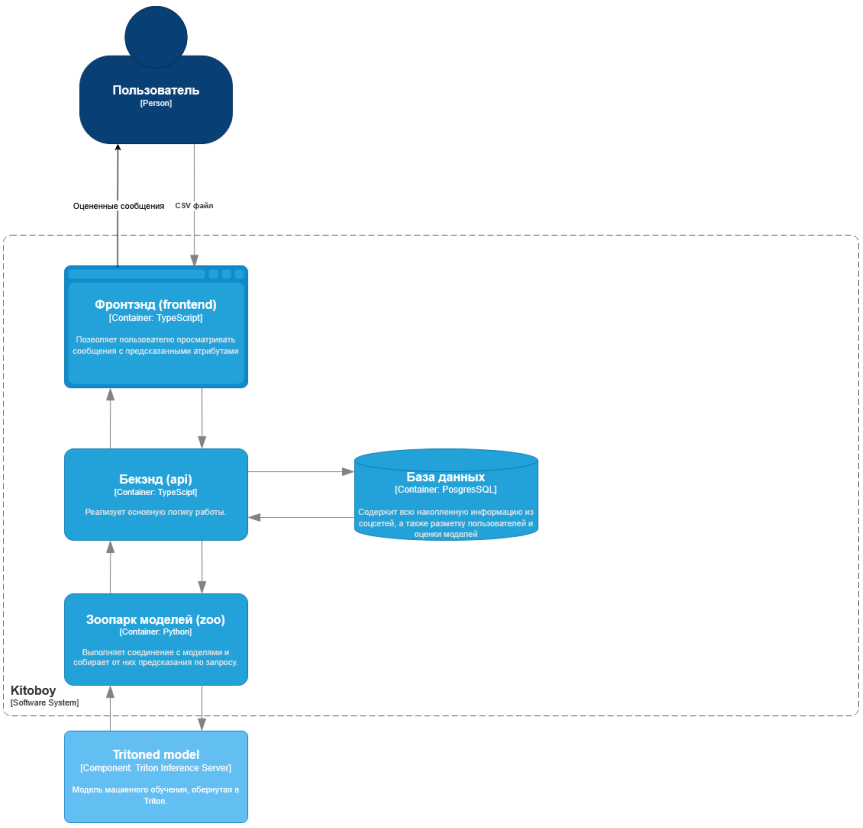

Рисунок 1 - Диаграмма взаимодействия компонентов

Рисунок 2 - Диаграмма последовательностей

Описание классов, функций и моделей

В этом разделе приведены описания функций, классов и моделей данных, важные для понимания работы библиотеки. Описания объединены по файлам или путям, где находятся файлы с описываевыми сущностями. Первым в пути файла обозначен соответствующий модулью.

api/src/handlers

addPostAttributeHandler
Создает связь между существующими в БД атрибутом и постом

- Входные параметры:
  - req: Request - запрос Express
  - res: Response<undefined | CommonError> - ответ Express

- Возвращаемые значения:
  - CommonError - при отсутствии postId/attributeId или некорректных данных
  - CommonError - при ошибке создания связи
- Возможные HTTP статусы:
  202 - успешное выполнение с данными постов
  - 400 - некорректные входные данные
  - 500 - внутренняя ошибка сервера

addPostsAttributesHandler
Создает связь между существующими в БД постами и новыми (создаваемыми впервые) или существующими атрибутами
Входные параметры:
- req: Request<AddPostsAttributesRequest> - объект запроса
- res: Response<AddPostsAttributesResponse | CommonError> - объект ответа

Выходные данные:
- Promise<void> - асинхронная функция без возвращаемого значения

HTTP коды ответов:
- 202 - успешное выполнение с данными постов
- 400 - некорректные входные данные
- 500 - внутренняя ошибка сервера

createAvatarHandler
Создает аватара с постами на основе CSV-файла и отправляет посты в сервис Zoo.
Особенности работы:
- При наличии personId привязывает аватара к существующей персоне
- При отсутствии personId создает новую персону (требуются все поля персоны)
- Создает аватара в БД и связывает с персоной
- Создает посты из CSV и связывает с аватаром
- Отправляет посты в сервис Zoo для получения предсказаний
- При ошибке от zoo данные остаются в БД

Входные параметры:
- req: Request<CreateAvatarRequest> - запрос с данными для создания аватара
- res: Response<CreateAvatarResponse | CommonError> - объект ответа

Выходные данные:
- 201 и id созданного аватара - при успешном выполнении
- 400 с текстом ошибки - при некорректных данных или ошибке от zoo

getAttributesHandler
Возвращает первые 100 атрибутов из БД

- Входные параметры:
  - req: Request - объект запроса
  - res: Response<GetAttributesResponse | CommonError> - объект ответа
- Выходные данные:
  - Promise<void> - асинхронная функция без возвращаемого значения

getAvatarHandler
Возвращает аватар с переданным Id из БД, обогащая данными из связанных таблиц

Входные параметры:

- req: Request - HTTP запрос

- res: Response<GetAvatarResponse | CommonError> - HTTP ответ

Выходные данные:

- GetAvatarResponse | CommonError - Данные аватара с дополнительной информацией или ошибка

HTTP коды ответов:

- 200 - Успешное выполнение запроса
- 400 - Некорректные входные данные
- 404 - Аватар не найден в БД

getAvatarsHandler
Возвращает аватары из БД, постранично, фильтруя по personId при его наличии в параметрах запроса. Упорядочивает аватары по дате создания по убыванию. Дополняет модель аватаров данными о персонах, связанных с аватарами, а также о статусе аватаров, их постах и атрибутах их постов.

Входные параметры:

- req: Request - HTTP запрос
- res: Response<GetAvatarsResponse | CommonError> - HTTP ответ

Выходные данные:

- GetAvatarsResponse | CommonError - Список аватаров с пагинацией или ошибка

getPersonWithAvatarsHandler
Возвращает персону с переданным id и связанные с ней аватары из БД. Упорядочивает аватары персоны по дате создания по убыванию. Дополняет модель персоны данными о статусе, аватарах, их статусах и постах с атрибутами.

Входные параметры:

- req: Request - HTTP запрос
- res: Response<GetPersonWithAvatarsResponse | CommonError> - HTTP ответ

Выходные данные:

- Promise<void>

HTTP коды ответов:

- 404 - персона с заданным id не найдена

getStatusesHandler
Возвращает все статусы из БД

Входные параметры:

- req: Request - HTTP запрос
- res: Response<GetStatusesResponse | CommonError> - HTTP ответ

Выходные данные:

- Response<GetStatusesResponse | CommonError> - Список статусов или ошибка

loginHandler
Авторизация пользователя по логину и паролю. При наличии пользователя в БД, сверяет пароль в захэшированном виде и предоставляет в ответе Auth-токен со сроком действия 12 часов. При отсутствии пользователя в БД, несоответствии пароля или некорректных данных возвращает 401

Входные параметры:

- req: Request - запрос
- res: Response<{message: string; token: string} | CommonError> - ответ

Выходные данные:

- message: string - сообщение об успешной авторизации
- token: string - авторизационный токен

HTTP коды ответов:

- 401 - Unauthorized (пользователь не найден/неверный пароль/некорректные данные)
- 200 - OK (успешная авторизация)

NotFoundError
Возвращает статус 404 и текст ошибки 'Not found'

Входные параметры:

- req: Request - HTTP запрос
- res: Response - HTTP ответ

HTTP коды ответов:

- 404 Not Found

registerHandler
Регистрация пользователя по логину и паролю

Входные параметры:

- req (Request) - HTTP запрос
- res (Response<{message: string} | CommonError>) - HTTP ответ

HTTP коды ответов:

- 201 - пользователь успешно создан
- 400 - пользователь с таким логином уже существует или некорректные данные

removePostAttributeHandler
Удаляет из БД связь между постом и атрибутом (запись из таблицы PostAttribute)

Входные параметры:

- req: Request - запрос
- res: Response<undefined | CommonError> - ответ

HTTP коды ответов:

- 200 - связь между постом и атрибутом с переданными id найдена и удалена

- 400 - не переданы id поста или атрибута
- 404 - связь между постом и атрибутом с переданными id не найдена

searchPersonHandler
Ищет персон в БД по текстовой подстроке по полям name, surname и secondName и возвращает их. Возвращает первые 20 совпадений.

Входные параметры:

- req: Request - HTTP запрос
- res: Response<SearchPersonResponse | CommonError> - HTTP ответ

HTTP коды ответов:

- 200 - Успешный поиск (даже если персона не найдена)
- 500 - Ошибка при поиске персоны

updateAvatarStatusHandler
Изменяет статус Аватара с переданным id

Входные параметры:

- req: Request - объект запроса
- res: Response<UpdateAvatarStatusResponse | CommonError> - объект ответа

HTTP коды ответов:

- 200 - успешное обновление статуса, возвращает аватар с обновленным статусом
- 400 - не переданы avatarId или statusId, или произошла ошибка

updatePersonHandler
Изменяет Персону с переданным id данными из body

Входные параметры:

- req: Request - HTTP запрос
- res: Response<UpdatePersonResponse | CommonError> - HTTP ответ

Коды ответов:

- 200 - Успешное обновление, возвращает персону с обновленными полями
- 400 - Ошибка при отсутствии personId или некорректных полях в body

updatePersonStatusHandler
Изменяет статус Персоны с переданным id

Входные параметры:

- req: Request - запрос
- res: Response<UpdatePersonStatusResponse | CommonError> - ответ

HTTP коды ответов:

- 200 - Успешное выполнение, возвращает персону с обновленным статусом
- 400 - Ошибка запроса (не переданы personId или statusId, или произошла другая ошибка)

api/src/middleware

authenticateToken
Middleware проверяет наличие авторизационного токена в заголовке 'authorization' запроса. При наличии токена верифицирует его с помощью jsonwebtoken. При успешной верификации передает исполнение дальше, при неуспешной - возвращает HTTP 401**.**

Входные параметры:

- req: Request - объект запроса
- res: Response - объект ответа
- next: NextFunction - функция передачи управления следующему обработчику

HTTP коды ответов:

- 401 - Unauthorized (неуспешная верификация токена)

api/src/shared/utils

parseAvatarCsv
Парсит CSV-файл с данными о постах Аватара. Формат: CSV. Заголовки колонок отсутствуют - данные парсятся, начиная с первой строки. Первая колонка - дата и время с тайм-зоной в формате UTC (напр. 2025-04-10T12:38:22.922Z). Вторая колонка - текст.

Входные параметры:

- filePath: string - путь к файлу

Выходные данные:

- Promise<ParsedPost[]> - Promise с массивом постов Аватара

frontend/src/pages/AvatarPage/ui/AttributeFilter/AttributeFilter.tsx

AttributeFilter
Компонент фильтра Постов Аватара по Атрибутам. Позволяет выбрать несколько Атрибутов из списка.

Входные параметры:

- attributeFilterValues: неизвестный тип - значения фильтра атрибутов
- onAttributeFilterChange: неизвестный тип - обработчик изменения фильтра атрибутов

frontend/src/pages/AvatarPage/ui/AvatarHeader/AvatarHeader.tsx

AvatarHeader
Компонент хедера страницы Аватара. Содержит данные об Аватаре и связанной с ним Персоне, а также общий список Атрибутов всех Постов

Входные параметры:

- avatarId: string - Идентификатор аватара
- username: string - Имя пользователя
- url: string - URL аватара

- personId: string - Идентификатор персоны
- fullName: string - Полное имя
- age: number - Возраст
- statusId: string - Идентификатор статуса
- attributes: Attribute[] - Список атрибутов

Выходные данные:

- ReactElement - React компонент

frontend/src/pages/AvatarPage/ui/AvatarPage/AvatarPage.tsx

AvatarPage

Компонент страницы Аватара. Содержит информацию об Аватаре и его Статусе, Постах и их Атрибутах. Позволяет просматривать и изменять Статус Аватара, просматривать общий список Атрибутов всех Постов Аватара, фильтровать Посты Аватара по выбранным Атрибутам, добавлять и удалять Атрибуты у отдельных Постов, переходить на страницу Персоны, связанной с Аватаром.

Входные параметры:

- отсутствуют

Выходные данные:

- React.ReactElement - JSX разметка компонента страницы

frontend/src/pages/AvatarPage/ui/AvatarPost/AvatarPost.tsx

AvatarPost

Компонент с данными о Посте Аватара и его Атрибутах. Позволяет добавлять и удалять Атрибуты и просматривать статус обработки Поста

Входные параметры:

- postId: string - Идентификатор поста
- text: string - Текст поста
- postedAt: Date - Дата публикации
- attributes: Array - Массив атрибутов
- onAddAttribute: function - Обработчик добавления атрибута
- onRemoveAttribute: function - Обработчик удаления атрибута

frontend/src/pages/CreateAvatarPage/model/createAvatar.model.ts

createAvatarModel

Модель данных для обработки создания Аватара и Персоны, загрузки CSV-файла, прикрепления Аватара к Персоне

- События:
  - changeTabValue
  - submitCreateAvatar
  - selectFile
  - searchPerson
- Хранилища:
  - $avatarCreated

- $selectedFile
- $sendCreateAvatarPending
- $tabValue
- $persons

frontend/src/pages/CreateAvatarPage/ui/CreateAvatarPage/CreateAvatarPage.tsx

CreateAvatarPage

Страница создания нового Аватара. Содержит форму для ввода данных об Аватаре и Персоне и для загрузки CSV-файла, а также обработку ошибок. Позволяет прикрепить Аватара либо к новой Персоне (и сразу же создать ее), либо к одной из имеющихся в системе (используя текстовый поиск)

Входные параметры:

- нет

Выходные данные:

- ReactElement - React компонент страницы создания аватара

frontend/src/pages/CreateAvatarPage/ui/CreateAvatarPage/CreateAvatarPage.tsx

CreateAvatarPage

Страница создания нового Аватара. Содержит форму для ввода данных об Аватаре и Персоне и для загрузки CSV-файла, а также обработку ошибок. Позволяет прикрепить Аватара либо к новой Персоне (и сразу же создать ее), либо к одной из имеющихся в системе (используя текстовый поиск).

Входные параметры:

- отсутствуют

Выходные данные:

- тип: JSX.Element
- описание: React-компонент страницы создания Аватара

frontend/src/pages/DynamicsPage/lib/addDataToDatasets.ts

addDataToDatasets

Добавляет в полученные датасеты данные для графика на основе информации из posts для даты из поля date. Мутирует полученный массив datasets, новый не создает.

Входные параметры:

- dateItemType: DateItem - тип элемента даты
- date: Dayjs - дата
- datasets: ChartDataset[] - массив объектов данных в формате:
  - label - название атрибута
  - data - массив числовых значений по оси Y для этого атрибута, соответствующих каждому значению на оси X
- posts: PostView[] - массив постов
- selectedAttributes: Attribute[] - массив выбранных атрибутов

Выходные данные:

- void - мутирует входной массив datasets

frontend/src/pages/DynamicsPage/lib/calculateChartData.ts

calculateChartData
Возвращает данные для графика в формате ChartData<"bar", TData, TLabel>

Входные параметры:

- endDate: Date - конечная дата
- posts: Array - массив постов
- selectedAttributes: Array - выбранные атрибуты
- selectedRevDutyCycleValue: number - выбранное значение рабочего цикла
- startDate: Date - начальная дата

Выходные данные:

- CalculatedChartData: Object
  - labels: Array<string> - массив текстовых подписей для значений на оси X (единицы времени)
  - datasets: Array<Object> - массив объектов данных
    - label: string - название атрибута
    - data: Array<number> - массив числовых значений по оси Y для этого атрибута

frontend/src/pages/DynamicsPage/lib/showMaxDateItemsCountToast.ts

showMaxDateItemsCountToast
Показывает toast о превышении максимально допустимого количества единиц времени

Входные параметры:

- dateItemType (DateItem) - тип элемента даты

Выходные параметры:

- void

frontend/src/pages/DynamicsPage/ui/DynamicsPage.tsx

DynamicsPage
Страница отслеживания динамики Атрибутов Аватара во времени. При открытии страницы использует route-параметр avatarId для получения данных об Аватаре. Получает данные об аватаре с бэкенда самостоятельно. Содержит базовую информаци об Аватаре и связанной с ним Персоне и график динамики. График позволяет выбрать необходимые Атрибуты, шаг агрегации по времени и временной интервал. При изменении настроек график автоматически перестраивается.

Входные данные:

- avatarId (route-параметр) - Идентификатор аватара

Выходные данные:

- React.FC - React функциональный компонент

frontend/src/pages/LoginPage/ui/LoginPage.tsx

LoginPage - Компонент страницы авторизации пользователя по логину и паролю

Входные данные:

- Отсутствют

Выходные данные:

- Отсутствуют

frontend/src/pages/MainPage/model/avatars.model.ts

avatarsModel

Модель для хранения данных об списке Аватаров и обработке их Постов, с учетом пагинации

- Stores:
    - $avatars
    - $allAvatarsLoaded
    - $paginationData
- Events:
    - clearLoadedAvatars
    - downloadAvatars
    - downloadAvatarsPage
    - resetPagination

frontend/src/pages/MainPage/ui/MainPage.tsx

MainPage

Компонент главной страницы. Содержит список Аватаров с краткой информацией об их Статусе и обработке их Постов, а также элементы пагинации по этому списку.

- Входные данные: отсутствуют
- Выходные данные:
    - React.ReactElement - JSX разметка компонента

frontend/src/pages/PersonPage/model/person.model.ts

personModel

Модель для хранения и изменения данных о Персоне и ее Аватарах (кроме Статуса)

- Stores:
    - $personWithAvatars
    - $personChangeSubmitted
    - $submitPersonChangePending
- Events:
    - getPersonWithAvatars
    - submitPersonChange

frontend/src/pages/PersonPage/ui/PersonInfoForm/PersonInfoForm.tsx

PersonInfoForm

Компонент формы для просмотра и редактирования данных о Персоне

Входные параметры:

- personId: number - Идентификатор персоны

frontend/src/pages/PersonPage/ui/PersonPage/PersonPage.tsx

PersonPage
Компонент страницы Персоны. Содержит информацию о Персоне и ее Статусе, а также список ее Аватаров и краткую информацию о Статусе обработки их Постов

- Входные данные:
  - Отсутствуют
- Выходные данные:
  - тип: ReactElement
  - описание: React компонент, отображающий страницу Персоны

frontend/src/pages/RootRoutePage/ui/RootRoutePage.tsx

RootRoutePage
Компонент корневой страницы для роутинга

- Входные данные:
  - Отсутствуют
- Выходные данные:
  - Отсутствуют

frontend/src/shared/api/addAttribute.ts

addAttribute
Добавляет один (существующий в системе) Атрибут отдельному Посту

Входные параметры:

- postId: string - Идентификатор поста
- attributeId: string - Идентификатор атрибута

frontend/src/shared/api/createAvatar.ts

createAvatar
Создает нового Аватара

Входные параметры:

- params: FormData - параметры для создания аватара

Выходные данные:

- Promise<void>

frontend/src/shared/api/getAttributes.ts

getAttributes
Получает данные обо всех Атрибутах в системе

Входные параметры:

- Отсутствуют

Выходные данные:

- Promise<void>

frontend/src/shared/api/getAvatarById.ts
getAvatarById
Получает данные о конкретном Аватаре, его Статусе, Постах и Атрибутах по id Аватара

Входные параметры:

- id: string - идентификатор аватара

Выходные данные:

- Promise<void>

frontend/src/shared/api/getAvatars.ts
getAvatars
Получает данные обо всех Аватарах, их Статусах, Постах и Атрибутах с учетом пагинации

Входные параметры:

- page: number - номер страницы
- size: number - размер страницы

Выходные данные:

- Promise<object> - объект с данными об аватарах, их статусах, постах и атрибутах

frontend/src/shared/api/getPersonWithAvatarsByPersonId.ts
getPersonWithAvatarsByPersonId
Получает развернутую информацию о конкретной Персоне, ее Аватарах, их Постах и Атрибутах по id Персоны

Входные параметры:

- id: string - Идентификатор персоны

Выходные параметры:

- Promise<...> - Объект с информацией о персоне, аватарах, постах и атрибутах

frontend/src/shared/api/getStatuses.ts
getStatuses
Получает данные обо всех Статусах

Выходные данные:

- Promise<void> - Промис без возвращаемого значения

frontend/src/shared/api/login.ts
login
Авторизует пользователя по логину и паролю

Входные параметры:

- username: string - логин пользователя
- password: string - пароль пользователя

frontend/src/shared/api/removeAttribute.ts
removeAttribute
Удаляет отдельный Атрибут Поста

Входные параметры:

- postId: string - идентификатор поста
- attributeId: string - идентификатор атрибута

frontend/src/shared/api/searchPerson.ts
searchPerson
Осуществляет текстовый поиск Персоны

Входные параметры:

- searchString: string - Строка для поиска

Выходные параметры:

- Promise<void> - Асинхронная функция без возвращаемого значения

frontend/src/shared/api/updateAvatarStatus.ts
updateAvatarStatus
Изменяет статус Аватара

Входные параметры:

- avatarId: string - Идентификатор аватара
- statusId: number - Идентификатор статуса

Выходные параметры:

- Promise<void> - Асинхронная операция без возвращаемого значения

frontend/src/shared/api/updatePerson.ts
updatePerson
Изменяет значения полей Персоны

Входные параметры:

- personId: string - Идентификатор персоны
- personData: Omit<Person, 'id'> - Данные персоны без поля id

frontend/src/shared/api/updatePersonStatus.ts
updatePersonStatus
Изменяет статус Персоны

Входные параметры:

- personId (unknown) - идентификатор персоны
- statusId (unknown) - идентификатор статуса

Выходные параметры:

- Promise<void>

frontend/src/shared/components/AvatarInfo/ui/AvatarInfo.tsx
AvatarInfo
Компонент, содержащий краткую информацию об Аватаре и статусе обработки его Постов, а также кнопки навигации на страницы, связанные с Аватаром

Входные параметры:

- avatar: AvatarInfoProps - объект с информацией об аватаре

frontend/src/shared/components/Header/ui/Header.tsx
Header
Компонент главного хедера приложения

Входные данные:
- Отсутствуют

Выходные данные:
- Отсутствуют

frontend/src/shared/components/Loader/ui/Loader.tsx
Loader
Компонент вращающегося лоадера

- Входные параметры:
  - className: string - CSS класс для стилизации компонента
- Выходные параметры:
  - JSX.Element - React компонент лоадера

frontend/src/shared/components/NotFound/ui/NotFound.tsx
NotFound
Компонент заглушки для ошибки 404 с кнопкой перехода на главную страницу

Входные параметры:

- отсутствуют

Выходные данные:

- React.ReactElement - React компонент с UI для страницы 404

frontend/src/shared/components/Paginator/ui/Paginator.tsx
Paginator
Компонент для навигации по страницам

Входные параметры:

- className (string) - CSS класс для стилизации
- count (number) - Общее количество элементов
- onPageChange (function) - Функция обработчик смены страницы
- page (number) - Текущая страница
- pageSize (number) - Количество элементов на странице

frontend/src/shared/components/SearchInput/ui/SearchInput.tsx

SearchInput

Компонент текстового инпута с выпадающим списком с возможностью выбора и с обработкой поиска при вводе

Входные параметры:

- label: string - текстовая метка поля
- name: string - имя поля
- onChange: function - обработчик изменения значения
- onInputChange: function - обработчик изменения ввода
- options: array - массив опций для выбора
- required: boolean - флаг обязательности поля (по умолчанию false)
- showOptionIds: boolean - флаг отображения идентификаторов опций
- size: string - размер компонента (по умолчанию 'md')
- value: any - значение поля

Выходные данные:

- JSX.Element - React компонент

frontend/src/shared/components/StatusBadge/ui/StatusBadge.tsx

StatusBadge

Компонент бэйджа Статуса Аватара или Персоны. Содержит логику загрузки Статусов и их изменения через select

Входные параметры:

- props: Props - параметры компонента

frontend/src/shared/lib/fetch/fetchApi.ts

fetchApi

Обработчик отправки запроса к сервису API. Содержит обработку ошибок авторизации и редирект на страницу логина, а также показ нотификации об ошибке

Входные параметры:

- path: string - путь запроса
- method: string - метод HTTP запроса
- body: BodyInit - тело запроса
- headers: HeadersInit - заголовки запроса
- errorOptions: ErrorOptionsByStatusCode - опции обработки ошибок по статус кодам

Выходные параметры:

- T - дженерик тип возвращаемых данных

frontend/src/shared/lib/fetch/fetchKitoboy.ts

fetchKitoboy

Обработчик отправки запроса к бэкенду. Маршрутизирует ответ между успешным и ошибочным обработчиками

Входные параметры:

- args: FetchInput[] - аргументы запроса

Выходные параметры:

- T - дженерик тип возвращаемых данных

frontend/src/shared/lib/fetch/requests.ts

processResponse
Обрабатывает успешный ответ бэкенда, распаковывает из него данные в формате, соответствующем заголовку ответа

Входные параметры:

- response: Response - HTTP ответ от бэкенда

Выходные данные:

- Promise<T | ErrorData> - Распакованные данные в формате JSON или текст

processError
Обрабатывает ответ бэкенда, содержащий ошибку

Входные параметры:

- response: Response - объект ответа
- parsedResponse: ErrorData - разобранный ответ с ошибкой
- args: FetchInput - дополнительные аргументы запроса

frontend/src/shared/lib/hooks/useAvatarAttributes.ts

useAvatarAttributes
Хук, собирающий все уникальные Атрибуты со всех Постов Аватара

Входные параметры:

- avatar: AvatarView - объект с данными аватара

Выходные данные:

- Attribute[] - массив уникальных атрибутов

frontend/src/shared/lib/hooks/useFullName.ts

useFullName
Хук, собирающий из отдельных полей Персоны строку, содержащую ФИО

Входные параметры:

- person (PersonBase | undefined) - объект с данными персоны

Выходные данные:

- string - строка с полным именем персоны

frontend/src/shared/lib/hooks/useStatusBadgeColor.ts

useStatusBadgeColor
Хук, определяющий цвет для бейджа статуса Персоны или Аватара

Входные параметры:

- statusId (UserSuicideStatus | undefined) - идентификатор статуса

Выходные данные:

- string - цветовой код для бейджа

frontend/src/shared/lib/showToast/showToastError.ts

showToastError
Показывает toast с переданным текстом или стандартным текстом ошибки и с типом "error"

Входные параметры:

- message (string) - текст сообщения об ошибке

Выходные параметры:

- void

frontend/src/shared/lib/showToast/showToastMessage.tsx

showToastMessage
Показывает toast с переданным текстом и типом (по умолчанию - "info")

Входные параметры:

- message: string - текст сообщения
- type: ToastType = 'info' - тип toast сообщения

Выходные данные:

- void

frontend/src/shared/lib/formatIntegerWithCaption.ts

formatIntegerWithCaption
Предоставляет нужное склонение существительного в зависимости от числа

Входные параметры:

- integer (number) - Само число
- caption1 (string) - Существительное в именительном падеже в единственном числе
- caption2 (string) - Для двух элементов
- caption5 (string) - Для множества элементов (*5-*0)

Выходные данные:

- string - Отформатированная строка с числом и правильным склонением

frontend/src/shared/models/auth.model.ts

authModel
Модель для работы с авторизацией

- Состояния:
  - $isAuthenticated
- Эффекты:
  - loginFx
- События:
  - login

frontend/src/shared/models/avatar.model.ts
avatarModel
Модель для хранения данных об Аватаре, его Постах и Атрибутах его Постов

- Хранилища:
  - $attributeSearchOptions
  - $avatar
  - $filteredPosts
- События:
  - addAttribute
  - filterPostsByAttributes
  - getAvatar
  - loadAttributes
  - removeAttribute

frontend/src/shared/models/status.model.ts
statusModel
Модель для хранения и изменения Статуса Аватара или Персоны

- Stores:
  - $statuses
  - $statusChanged
- Events:
  - downloadStatuses
  - setAvatarStatus
    - setPersonStatus

zoo/app/main.py
collect_predictions
Сбор предсказаний от сервисов модели и отправка их в платформу BD. Это фоновая задача, которая запускается по вызову конечной точки “/predict_on_batch

Входные параметры:

- input_texts (ServiceInput) - текст для предсказания.
- label_mapping (dict[str, tuple[str, str]]) - сопоставление имен классов, где ключ - имя класса, а значение - кортеж из имени класса и цветового кода в виде строк
- separator (str) - разделитель для мультиклассовых меток
- irrelevant_class_name (str) - название для нерелевантного класса
- url_to_send (str | None) - URL-адрес конечной точки для отправки (по умолчанию None)

connect_server
Выполнить подключение к сервисам модели Triton

Входные параметры:

- server (TritonServerAddr) - Адрес сервиса

Выходные параметры:

- int - Код статуса

HTTP коды:

- 200 - Успешное выполнение
- 400 - Ошибка выполнения

disconnect_server
Отключение сервиса модели Triton от Zoo

Входные параметры:

- model_name (str) - Имя зарегистрированной модели в Zoo

Выходные данные:

- (int) - Код статуса

HTTP коды:

- 200 - успешное выполнение
- 400 - ошибка выполнения

show_services
Показывает подключенные сервисы моделей к Zoo

Выходные данные:

- list[dict[str,str]] - Список объектов с именами моделей и соответствующими URL

update_platform_endpoint
Обновляет конечную точку платформы, куда должны отправляться результаты прогнозирования. Предназначен для конечной точки, которая принимает прогнозы и сохраняет их в БД платформы.

Входные параметры:

- new_endpoint (str) - URL новой конечной точки

Выходные параметры:

- (int) - Код статуса, 200 в случае успеха

Коды HTTP ответов:

- 200 - Успешное выполнение

predict_on_batch
Принимает тексты для предсказания и создает фоновую задачу. Фоновая задача предназначена для сбора предсказаний и отправки результата на платформу.

Входные параметры:

- data (ServiceInput) - Данные с платформы для предсказания
- background_tasks (BackgroundTasks) - Зависимость фоновой задачи

Выходные данные:

- dict[str, str] - Объект с сообщением о принятии

HTTP коды:

- 202 - Accepted

process_text
Сбор предсказаний для списка текстов. Простая функция без пост-обработки. Только собирает предсказания от зарегистрированных сервисов Triton.

Входные параметры:

- texts (TextList) - Список текстов для предсказания

Выходные параметры:

- list[list[str]] - Предсказанные классы

HTTP коды:

- 200 - Успешное выполнение

process_text_list
Собирает предсказания и форматирует их по соответствию классов. Это модификация метода predict, которая выполняет постобработку, включая сопоставление имен классов и нормализацию списков предсказаний.

Входные параметры:

- texts (TextList) - Список текстов для предсказания

Выходные данные:

- list[list[str]] - Предсказанные классы

HTTP коды ответов:

- 200 - Успешный запрос
- 422 - Ошибка валидации входных данных

zoo/app/models.py

TextList
Простая структура для текстового ввода

Входные данные:

- text_list: list[str] - список строк текста

TextToPredict
Структура объекта из платформы

Входные параметры:

- text_id (str) - идентификатор текста в БД платформы
- text (str) - текст для предсказания

ServiceInput
Ожидаемый ввод от платформы

Входные данные:

- texts (list[TextToPredict]) - список текстов для предсказания

TritonServerAddr
Кортеж с адресом сервиса модели Triton

Входные параметры:

- url: str - базовый URL
- port: str - порт сервиса
- model_name: str - имя модели в Triton

zoo/app/triton_api_client.py

TritonApiClient
Простой API клиент для сервера Triton inference. Сервер выполняет две задачи: 1. Проверка готовности модели к выводу. 2. Отправка текстов для получения предсказаний.

Входные параметры:

- base (str) - базовый URL
- port (int) - порт сервиса
- model_name (str) - название модели в сервисе Triton

Методы:

- is_ready() - Проверка готовности сервиса принимать запросы Выходные данные:
    - bool - True если готов, False в противном случае
- _make_object() - Создание внутреннего объекта по спецификации Triton Входные параметры:
    - text_list (list[str]) - тексты для предсказания Выходные данные:
    - dict - полный объект для запросов
- make_prediction_on_batch() - Запрос предсказаний от сервисов Triton для одного батча
  Входные параметры:
    - batch (list[str]) - список текстов Выходные данные:
    - list[str] - список предсказаний
- make_prediction() - Запрос предсказаний от сервисов Triton Входные параметры:
    - texts (list[str]) - список текстов произвольной длины Выходные данные:
    - list[str] - список предсказаний
- close_connection() - Корректное закрытие соединения с сервисами Triton

HTTP коды ответов:

- 200 - успешный запрос

zoo/app/utils.py

init_triton_connections
Инициализация подключений к Triton по списку адресов

- Входные параметры:
    - config (list[tuple[str, str, str]]) - список сервисов Triton с базовым url, портом и именем модели

- Выходные данные:
    - list[TritonApiClient] - список с объектами клиента Triton

make_normalized_pred_obj
Создание объекта для платформы

- Входные параметры:
    - pred (str) - значение предсказания
    - color (str) - hex-код цвета
- Выходные данные:
    - dict[str, str] - полный объект

normalize_predictions
Нормализация собранных предсказаний моделей Triton в единый формат

- Входные параметры:
    - predictions (list[list[str]]) - список предсказаний для каждого текста
    - label_mapping (dict[str, tuple[str, str]]) - маппинг для классов, значения - новое имя класса и hex-код цвета
    - separator (str) - разделитель для мультиклассовых меток
    - irrelevant_class_name (str) - название нерелевантного класса
- Выходные данные:
    - list[list[str]] - нормализованная коллекция предсказаний