

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ им.
А.Н. ТИХОНОВА

Иванов Евгений Дмитриевич, группа БПМ 214
Ермаков Семен Дмитриевич, группа БПМ 214
Гайнутдинов Расул Маратович, группа БПМ 214
Антонов Егор Алексеевич, группа БПМ 214
Цыплаков Александр Александрович, группа БПМ 214
Шишин Андрей Максимович, группа БПМ 214
Карнаушенко Михаил Алексеевич, группа БПМ 214

Документация разработчика к программной работе № 1462
«Разработка ПО для захвата движения человека и мимики лица»

Руководитель проекта:

Мотайленко И. А.

Москва 2023

Содержание

1.Общая архитектура	4
1.1. Введение	4
1.2. UML-диаграммы, логика и принципы работы	4
2. Блоки ПО	6
2.1 Клиент	6
2.1.1 UI:	11
2.1.1.1 Описание работы:	11
2.1.1.2 Импортные зависимости/библиотеки/фреймворки:	17
2.1.1.3 Архитектура блока:	20
2.1.1.4 Дизайн	38
2.1.2 Client Worker	39
2.1.2.1 Архитектура блока	39
2.1.2.2 Импортные зависимости/библиотеки/фреймворки	45
2.1.2.3 Описание работы	45
2.2. Сервер	46
2.2.1 Server	47
2.2.1.1. Архитектура блока	47
2.2.1.2. Импортные зависимости/библиотеки/фреймворки	48
2.2.1.3. Описание работы	48
2.2.2 Server Worker	48
2.2.2.1. Архитектура блока	49
2.2.2.2. Импортные зависимости/библиотеки/фреймворки	52
2.2.2.3. Описание работы	52
2.2.3. Orchestrator	52
2.2.3.1. Архитектура блока	53
2.2.3.2. Импортные зависимости/библиотеки/фреймворки	54
2.2.3.3. Описание работы	54
2.2.4 DB	54
2.2.4.1. Архитектура блока	55
2.2.4.2. Импортные зависимости/библиотеки/фреймворки	57
2.2.4.3. Описание работы	57
2.3 Вычислительный блок	58
Этапы работы EasyMocar:	60
2.3.1 A_Machine	61
2.3.1.1. Архитектура блока	62
2.3.1.2. Импортные зависимости/библиотеки/фреймворки	62
2.3.1.3. Описание работы	62
2.3.2 Pipeline	62
2.3.2.1. Архитектура блока	63
2.3.2.2. Импортные зависимости/библиотеки/фреймворки	65

2.3.2.3. Описание работы	65
2.3.3 FaceHunter	65
2.3.3.1. Архитектура блока	66
2.3.3.2. Импортируемые зависимости/библиотеки/фреймворки	68
2.3.3.3. Описание работы	69
3. Приложение к Документации Разработчика	69
3.1 Общая UML-диаграмма, отражающая связь классов между собой	69

1.Общая архитектура

Разработанное ПО рассчитано на использование конечным пользователем, как в автономном(оффлайн) режиме, так и в вариации клиент-серверного решения, когда вычислительные процессы отдаются на выполнение на стороннюю вычислительную машину, являющуюся частью располагаемых проектом ресурсов.

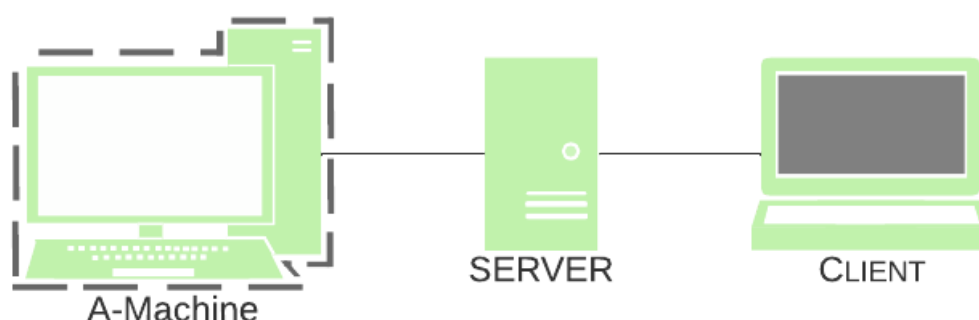
1.1. Введение

Разработанное ПО для захвата движения и мимики лица можно поделить на 3 основных автономных блока, работающих параллельно. Основные блоки отражены в пункте 2 данного документа. Данный проект может быть как целиком имплементирован в уже работающую систему (например, в локальную лабораторию), так и быть развернут независимо(например, для коммерческих целей).

1.2. UML-диаграммы, логика и принципы работы

В данном пункте отражена основная функциональная и нефункциональная логика. Общий принцип работы заключается в следующем:

Пользователь на своем устройстве, на котором установлен клиентский блок ПО, запускает программу клиента (см. Документация Пользователя). В этом блоке пользователь подключает необходимые ему для записи видео встроенные или сторонние (ip) камеры, выбирает конфигурацию камер, включает запись и управляет ей. В другом окне пользователь выбирает параметры обработки видео, в том числе способы запуска - оффлайн или онлайн. После этого запускается набор скриптов для выполнения запроса пользователя.



В оффлайн режиме блок клиента проверяет необходимые зависимости для запуска обработки видео, и в случае успеха выполняет запрос

В онлайн режиме запрос клиента отправляется на распределительный сервер, откуда забирается вычислительной машиной для выполнения. После выполнения запроса вычислительная машина отправляет результаты запроса обратно на распределительный сервер, после чего данные отправляются обратно клиенту.

Все внутренние процессы клиентского блока сокрыты от пользователя, а управление происходит через интерфейс клиента.

Полная UML-диаграмма ПО (см. Приложение к Документации Разработчика) описывает поведение классов внутри каждого блока и их взаимодействие.

На вычислительной машине или на устройстве клиента проводится обработка видео материалов, предоставленных Пользователем. При помощи инструментов ИИ на видео наносится сетка точек, а также формируются метаданные по типу ребер и весов точек. В зависимости от параметров запроса Пользователя, вычислительный или клиентский блок формирует 3д модель и/или анимацию, созданную при помощи ИИ.

2. Блоки ПО

В данном пункте отражена документация по каждому блоку ПО и его сабблокам.

2.1 Клиент

Пользователь имеет возможность работать с инструментами интерфейса в двух окнах: В первом окне происходит вся работа с камерами - за это отвечает класс CameraScreenApp.

Функции класса CameraScreenApp используют в себе методы класса Client_Worker

Во втором окне происходит сбор данных, нужных для дальнейшей обработки снятых видео- за это отвечает класс MocapScreenApp.

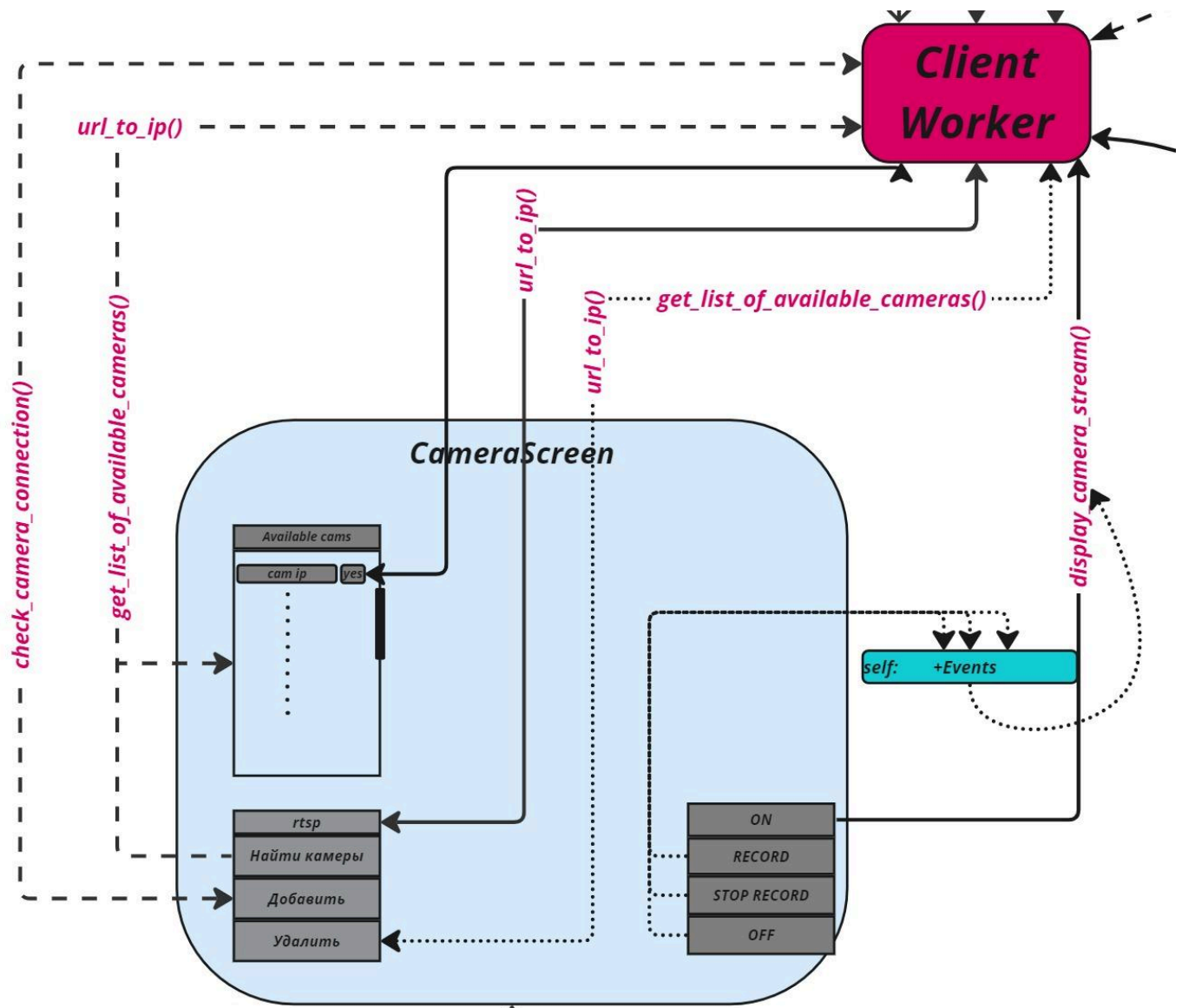
Функции класса MocapScreenApp нацелены на сборку частей итогового json-а, который отправляется в Client_Worker в котором вызывается функция submit по нажатию на соответствующую кнопку интерфейса окна.

JSON состоит из:

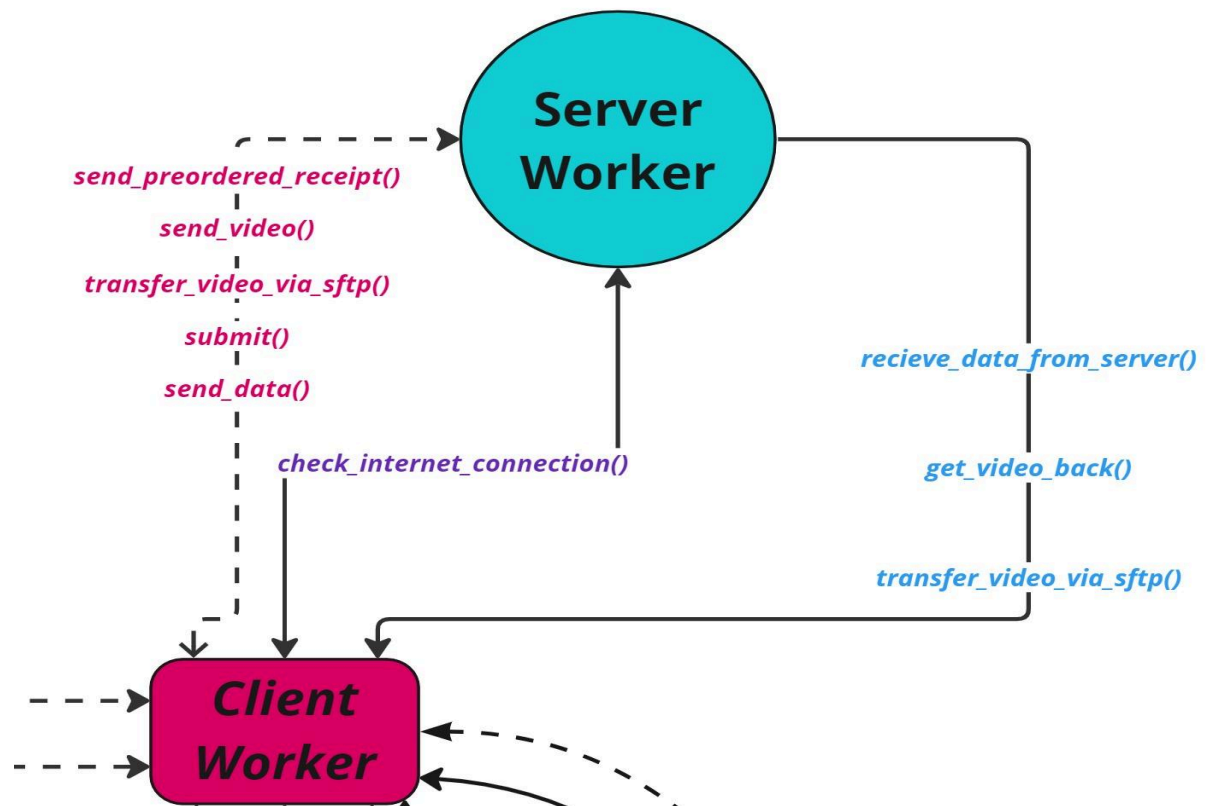
- списка отправляемых на обработку видеозаписей
- режима сборки online\offline
- режима сборки Face_hunting
- итогового скрипта для запуска сборки

Все действия клиента происходят только через интерфейс, все сетевые действия логируются.

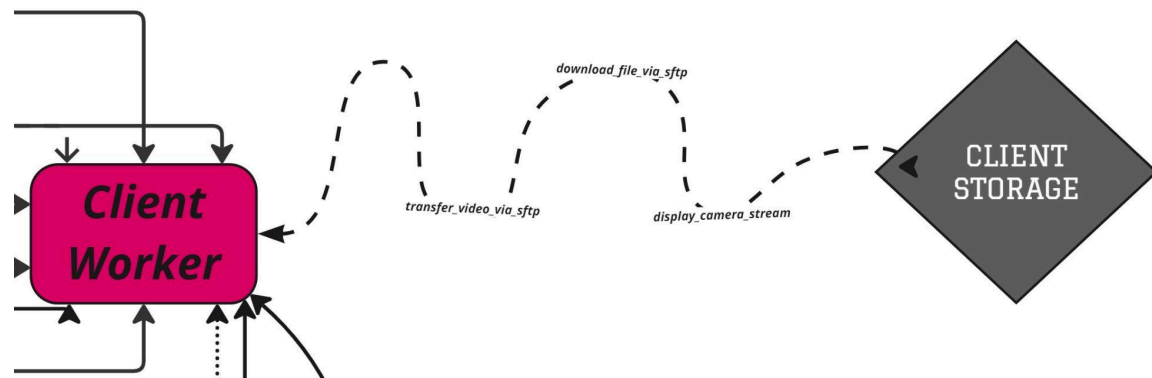
Связь CameraScreen и Client_Worker:



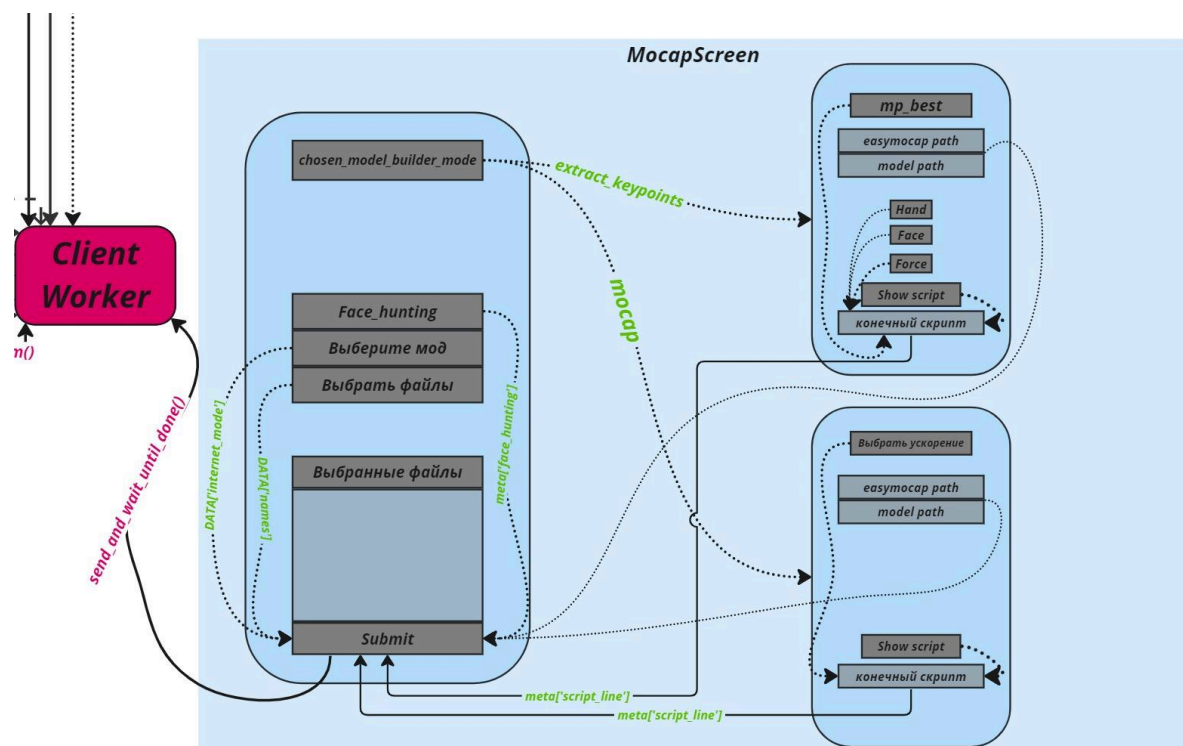
Связь Client_Worker и Server_Worker:



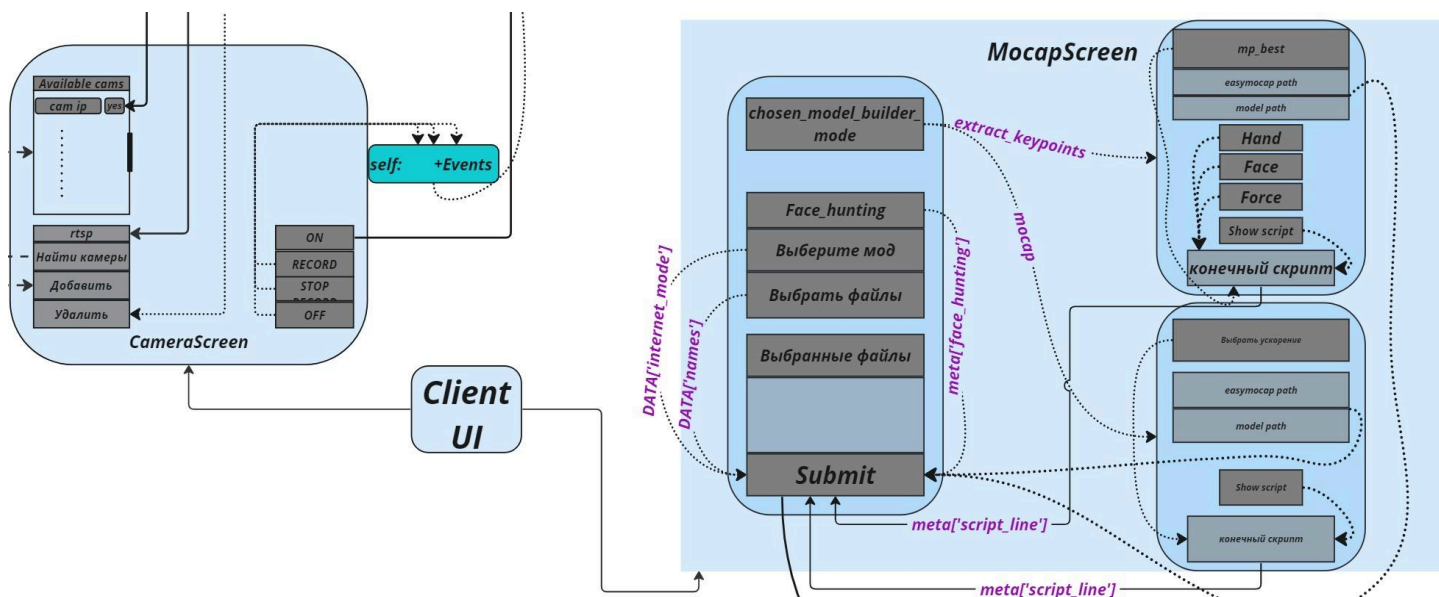
Связь Client_Worker и Client_Storage:



Связь MocapScreen и Client_Worker:



Связь MocapScreen и CameraScreen:



2.1.1 UI:

Визуал программы делится на два основных скрина - CameraScreen и MocapScreen, их работу отображают классы CameraScreenApp и MocapScreenApp

2.1.1.1 Описание работы:

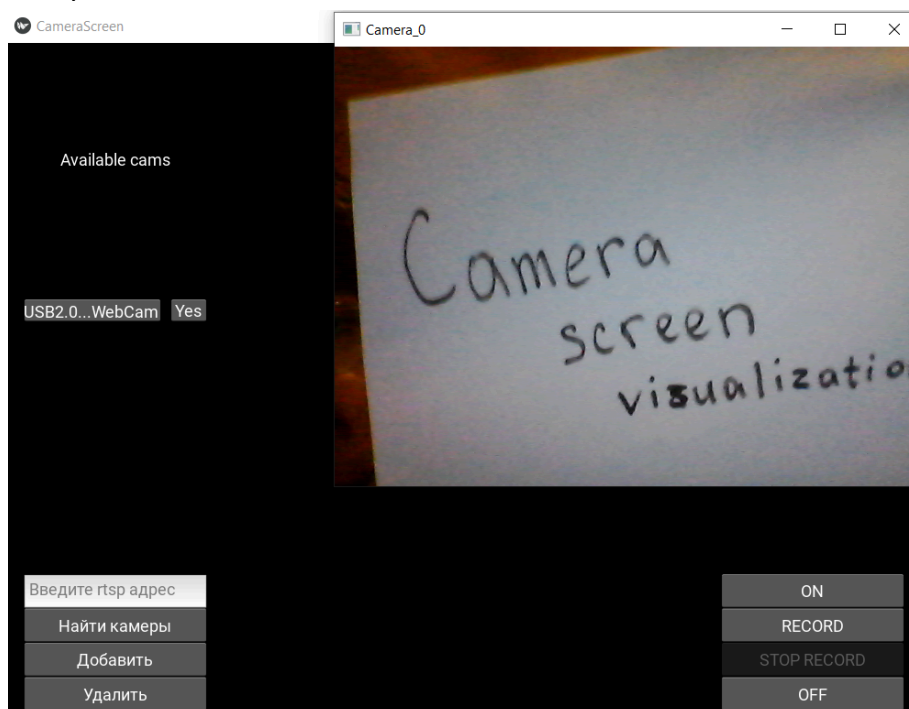
Описание работы окна CameraScreen:

Объект класса CameraScreenApp запускается в работу с помощью команды `run()`, после чего срабатывает метод `build()` в котором создается весь визуал окна. В этом окне происходит вся работа с камерами - добавление, удаление включение запись и сохранение, после чего записанные файлы будут использованы во втором окне MocapScreenApp

В левом нижнем углу находится блок кнопок “Найти камеры”, “Добавить” и “Удалить”, которые обновляют прокручиваемое поле “Available cams”:
(При добавлении камер вызываются функции класса `Client_Worker` проверяющие введенное название камеры на наличие корректного ip и корректного порта : `Client_worker.url_to_ip_port`)



Кнопки “ON”, “RECORD”, “STOP RECORD” и “OFF” обеспечивают работу с записью камер:



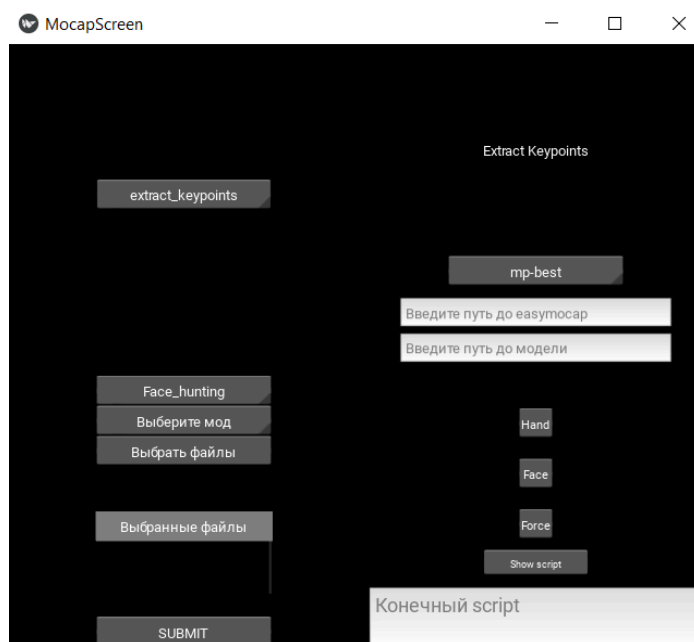
Описание работы окна MosapScreen:

Объект класса MosapScreenApp запускается в работу с помощью команды `run()`, после чего срабатывает метод `build()` в котором создается весь визуал окна.

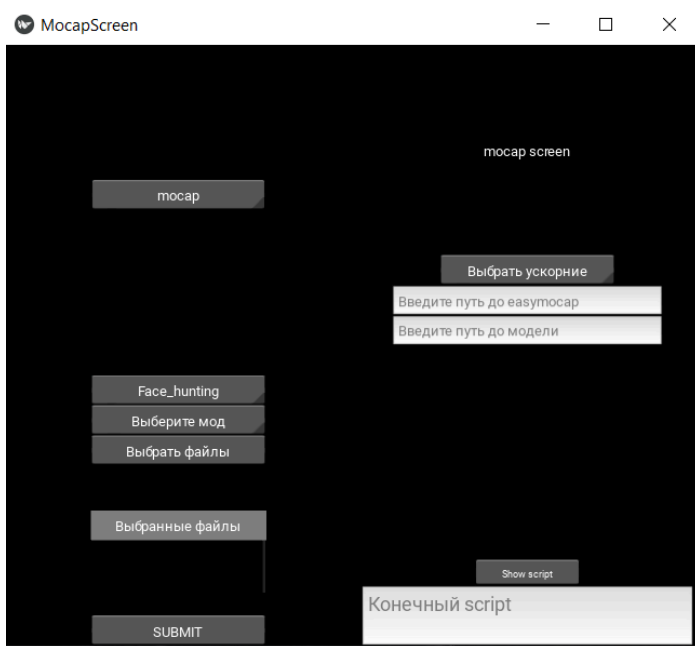
Основная задача функционала этого окна собрать json, содержащий поля с выбранными файлами, информацией о выбранных режимах сборки и основной скрипт для запуска последующей работы обработки снятых видео

Окно разделено на две половины, причем за левую половину отвечает статичный VBoxLayout, правая же половина меняется в зависимости от выбора варианта сборки скрипта:

Режим сборки скрипта extract_keypoints:



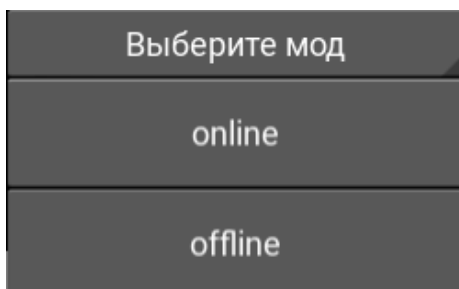
Режим сборки скрипта mocap:



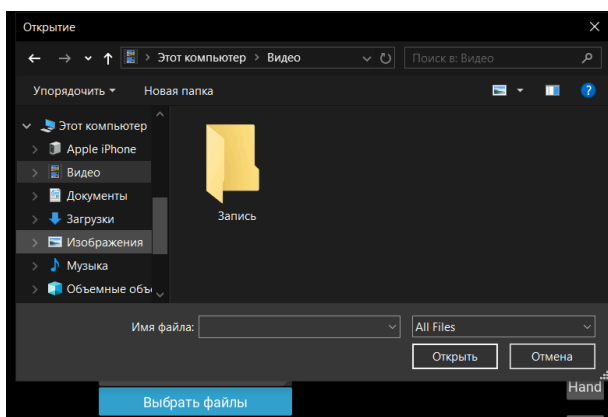
Всплывающий список “Face_hunting” обеспечивает добавление модели обработки лица:



Всплывающий список “Выберите мод” обеспечивает выбор интернет мода запуска online\offline:



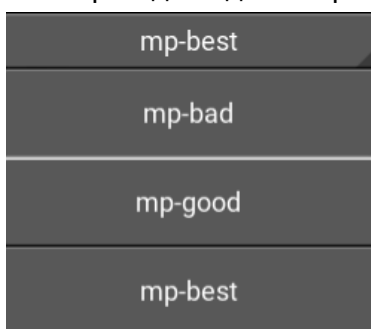
Кнопка “выберите файлы” обеспечивают выбор файлов для обработки:



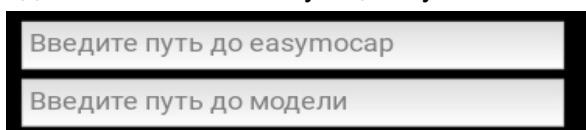
Всплывающий список “Выбрать ускорение” обеспечивает выбор значений для флага `-fps` в собираемом тосар скрипте:



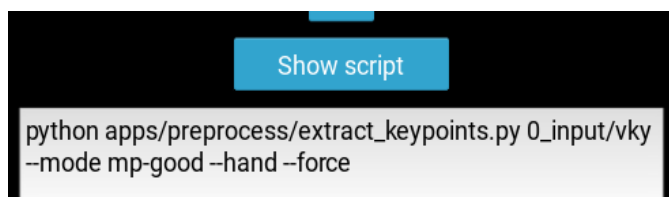
Всплывающий список “mp-best” обеспечивает выбор модели для сборки скрипта `extract_keypoints`:



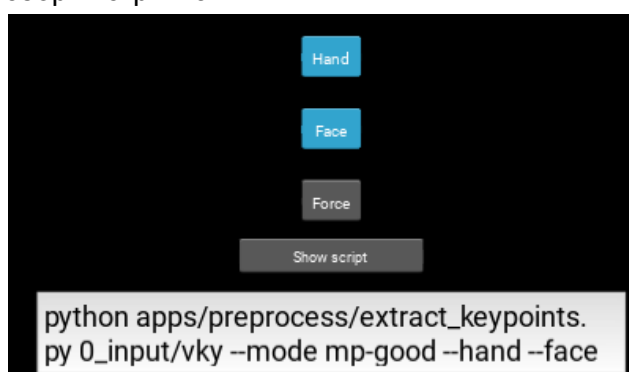
Текстовые поля “Введите путь до easумосар” и “Введите путь до модели” позволяют добавить соответствующие пути:



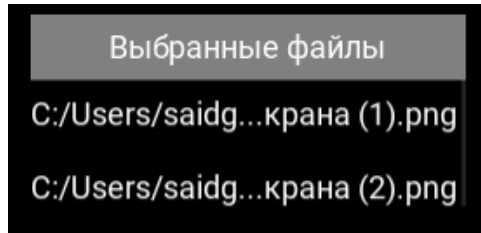
Кнопка “Show script” отображает текущий собранный скрипт в текстовом поле “Конечный скрипт”



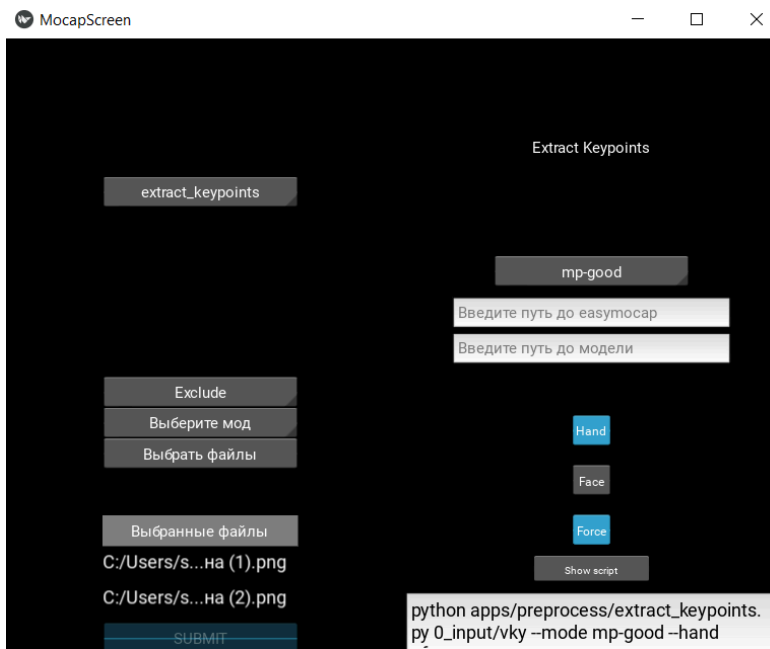
Кнопки “Face”, “Force” и “Hand” позволяют добавить соответствующие флаги для сборки скрипта:



Поле “Выбранные файлы” отображает директории всех выбранных файлов:



Кнопка submit вызывает функцию из класса `Client_Worker` в которой достается итоговый скрипт и собирается json с параметрами для запуска отправленных файлов:



2.1.1.2 Импортруемые зависимости/библиотеки/фреймворки:

Импортруемые зависимости/библиотеки/фреймворки для работы над CameraScreenApp:

- **threading** - Модуль Thread, эмулирующий подмножество модели потоков Java, используется для работы с потоками
- **cv2** - добавляет возможность подключения к камерам
- **kivy.app** - подключает класс App, который является основой для создания приложений Kivy, в нашем случае является основной точкой входа в цикл выполнения Kivy
- **kivy.uix.boxlayout** - подключает BoxLayout, который размещает дочерние элементы в вертикальном или горизонтальном блоке.
используется чтобы расположить виджеты друг относительно друга
- **kivy.uix.button** - подключает возможность использовать кнопку, которая представляет собой ~kivy.uix.label.Label со связанными действиями, которые запускаются при нажатии кнопки (или отпускании после щелчка/прикосновения).
- **kivy.uix.anchorlayout** - подключает возможность использования AnchorLayout, который выравнивает своих дочерних элементов по границе (сверху, снизу, слева, справа) или по центру.
- **kivy.uix.textinput** - подключает возможность использования виджета TextInput, который предоставляет поле для редактируемого обычного текста.

- **kivy.ui.label** - подключает возможность использования Label, который используется для рендеринга текста
- **kivy.ui.scrollview** - подключает возможность использования виджета ScrollView, который предоставляет прокручиваемую/панорамируемую область просмотра, которая обрезается по ограничивающей рамке прокрутки.
- **kivy.ui.gridlayout** - подключает возможность использования GridLayout, который упорядочивает дочерние элементы в матрице. Он берет доступное пространство и делит его на столбцы и строки, а затем добавляет в полученные «ячейки» виджеты.
- **kivy.core.window** - подключает возможность использования базового класса для создания окна Kivy по умолчанию, поддерживает только одно окно для каждого приложения
- **client_worker** - подключает возможность использования атрибутов класса Client_Worker()

Импортируемые зависимости/библиотеки/фреймворки для работы над MosapScreenApp:

- **json** - подключает возможность использования json, который используется в качестве облегченного формата обмена данными.
- **subprocess** - подключает возможность использования модуля, который позволяет запускать процессы, подключаться к их каналам ввода/вывода/ошибок и получать их коды возврата.
- **threading** - Модуль Thread, эмулирующий подмножество модели потоков Java, используется для работы с потоками
- **time** - подключает возможность пользования модулем, который предоставляет различные функции для управления значениями времени.
- **kivy.app** - подключает класс App, который является основой для создания приложений Kivy, в нашем случае является основной точкой входа в цикл выполнения Kivy
- **kivy.ui.boxlayout** - подключает BoxLayout, который размещает дочерние элементы в вертикальном или горизонтальном блоке.
используется чтобы расположить виджеты друг относительно друга
- **kivy.ui.spinner** - подключает возможность использования Spinner - виджета, который позволяет быстро выбрать одно значение из набора.
- **kivy.ui.button** - подключает возможность использовать кнопку, которая представляет собой ~kivy.ui.label.Label со связанными действиями, которые запускаются при нажатии кнопки (или отпуске после щелчка/прикосновения).
- **kivy.ui.anchorlayout** - подключает возможность использования AnchorLayout, который выравнивает своих дочерних элементов по границе (сверху, снизу, слева, справа) или по центру.
- **kivy.ui.textinput** - подключает возможность использования виджета TextInput, который предоставляет поле для редактируемого обычного текста.
- **kivy.ui.label** - подключает возможность использования Label, который используется для рендеринга текста
- **kivy.ui.scrollview** - подключает возможность использования виджета ScrollView, который предоставляет прокручиваемую/панорамируемую область просмотра, которая обрезается по ограничивающей рамке прокрутки.
- **kivy.core.window** - подключает возможность использования базового класса для создания окна Kivy по умолчанию, поддерживает только одно окно для каждого приложения
- **kivy.ui.togglebutton** - подключает возможность использования виджетом ToggleButton, который действует как флажок

- **PyQt5.QtWidgets** - подключает возможность использования виджетов библиотеки Qt
- **kivy.uix.gridlayout** - подключает возможность использования GridLayout, который упорядочивает дочерние элементы в матрице. Он берет доступное пространство и делит его на столбцы и строки, а затем добавляет в полученные «ячейки» виджеты.
- **sys** - подключает возможность использования модуля, который обеспечивает доступ к некоторым объектам, используемым или поддерживаемым интерпретатором, а также к функциям, которые тесно взаимодействуют с интерпретатором.
- **kivy.clock** - подключает возможность использования объекта Clock, который позволяет запланировать вызов функции в будущем - один или несколько раз через определенные промежутки времени.
- **kivy.graphics** - подключает возможность использования пакета, который объединяет множество функций низкого уровня, используемых для рисования.

2.1.1.3 Архитектура блока:

Архитектура приложения для окна *CameraScreen*:



Поля класса *CameraScreenApp*:

MAX_NUMBER_OF_CAMERAS

Тип: *int*

Описание: Максимальное допустимое число внутренних камер

MAX_NUMBER_OF_IP_CAMERAS

Тип: *int*

Описание: Максимальное допустимое число ip камер

FOURCC

Тип: *cv2.VideoWriter_fourcc*

Описание: Кодек для записи камер в формате mp4

FRAMES_PER_SECOND

Тип: *float*

Описание: Количество кадров в секунду

RESOLUTION

Тип: *tuple[int, int]*

Описание: Разрешение видео

DEMONS

Тип: *Any[]*

Описание: список демонов

CAM_DICT

Тип: *{}*

Описание: словарь из внутренних камер

CAM_IP_DICT

Тип: *{}*

Описание: словарь из ip камер

CAM_DICT_ON_AIR**Тип:** *{}***Описание:** словарь из внутренних пишущих камер**CAM_IP_DICT_ON_AIR****Тип:** *{}***Описание:** словарь из пишущих ip камер**NUMBER_OF_AVAILABLE_CAMERAS****Тип:** *int***Описание:** Количество доступных для записи внутренних камер**NUMBER_OF_AVAILABLE_IP_CAMERAS****Тип:** *int***Описание:** Количество доступных для записи ip камер**VIDEO_WRITERS****Тип:** *{}***Описание:** словарь из cv2.VideoWriter для внутренних камер**VIDEO_WRITERS_IP****Тип:** *{}***Описание:** словарь из cv2.VideoWriter для ip камер**events_to_turn_on****Тип:** *threading.Event[]***Описание:** список объектов *threading.Event()* для включения внутренних камер**events_to_start_recording****Тип:** *threading.Event[]***Описание:** список объектов *threading.Event()* для старта записи внутренних камер

events_to_stop_recording

Тип: *threading.Event[]*

Описание: список объектов *threading.Event()* для остановки записи внутренних камер

events_to_turn_off

Тип: *threading.Event[]*

Описание: список объектов *threading.Event()* для выключения внутренних камер

events_to_turn_on_ip

Тип: *{}*

Описание: словарь объектов *threading.Event()* для включения внутренних камер

events_to_start_recording_ip

Тип: *{}*

Описание: словарь объектов *threading.Event()* для старта записи внутренних камер

events_to_stop_recording_ip

Тип: *{}*

Описание: словарь объектов *threading.Event()* для остановки записи внутренних камер

events_to_turn_off_ip

Тип: *{}*

Описание: словарь объектов *threading.Event()* для выключения внутренних камер

layout

Тип: *kivy.uix.boxlayout*

Описание: основной контейнер для размещения элементов первого окна

scroll_view

Тип: *kivy.uix.scrollview*

Описание: прокручиваемый контейнер для размещения камер

inner_layout

Тип: *kivy.uix.boxlayout*

Описание: контейнер для хранения GridLayout - строчек для камер состоящих из двух кнопок

available_cams

Тип: *string[]*

Описание: список из всех камер

button_container

Тип: *kivy.uix.boxlayout*

Описание: контейнер для размещения кнопок для работы с записью камер

button1

Тип: *kivy.uix.button*

Описание: кнопка для включения камер

button2

Тип: *kivy.uix.button*

Описание: кнопка для старта записи включенных камер камер

button3

Тип: *kivy.uix.button*

Описание: кнопка для остановки записи включенных камер

button4

Тип: *kivy.uix.button*

Описание: кнопка для выключения камер

text_input_container

Тип: *kivy.uix.boxlayout*

Описание: контейнер для размещения кнопок для работы с добавлением камер

text_input_1

Тип: *kivy.uix.textinput*

Описание: текстовое поле для ввода добавляемой камеры

find_cams_button

Тип: *kivy.uix.button*

Описание: кнопка для добавления всех внутренних камер

confirm_button

Тип: *kivy.uix.button*

Описание: кнопка для добавления камеры

confirm_to_delete

Тип: *kivy.uix.button*

Описание: кнопка для удаления камеры

label

Тип: *kivy.uix.label*

Описание: Лейбл с текстом "Available cams"

combined_box

Тип: *kivy.uix.boxlayout*

Описание: контейнер для размещения элементов в левой половине первого окна

anchor_layout2

Тип: *kivy.uix.anchorlayout*

Описание: контейнер для упорядоченного размещения элементов в правой половине первого окна

Методы класса *CameraScreenApp*:

build() → `kivy.uix.boxlayout.BoxLayout`

Описание:

Метод вызывается при вызове функции `run()`

Внутри метода создаются все сущности класса с начальными параметрами а также встраивается их взаимосвязь

Возвращаемое значение:

- **`kivy.uix.boxlayout.BoxLayout`** - основной контейнер окна приложения

on_button1_click(*instance*) → `None`

Описание:

Метод вызывается при нажатии на кнопку `button1`

Внутри метода происходит добавление камер в словарь камер готовых к записи, заводится объект класса `Client_Worker`, а также создается новый поток и демон и устанавливаются события приуроченные к включению камер

Параметры:

- **`instance: Any`** - Kivy виджет, в данном случае Kivy Button

on_button2_click(*instance*) → `None`

Описание:

Метод вызывается при нажатии на кнопку `button2`

Внутри метода происходит установка событий приуроченных к старту записи камер

Параметры:

- **`instance: Any`** - Kivy виджет, в данном случае Kivy Button

on_button3_click(*instance*) → `None`

Описание:

Метод вызывается при нажатии на кнопку `button3`

Внутри метода происходит установка событий приуроченных к остановке записи камер

Параметры:

- **`instance: Any`** - Kivy виджет, в данном случае Kivy Button

on_button4_click(instance) → None

Описание:

Метод вызывается при нажатии на кнопку button4

Внутри метода происходит установка событий приуроченных к выключению камер, а также удаление камер из словаря камер

Параметры:

- **instance: Any** - Kivy виджет, в данном случае Kivy Button

on_find_cams_click() → None

Описание:

Метод вызывается при нажатии на кнопку find_cams_button

Внутри метода происходит добавление всех внутренних камер в словарь камер готовых к записи и добавление камер в визуал приложения, заводится объект класса Client_Worker

on_confirm_click(text_input_1) → None

Описание:

Метод вызывается при нажатии на кнопку confirm_button

Внутри метода происходит проверка добавляемой камеры на то, внутренняя она или же ip - камера, далее она добавляется в соответствующий словарь камер, а также в визуал приложения

Параметры:

- **text_input_1: string** - строка с названием камеры

on_confirm_to_delete_click(text_input_2) → None

Описание:

Метод вызывается при нажатии на кнопку confirm_to_delete

Внутри метода происходит проверка добавляемой камеры на то, внутренняя она или же ip - камера, далее она удаляется из соответствующего словаря камер, а также из визуала приложения

Параметры:

- **text_input_2: string** - строка с названием камеры

toggle_button_text(instance) → None

Описание:

Метод вызывается при нажатии на вторую кнопку из строки с визуалом камеры в окне приложения

Внутри метода происходит проверка добавляемой камеры на то, внутренняя она или же ip - камера, далее она добавляется в соответствующий словарь камер, или же удаляется из соответствующего словаря камер в зависимости от того какой флаг содержался в ее тексте

Параметры:

- **instance: Any** - Kivy виджет, в данном случае Kivy Button

button_exists_with_text(text) → None

Описание:

Внутри метода происходит проверка на то, существует ли камера с таким названием в визуале, для отсутствия возникновения коллизий

Параметры:

- **text: string** - строка с названием камеры

Возвращаемое значение:

- **boolean**

Архитектура приложения для окна MocapScreen:

архитектура полей:

MocapScreenApp

```
+ main_layout : kivy.uix.boxlayout.BoxLayout
+ chosen_internet_mode : string = "online"
+ chosen_model_builder_mode : string
+   + chosen_fps : string
+ chosen_keypoints_model : string
+   + chosen_parts : string[]
+   + selected_files : string[]
+ is_right_keypoints_layout : boolean = True
+   + chosen_face_hunting : string
+ left_layout : kivy.uix.boxlayout.BoxLayout
+ bottom_container : kivy.uix.boxlayout.BoxLayout
+   + button_transfer_test : kivy.uix.button
+   + mode_layout : kivy.uix.spinner.Spinner
+   + selected_files_button : kivy.uix.button
+ label_and_scrollview_layout : kivy.uix.boxlayout.BoxLayout
+   + label : kivy.uix.label.Label
+ selected_files_scrollview : kivy.uix.scrollview.ScrollView
+   + selected_files_layout : kivy.uix.gridlayout.GridLayout
+   + build_spinner : kivy.uix.spinner.Spinner
+   + spinner_upper : kivy.uix.spinner.Spinner
+ upper_layout : kivy.uix.anchorlayout.AnchorLayout
+   + right_layout : kivy.uix.boxlayout.BoxLayout
+ right_keypoints_layout : kivy.uix.boxlayout.BoxLayout
+   + keypoints_box : kivy.uix.boxlayout.BoxLayout
+   + bottom_box : kivy.uix.boxlayout.BoxLayout
+   + keypoints_label : kivy.uix.label.Label
+   + keypoints_spinner : kivy.uix.spinner.Spinner
+ path_easymocap_input_1 : kivy.uix.textinput.TextInput
+ path_model_input_1 : kivy.uix.textinput.TextInput
+ check_layout : kivy.uix.boxlayout.BoxLayout
+   + toggle_buttons : kivy.uix.togglebutton[]
+   + show_script_button1 : kivy.uix.button
+   + script_input1 : kivy.uix.textinput.TextInput
+ mocap_box : kivy.uix.boxlayout.BoxLayout
+   + mocap_label : kivy.uix.label.Label
+   + mocap_spinner : kivy.uix.spinner.Spinner
+ path_easymocap_input_2 : kivy.uix.textinput.TextInput
+ path_model_input_2 : kivy.uix.textinput.TextInput
+ mocap_bottom_box : kivy.uix.boxlayout.BoxLayout
+   + show_script_button2 : kivy.uix.button
+   + script_input2 : kivy.uix.textinput.TextInput
```

архитектура методов:

```
+ build(self) : kivy.uix.boxlayout.BoxLayout
+ on_window_resize(self, window : kivy.core.window., width : int, height : int) : None
+ on_show_script_button1_press(self, instance : Any) : None
+ on_show_script_button2_press(self, instance : Any) : None
+ get_script(self) : string !!!!
+ send_and_wait_until_done(self) : None
+ update_bg_rect(self, instance : Any, value : Any) : None
+ test_transfer(self, instance : Any) : None
+ on_mode_layout_select(self, spinner : kivy.uix.spinner.Spinner, text : string) : None
+ on_build_spinner_select(self, spinner : kivy.uix.spinner.Spinner, text : string) : None
+ on_spinner_upper_select(self, spinner : kivy.uix.spinner.Spinner, text : string) : None
+ on_mocap_spinner_select(self, spinner : kivy.uix.spinner.Spinner, text : string) : None
+ on_keypoints_spinner_select(self, spinner : kivy.uix.spinner.Spinner, text : string) : None
+ on_toggle_button_press(self, toggle_button : kivy.uix.togglebutton) : None
+ doo3(self, instance : Any) : None
+ open_file_dialog(self, args : Iterable) : None
+ update_selected_files_layout(self) : None
```

Поля класса MocapScreenApp:

main_layout

Тип: *kivy.uix.boxlayout*

Описание: Основной контейнер для размещения элементов второго окна

chosen_internet_mode

Тип: *string*

Описание: Строка, содержащая выбранный интернет мод: онлайн\оффлайн

chosen_model_builder_mode

Тип: *string*

Описание: Строка, содержащая выбранный вариант сборки скрипта: extr_keypoints\mocap

chosen_fps

Тип: *string*

Описание: Строка содержащая выбранное количество fps - параметр собираемого скрипта

chosen_keypoints_model

Тип: *string*

Описание: Строка содержащая выбранный вариант сборки для скрипта extract_keypoints

chosen_parts

Тип: *string[]*

Описание: Список выбранных флагов частей тела для добавления их в собираемый скрипт

selected_files

Тип: *string[]*

Описание: Список выбранных файлов для обработки

chosen_face_hunting

Тип: *string*

Описание: Строка являющаяся флагом о работе модели распознавания лица

left_layout

Тип: *kivy.uix.boxlayout*

Описание: Основной контейнер для размещения элементов на левой половине второго окна

bottom_container

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения элементов на левой нижней половине второго окна

mode_layout

Тип: *kivy.uix.spinner*

Описание: Выпадающий список с интернет модами

selected_files_button

Тип: *kivy.uix.button*

Описание: Кнопка для выбора обрабатываемых файлов

label_and_scrollview_layout

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения сущностей для работы с файлами

selected_files_layout

Тип: *kivy.uix.gridlayout*

Описание: Контейнер для хранения списка обрабатываемых файлов

build_spinner

Тип: *kivy.uix.spinner*

Описание: Выпадающий список с вариантами добавления модели для захвата лица

spinner_upper

Тип: *kivy.uix.spinner*

Описание: Выпадающий список с вариантами сборки скрипта: extr_keypoints\mocap

upper_layout

Тип: *kivy.uix.anchorlayout*

Описание: Контейнер для размещения spinner_upper

right_layout

Тип: *kivy.uix.boxlayout*

Описание: Основной контейнер для размещения элементов на правой половине второго окна

right_keypoints_layout

Тип: *kivy.uix.boxlayout*

Описание: Основной контейнер для размещения элементов на левой половине второго окна в случае сборки скрипта extract_keypoints

keypoints_box

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения keypoints_label и keypoints_spinner

bottom_box

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения path_easymocap_input_1 и path_model_input_1

keypoints_label

Тип: *kivy.uix.label*

Описание: Лейбл с текстом Extract Keypoints

keypoints_spinner

Тип: *kivy.uix.spinner*

Описание: Выпадающий список с вариантами добавления модели для скрипта extract_keypoints

path_easymocap_input_1

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для ввода пути до easymocap в случае выбора сборки скрипта extract_keypoints

path_model_input_1

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для ввода пути до модели в случае выбора сборки скрипта extract_keypoints

check_layout

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения чекбоксов флагов частей тела

toggle_buttons

Тип: *kivy.uix.togglebutton[]*

Описание: Лист для хранения включаемых флагов частей тела

show_script_button1

Тип: *kivy.uix.button*

Описание: Кнопка для отображения собранного скрипта extract_keypoints

script_input_1

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для отображения конечного скрипта extract_keypoints

right_mosap_layout

Тип: *kivy.uix.boxlayout*

Описание: Основной контейнер для размещения элементов на левой половине второго окна в случае сборки скрипта мосар

mosap_box

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения элементов сборки скрипта мосар в правой верхней части окна

path_easymosap_input_2

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для ввода пути до easymosap в случае выбора сборки скрипта мосар

path_model_input_2

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для ввода пути до модели в случае выбора сборки скрипта мосар

mosap_bottom_box

Тип: *kivy.uix.boxlayout*

Описание: Контейнер для размещения элементов сборки скрипта мосар в правой нижней части окна

show_script_button2

Тип: *kivy.uix.button*

Описание: Кнопка для отображения собранного скрипта мосар

script_input_2

Тип: *kivy.uix.textinput*

Описание: Текстовое поле для отображения конечного скрипта мосар

Методы класса *MocapScreenApp*:

on_window_resize(*window, width, height*) → None

Описание:

Метод вызывается при изменении размеров окна (*window*). Внутри метода происходит обновление размеров всех сущностей пользовательского интерфейса в соответствии с новыми размерами окна:

Параметры:

- **window: Any** - окно приложения
- **width: int** - ширина окна
- **height: int** - высота окна

build() → kivy.uix.boxlayout.BoxLayout

Описание:

Метод вызывается при вызове функции `run()`

Внутри метода создаются все сущности класса с начальными параметрами а также встраивается их взаимосвязь

Возвращаемое значение:

- **kivy.uix.boxlayout.BoxLayout** - основной контейнер окна приложения

on_show_script_button1_press(*instance*) → None

Описание:

Метод вызывается при нажатии на кнопку `show_script_button1`

Внутри метода обновляется значение текстового поля `script_input1` на текущее значение собираемого скрипта

Параметры:

- **instance: Any** - Kivy виджет, в данном случае Kivy Button

on_show_script_button2_press(instance) → None

Описание:

Метод вызывается при нажатии на кнопку show_script_button2

Внутри метода обновляется значение текстового поля script_input2 на текущее значение собираемого скрипта

Параметры:

- **instance: Any** - Kivy виджет, в данном случае Kivy Button

get_script() → String

Описание:

Внутри метода собирается итоговый запускаемый скрипт, компоненты которого собираются с заполненного интерфейса пользователя

Возвращаемое значение:

- **String** - собранный скрипт

send_and_wait_until_done() → None

Описание:

Внутри метода происходит сборка Json отправляемого на Client_Worker и запуск subprocess для transfer.py

update_bg_rect(instance, value) → None

Описание:

Этот метод обновляет позицию (pos) и размер (size) заднего фона (bg_rect) в соответствии с позицией и размером виджета (instance), который передается в метод.

Параметры:

- **instance: Any** - Kivy виджет, в данном случае kivy.graphics Rectangle
- **value: int** - численное значение

test_transfer(*instance*) → None

Описание:

Метод вызывается при нажатии на кнопку `button_transfer_test`

Этот метод отключает кнопку `button_transfer_test`, создает новый поток (`threading.Thread`), который запускает функцию `self.send_and_wait_until_done` в фоновом режиме. Отключение кнопки и выполнение в фоновом режиме помогает избежать блокировки пользовательского интерфейса во время выполнения длительных операций.

Параметры:

- **instance: Any** - Kivy виджет, в данном случае Kivy Button

on_mode_layout_select(*spinner, text*) → None

Описание:

Этот метод вызывается при выборе значения в спиннере для выбора режима интернета. Обновляет переменную `chosen_internet_mode` и выводит выбранное значение в консоль.

Параметры:

- **spinner: kivy.ui.spinner** - передаваемый объект спиннер
- **text: String** - передаваемая строка в спинере

on_build_spinner_select(*spinner, text*) → None

Описание:

Этот метод вызывается при выборе значения в спиннере для выбора режима сборки. Обновляет переменную `chosen_face_hunting` и выводит выбранное значение в консоль.

Параметры:

- **spinner: kivy.ui.spinner** - передаваемый объект спиннер
- **text: String** - передаваемая строка в спинере

on_spinner_upper_select(*spinner, text*) → None

Описание:

Этот метод вызывается при выборе значения в спиннере для выбора режима сборки модели. Обновляет переменную `chosen_model_builder_mode` и в зависимости от выбранного значения изменяет содержимое правого основного лейаута.

Параметры:

- **spinner: kivy.ui.spinner** - передаваемый объект спиннер
- **text: String** - передаваемая строка в спинере

on_mocap_spinner_select(*spinner*, *text*) → None

Описание:

Этот метод вызывается при выборе значения в спиннере для выбора ускорения в режиме mocap. Обновляет переменную `chosen_fps` и выводит выбранное значение в консоль.

Параметры:

- **spinner:** `kivy.ui.spinner` - передаваемый объект спиннер
- **text:** `String` - передаваемая строка в спинере

on_keypoints_spinner_select(*spinner*, *text*) → None

Описание:

Этот метод вызывается при выборе значения в спиннере для выбора модели keypoints. Обновляет переменную `chosen_keypoints_model` и выводит выбранное значение в консоль.

Параметры:

- **spinner:** `kivy.ui.spinner` - передаваемый объект спиннер
- **text:** `String` - передаваемая строка в спинере

on_toggle_button_press(*toggle_button*) → None

Описание:

Этот метод вызывается при нажатии на кнопку переключения (ToggleButton). Обрабатывает состояние кнопки (включено/выключено) и соответствующим образом изменяет список `hosen_parts`.

Параметры:

- **toggle_button:** `kivy.ui.togglebutton` - передаваемая кнопка

do03(*instance*) → None

Описание:

Этот метод запускает новый поток, который вызывает функцию `open_file_dialog` с аргументами.

Параметры:

- **instance:** `Any` - в данном случае Kivy Button

open_file_dialog(*args*) → None

Описание:

Этот метод открывает диалоговое окно для выбора файлов с использованием библиотеки `QFileDialog` из Qt. Выбранные файлы сохраняются в переменной `self.selected_files`. Затем используется `Clock.schedule_once` для запланированного выполнения метода `self.update_selected_files_layout` в основном потоке Kivy.

Параметры:

- **args:** `Any` - любые параметры запуска диалогового окна Qt

update_selected_files_layout() → None

Описание:

Этот метод обновляет виджеты пользовательского интерфейса для отображения выбранных файлов в виде виджетов Label. Очищает текущее содержимое, а затем добавляет Label для каждого выбранного файла в GridLayout.

2.1.1.4 Дизайн

Встраивание дизайна для CamerScreenApp и MosapScreenApp осуществляется напрямую в код этих классов.

Для работы с дизайном дополнительно были добавлены библиотеки:

- **SpinnerOption** - надстройка для Spinner, наследуемая от нее для класса CustomSpinnerOption
- **ctypes** - для того чтобы создавать типы данных C и манипулировать ими в Python (пользуемся при создании заголовка окна)

Для покраски сущностей `kivy.uix.label`, `kivy.uix.button`, `kivy.uix.boxlayout` добавлены параметры `background_normal` и `background_down`

непосредственно в эти же сущности

Для покраски `kivy.uix.spinner` добавлен кастомный класс:

CustomSpinnerOption
+ background_normal : string + background_down : string

Также добавлена настройка заднего фона и функция

set_window_icon(icon_path) → None

Описание:

В этой функции происходит установка заголовка для консоли `ctypes` и устанавливается значок для окна

Параметры:

- **icon_path** - путь до встраиваемой иконки

2.1.2 Client Worker

Client_Worker - класс исполнитель всего бекенда блока Клиент.

2.1.2.1 Архитектура блока

Структура класса Client_Worker:

Client_Worker
<div>+ internet_connection : bool + inner_cameras = string[] + ip_cameras = string[] + ID : int + names : string[] + number_of_files : int</div>
<div>+ get_list_of_all_available_inner_cameras() : string[] + check_correct_ip_address(address : string) : bool + check_correct_port(port : string) : bool + check_correct_ip_port(ip_port : string) : bool + url_to_ip_port(url : string) : (boolean, string) + check_camera_connection(source : string, timeout_seconds : int = 5) : bool + check_internet_connection() : bool + recieve_data_from_server(server_host : string, server_port : string, save_path : string) : none + send_video(server_host : string, server_port : string) : none + send_preordered_receipt(files : file[], meta : META, server_host : string, server_port : string) : none + get_number_of_files(meta : META) : int + punch_container(server_host : string, server_port : string) : bool + get_video_back(id : int, name : string, server_host : string, server_port : string) : none + punch_wrapper(server_host : string, server_port : string) : none + submit(DATA, event : Event, server_host : string, server_port : string) : none + display_camera_stream(stream_source : string, window_name : string, FOURCC : int., FRAMES_PER_SECOND : float, RESOLUTION : (int, int), VIDEO_WRITERS : {string : cv2.VideoWriter}, events_to_turn_on : {string : threading.Event}, events_to_start_recording : {string : threading.Event}, events_to_stop_recording : {string : threading.Event}, events_to_turn_off : {string : threading.Event}): none</div>

Поля класса Client_Worker:

internet_connection

Тип: bool

Описание: показывает наличие подключения к интернету

inner_cameras

Тип: string[]

Описание: список подключенных встроенных камер устройства

ip_cameras

Тип: string[]

Описание: список подключенных IP-камер

ID

Тип: *int*

Описание: текущих идентификационный номер пакета для обработки

names

Тип: *string[]*

Описание: Список имен файлов, отправленных в пакете для обработки

NUMBER_OF_FILES

Тип: *int*

Описание: Количество файлов в пакете для обработки

check_internet_connection() → None:

Проверяет подключение к интернету. Сеттер поля *internet_connection*

transfer_video_via_sftp(*local_path*, *remote_path*, *hostname*, *username*, *password*, *port*=22) → None

Описание: Метод для передачи видеофайла на удаленный сервер по протоколу SFTP.

Параметры:

- *local_path* (тип: **str**): Путь к локальному видеофайлу.
- *remote_path* (тип: **str**): Путь на удаленном сервере, куда будет передан видеофайл.
- *hostname* (тип: **str**): Хост сервера SFTP.
- *username* (тип: **str**): Имя пользователя для подключения по SFTP.
- *password* (тип: **str**): Пароль для подключения по SFTP.
- *port* (тип: **int**, по умолчанию: 22): Порт для подключения по SFTP.

download_file_via_sftp(*remote_path, local_path, hostname, username, password, port=22*) -> **None**

Описание: Метод для загрузки файла с удаленного сервера по протоколу SFTP.

Параметры:

- *remote_path* (тип: **str**): Путь к файлу на удаленном сервере.
- *local_path* (тип: **str**): Путь, по которому будет сохранен файл локально.
- *hostname* (тип: **str**): Хост сервера SFTP.
- *username* (тип: **str**): Имя пользователя для подключения по SFTP.
- *password* (тип: **str**): Пароль для подключения по SFTP.
- *port* (тип: **int**, по умолчанию: 22): Порт для подключения по SFTP.

send_preordered_receipt(*files, meta, server_host, server_port*) → **None**:

Описание: Подключается к удаленному серверу, передает информацию пакете, получает обратно id пакета, устанавливает значение поля ID .

Параметры:

- *files*(тип: **list**):название файлов пакета.
- *meta*(тип: **dict**): Метаинформация о передаваемом пакете.
- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

send_video(*name, server_host, server_port*) → **None**:

Описание: Подключается к удаленному серверу, передает информацию об элементе пакета, вызывает отправку файла, отправляет терминирующий суб-пакет.

Параметры:

- *name*(тип: **str**):название файла.
- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

punch_container(*server_host, server_port*) → **None**:

Описание: Подключается к удаленному серверу, пытается получить информацию об обработанном пакете, в случае успеха вызывает загрузку видео.

Параметры:

- *name*(тип: **str**):название файла.
- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

get_video_back(*id*, *name*, *server_host*, *server_port*) → **None**:

Описание: Подключается к удаленному серверу, вызывает загрузку файлов..

Параметры:

- *name*(тип: **str**): название файла.
- *id* (тип: **int**) : идентификационный номер пакета
- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

punch_wrapper(*server_host*, *server_port*) → **None**:

Описание: функция обертка, раз в несколько секунд вызывающая методы обращения к серверу, чтобы получить обратно обработанный пакет.

Параметры:

- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

submit(*DATA*, *server_host*, *server_port*) → **None**:

Описание: вызывается для отправки всех файлов из DATA на удаленный сервер вместе с инструкцией к выполнению. Получает обработанные файлы обратно.

Параметры:

- *DATA* (тип: **dict**): Полная информация о пакете .
- *server_host*(тип: **str**): Хост сервера.
- *server_port*(тип: **int**): Порт для подключения

get_list_of_all_available_inner_cameras(*self*) → (**int**, **str**)[]:

Возвращает пронумерованный список всех доступных встроенных камер устройства с помощью граббера.

display_camera_stream(*self*, *stream_source*, *window_name*, *FOURCC*, *FRAMES_PER_SECOND*, *RESOLUTION*, *VIDEO_WRITERS*, *events_to_turn_on*, *events_to_start_recording*, *events_to_stop_recording*, *events_to_turn_off*) → **None**:
Отображает входящий с *stream_source* поток. Позволяет сохранять записанный материал.

Параметры:

stream_source – источник видео

window_name – название окна в котором будет отображаться видео

FOURCC – Four Character Code

FRAMES_PER_SECOND – количество кадров в секунду для записи

RESOLUTION – разрешение видео

VIDEO_WRITERS – словарь событий отключения камер

events_to_turn_on – словарь событий подключения камер

events_to_start_recording – словарь событий начала записи с камер

events_to_stop_recording – словарь событий остановки записи с камер

events_to_turn_off – словарь событий отключения камер

check_correct_ip_address(*self*, *address*) → **bool**:

Проверяет, является ли введенная строка IP-адресом с портом.

Параметры:

address – проверяемая строка

Возвращаемые значения:

True – *address* является действительным IP-адресом

False – *address* не является IP-адресом

check_correct_port(*self*, *port*) → **bool**:

Проверяет, является ли введенная строка портом.

Параметры:

port – проверяемая строка

Возвращаемые значения:

True – *port* является допустимым значением порта

False – *port* не является портом

check_correct_ip_port(*self*, *ip_port*) → **bool**:

Проверяет, является ли введенная строка IP-адресом с портом.

Параметры:

ip_port – проверяемая строка

Возвращаемые значения:

True – *ip_port* является действительным IP-адресом с портом

False – *ip_port* не является портом IP-адресом с портом

url_to_ip(self, url) → (bool, string):

Преобразует RTSP URL в IP-адрес

Параметры:

url – преобразуемая строка

Возвращаемые значения:

True – успешно получен IP-адрес

False – не удалось извлечь IP-адрес

url_to_ip_port(self, url) → (bool, string):

Преобразует RTSP URL в IP-адрес с портом

Параметры:

url – преобразуемая строка

Возвращаемые значения:

True – успешно получен IP-адрес с портом

False – не удалось извлечь IP-адрес с портом

check_camera_connection(self, source, timeout_seconds) → bool:

Проверяет подключение к IP-камере

Параметры:

source – источник видеопотока (RTSP)

timeout_seconds – время попыток подключения

Возвращаемые значения:

True – успешное подключение по *source*

False – не удалось подключиться по *source*

2.1.2.2 Импортруемые зависимости/библиотеки/фреймворки

threading – возможность запускать код на разных потоках

cv2 – OpenCV – подключение камер

fpygrabber.dshow_graph.FilterGraph – класс для получения информации о
встроенных устройствах

os – доступ к файлам системы

datetime – работа с датой и временем

subprocess – библиотека для работы с процессами

platform – библиотека для работы с системными данными

re – используется для верификации корректности введенных данных

time – библиотека для работы со временем

ipaddress – библиотека для работы с ip

2.1.2.3 Описание работы

Класс *Client_Worker* отвечает за подключение к IP-камерам и ко встроенным камерам устройства, позволяет сохранять видеоматериал, полученный с камер; позволяет отправлять файлы на сервер и получать обратно обработанные данные. Содержит вспомогательные методы для обработки введенных пользователем значений. *Client_Worker* вызывается через обращения к его объектам и методам из блока UI, и работает в скрытом от Пользователя режиме, за исключением логирования передачи данных.

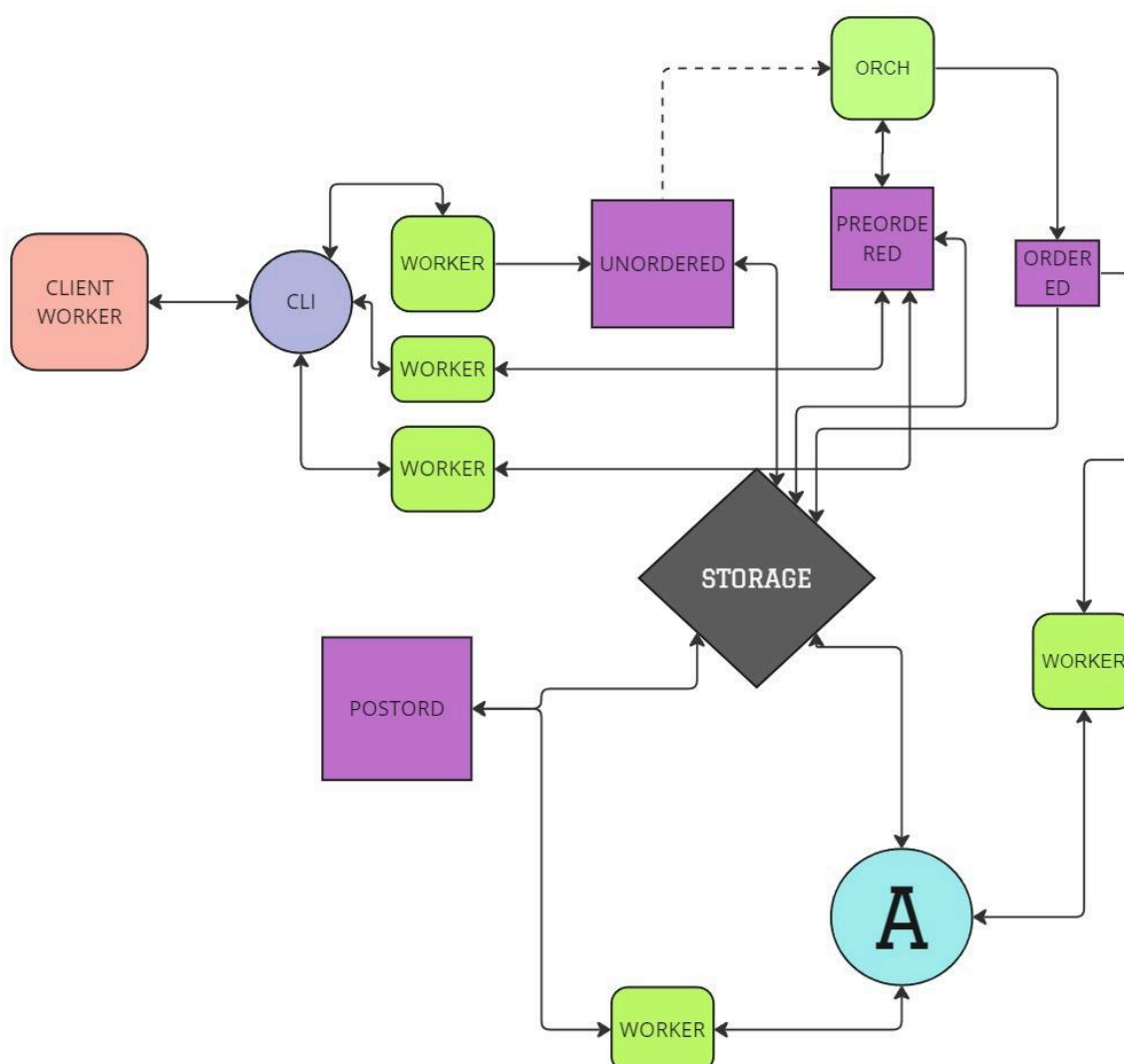
2.2. Сервер

Блок Сервер - это распределительный облачный сервер, соединяющий вычислительный блок и клиента, а также формирующий очередь выполнения запросов от клиентов.

Запросы, которые посылает Пользователь представляют из себя пакеты с метаданными, которые передаются через сокеты и большие видеоданные, которые передаются по защищенному протоколу трансфера файлов.

На рисунке ниже показана общая логика имплементации Сервера в работу ПО:

- CLI - блок клиента (пункт 2.1)
- A - Вычислительный блок (пункт 2.3)
- Worker - Server_Worker (пункт 2.2.1)
- Postord, Unordered, Preordered, Ordered - таблицы Базы Данных (пункт 2.2.3)
- ORCH - Оркестратор (пункт 2.2.2)

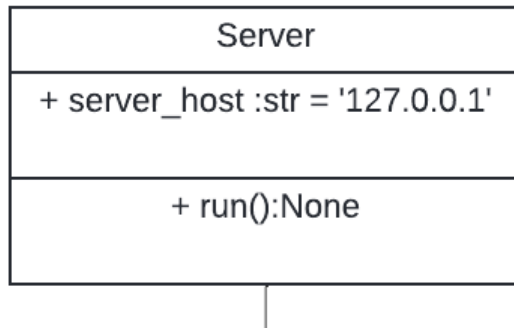


Клиент может общаться с Сервером через подключения по сети, управляет этими подключениями на стороне Сервера класс `Server_Worker`. Аналогично для Вычислительного блока. На сервере постоянно работают в фоновых потоках (далее демон-треды) функции, являющиеся методами класса `Server_Worker`.

2.2.1 Server

Server - внешнеуровневый класс для управления блоком Сервер.

2.2.1.1. Архитектура блока



Поля класса Server:

server_host

Тип: *str*

Описание: хост распределительного облачного сервера

Методы класса Server:

run() → None

Описание:

Метод, создающий объекты классов Server Worker и Orchestrator и запускающий их методы в демон тредах.

2.2.1.2. Импортируемые зависимости/библиотеки/фреймворки

В своей работе класс Server использует:

- **sys** - для отключения системы администратором изнутри
- **threading** - для создания демон тредов с работающими функциями
- **Server_worker** - класс Server_Worker для работы выполнения основной работы на блоке Сервер
- **Orchestrator** - класс Orchestrator, отвечающий за распределение пользовательских запросов, формирование исполнительной очереди ради снижения нагрузки на вычислительную машину.

2.2.1.3. Описание работы

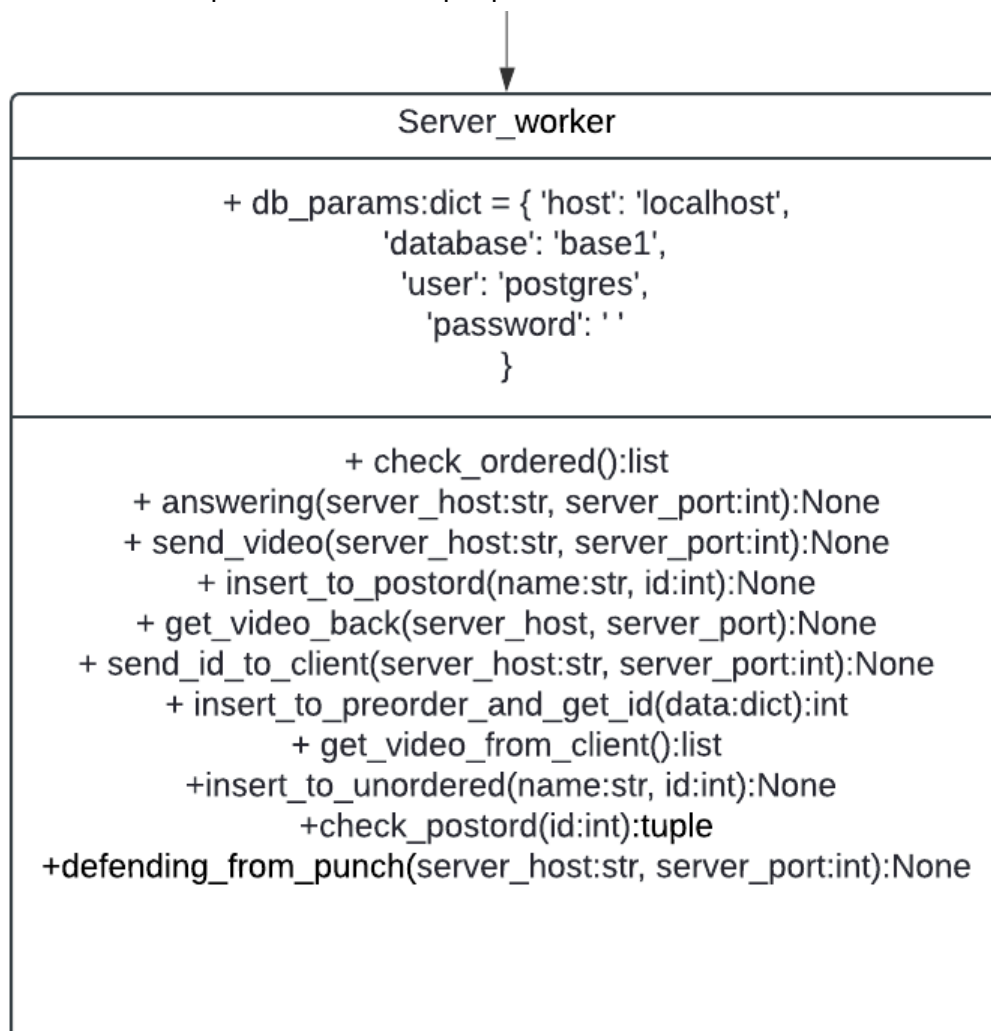
Объект класса Server запускается в работу с помощью команды `run()`, после чего блок Сервер активируется и начинает принимать и отправлять сигналы.

2.2.2 Server Worker

Класс `Server_Worker` - класс для выполнения задач принятия и отправки данных, расположения их в хранилище сервера, а также для первичного распределения в базу данных.

2.2.2.1. Архитектура блока

Блок `Server_Worker` представляет из себя класс, методы которого вызываются в различные моменты работы блока Сервер.



Поля класса *Server_Worker*:

db_params

Тип: *dict*

Описание: Словарь с конфигурационными данными для базы данных

Методы класса *Server_Worker*:

check_ordered() → **list**

Описание: Проверяет, появились ли в таблице `ordered` новые запросы для выполнения на вычислительной машине.

Возвращаемое значение:

- **list** - объект из базы данных

answering(server_host, server_port) → **None**

Описание: Принимает входящие подключения от Вычислительного Блока, вызывает `check_ordered()`, отправляет пакет с мета информацией о пакете обратно на Вычислительный Блок

Параметры:

- **server_host : str** - хост сервера
- **server_port: int** - порт для подключения

insert_to_postord(name, id) → **None**

Описание: Добавляет в таблицу `postord` информацию об обработанном Вычислительным блоком элементе пакета

Параметры:

- **name: str** - имя файла
- **id: int** - идентификационный номер пакета

get_video_back(server_host, server_port) → **None**

Описание: Принимает входящие подключения от Вычислительного Блока, принимает входящую мета информацию о пакете с Вычислительного блока, вызывает `insert_to_postord()`

Параметры:

- **server_host : str** - хост сервера
- **server_port: int** - порт для подключения

send_id_to_client(server_host, server_port) → None

Описание: Принимает входящие подключения от Клиента, принимает метаинформацию о пакете, вызывает insert_to_preorder_and_get_id(), отправляет id пакета обратно на Клиент

Параметры:

- *server_host* : **str** - хост сервера
- *server_port*: **int** - порт для подключения

insert_to_preorder_and_get_id(data) → int

Описание: Добавляет в таблицу preordered метаинформацию о пакете, возвращает id пакета

Параметры:

- *data*: **dict**- метаинформация о пакете

Возвращаемое значение:

- **int**- идентификационный номер пакета

get_video_from_client(server_host, server_port) → None

Описание: Принимает входящие подключения от клиента, принимает информацию об элементе пакета, вызывает insert_to_unordered()

Параметры:

- *server_host* : **str** - хост сервера
- *server_port*: **int** - порт для подключения

insert_to_unordered(name, id) → None

Описание: Добавляет в таблицу unordered информацию об элементе пакета

Параметры:

- *name*: **str** - название файла
- *id*: **int** - идентификационный номер пакета

check_postord(id) → tuple()

Описание: Проверяет количество записей в таблице postord с id = *id*, возвращает (количество, list[имена файлов])

Параметры:

- *id* : **int**- идентификационный номер пакета

Возвращаемое значение:

- **tuple**- (количество элементов postord с id = *id* , list[имена файлов])

defending_from_punch(server_host, server_port) → None

Описание: Принимает входящее подключение со стороны Клиента, принимает входящую метаинформацию о пакете, вызывает check_postord(), отправляет ответ Клиенту

Параметры:

- *server_host* : **str** - хост сервера
- *server_port*: **int** - порт для подключения

2.2.2.2. Импортируемые зависимости/библиотеки/фреймворки

- **pickle** - библиотека для сериализации данных
- **socket** - библиотека для работы с сокетами
- **os** - библиотека для работы с системой
- **threading** - библиотека для работы с тредами
- **psycopg2** - библиотека для работы с базами данных
- **psycopg2.extras.Json** - класс для работы с сериализованными для БД данными
- **DB.config_for_DB.db_params** - словарь с конфигурационными данными для БД

2.2.2.3. Описание работы

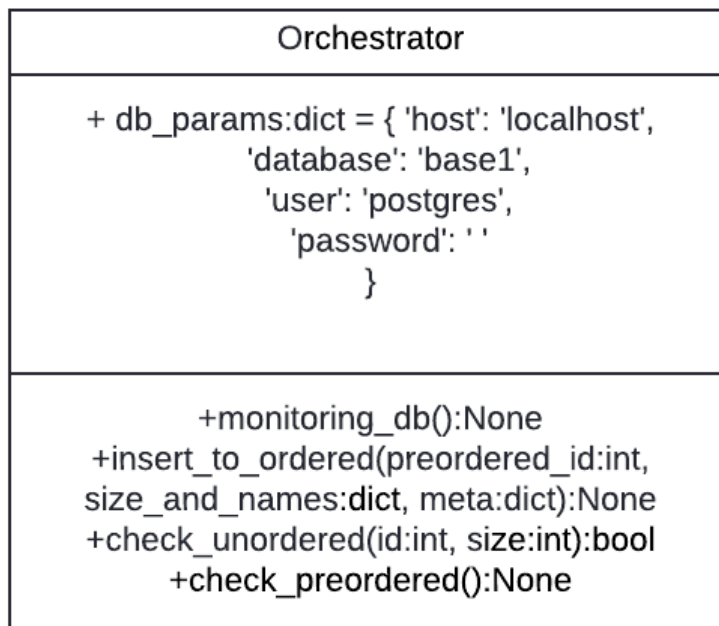
При запуске run() класса Server создается 5 объектов класса Server_Worker. Все они в демон тредах слушают определенные порты.

Когда происходит подключение вычислительного блока или клиента, эти функции запускают свои сценарии, в том числе передачу данных через сокеты и распределение поступающей информации в базу данных.

2.2.3. Orchestrator

Orchestrator - управляющий класс блока Сервер. Работа его методов организует очередь выполнения запросов на Вычислительном блоке, а также управляет данными в базе данных.

2.2.3.1. Архитектура блока



Поля класса Orchestrator:

db_params

Тип: *dict*

Описание: Словарь с конфигурационными данными для базы данных

Методы класса Orchestrator:

monitoring_db() → None

Описание: Функция, которая каждые 5 секунд вызывает check_preordered

insert_to_ordered(preordered_id, size_and_names, meta) → None

Описание: Функция, которая добавляет информацию о пакете в таблицу ordered базы данных.

Параметры:

- *preordered_id: int* - идентификационный номер пакета
- *size_and_names: dict* - метаданные о пакете
- *meta: dict* - метаданные о пакете

check_preordered() → **None**

Описание: Функция, которая проверяет таблицу `preordered` на наличие новых пакетов и в случае обнаружения запускает проверку на полноту пакета используя `check_unordered()`, после чего может отправить информацию о пакете в `ordered`

check_unordered(id, size) → **bool**

Описание: Функция, которая проверяет полноту пакета на сервере

Параметры:

- **id: int**- идентификационный номер пакета
- **size: int**- метаданные о пакете(размер пакета)

Возвращаемое значение:

- **bool** - True, если пакет полный, False иначе

2.2.3.2. Импортируемые зависимости/библиотеки/фреймворки

- **time** - библиотека для работы со временем
- **psycpg2.extras.Json** - класс для работы с сериализованными данными в бд
- **DB.config_for_DB.db_params** - словарь с конфигурацией БД
- **import psycpg2** - библиотека для работы с БД

2.2.3.3. Описание работы

Объект класса `Orchestrator` создается при выполнении `run()` класса `Server`, после чего функция `monitoring_db` запускается в демон тред. Она следит за новыми объектами в таблице `preordered`, для каждого объекта (пакета) проверяя его полноту (загруженность `unordered`), после чего отправляет полные пакеты в определенном порядке в `ordered`, формируя тем самым очередь для выполнения на Вычислительном блоке.

Класс `Orchestrator` необходим для распределения на облачном распределительном сервере.

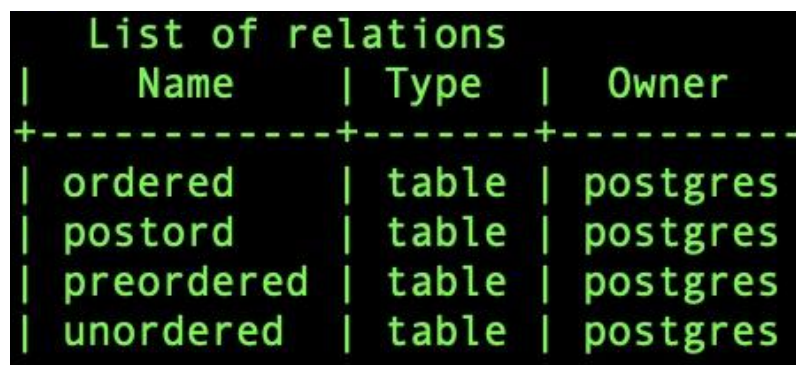
2.2.4 DB

Блок DB - это блок базы данных. Она развернута на распределительном сервере, то есть в блоке Сервер и управляется с помощью классов `Server_Worker` и `Orchestrator`. Этот блок нужен, чтобы работать с приходящими запросами пользователей не в рантайме, а через постоянные объекты. Реализация через PostgreSQL и `psycopg2`, скрипты для разворачивания хранятся в исходных кодах.

2.2.4.1. Архитектура блока

Основная база - `base1`, доступ к которой получается при помощи конфигурационных данных **DB.config_for_DB.db_params**

База состоит из 4 основных таблиц:



List of relations		
Name	Type	Owner
ordered	table	postgres
postord	table	postgres
preordered	table	postgres
unordered	table	postgres

unordered				
Название поля	id	name	package_id	counted
Тип	SERIAL PRIMARY KEY	VARCHAR(255) NOT NULL	INTEGER	BOOLEAN NOT NULL
Назначение	id объекта	название файла	id пакета	Учтен ли при подсчете полноты пакета

preordered				
Название поля	id	size_and_names	meta	ready_to_evolve
Тип	SERIAL PRIMARY KEY	JSON	JSON	BOOLEAN NOT NULL
Назначение	id пакета	json, содержащий размер пакета и имена всех файлов в пакете	метаинформация пакета(задача для Вычислительного блока)	Отправлен ли пакет в очередь для отправки на Вычислительный блок

ordered					
Название поля	id	package_id	size_and_names	meta	send_to_pipe
Тип	SERIAL PRIMARY KEY	INTEGER	JSON	JSON	BOOLEAN NOT NULL
Назначение	id объекта	id пакета	json, содержащий размер пакета и имена всех файлов в пакете	метаинформация пакета(задача для Вычислительного блока)	Отправлен ли пакет в обработку на Вычислительный Блок

postord			
Название поля	id	package_id	name
Тип	SERIAL PRIMARY KEY	INTEGER	VARCHAR(255) NOT NULL
Назначение	id объекта	id пакета	Имя файла

2.2.4.2. Импортируемые зависимости/библиотеки/фреймворки

PostgreSQL 12

2.2.4.3. Описание работы

Контролируемое подключение пользователей происходит через функции `Server_Worker`. Передаваемый пакет по частям обрабатывается Сервером:

- Метаинформация о пакете передается в таблицу `preordered`.
- Основные файлы передаются в `unordered`
- `Orchestrator` формирует очередь исполняемых запросов на основе метаданных из `preordered` и `unordered`, и помещает их в таблицу `ordered`.
- Вычислительный блок забирает данные из `ordered`.
- Когда с Вычислительного блока возвращаются обработанные данные, они помещаются в таблицу `postord`, откуда забираются классом `Server_worker` для передачи обратно пользователю.

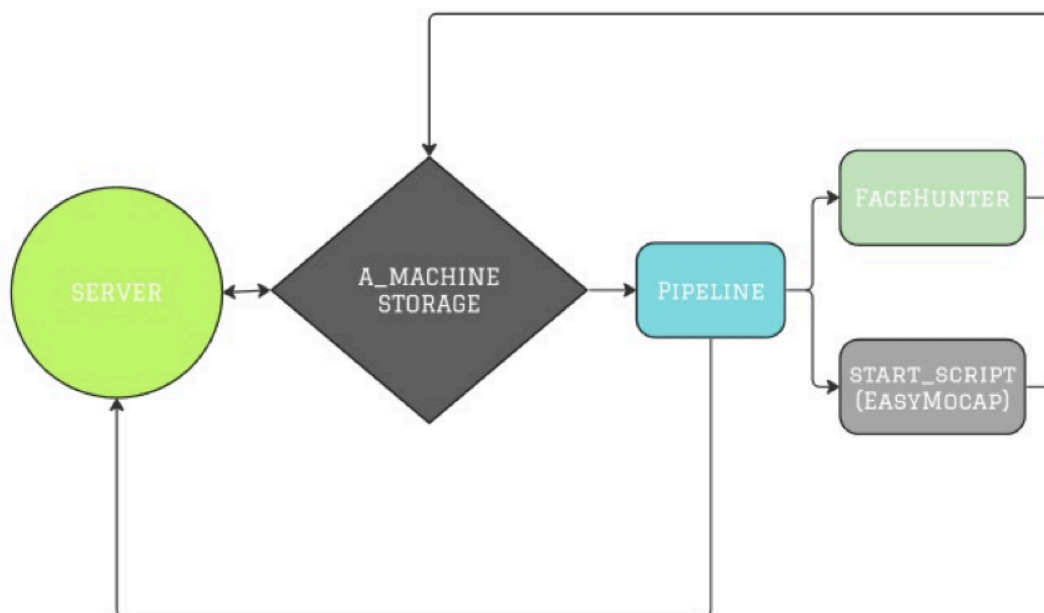
Таким образом каждый пакет идентифицируется в системе уникальным `id`.

2.3 Вычислительный блок

На этом этапе происходит вся работа с видео файлом, включающая в себя разбиение видео на кадры, выделение ключевых точек, создание модели или маски, и обратный сбор кадров в видео.

Блок, который получает видео клиента, после чего обрабатывает его, в зависимости от выбранных режимов. Результатом его работы будет либо 3D модель человеческого тела, наложенная на видео или созданная отдельным файлом со специальным расширением .bvh, либо маска на лице, характеризующая его мимику.

Далее более подробно показаны этапы работы этого блока:

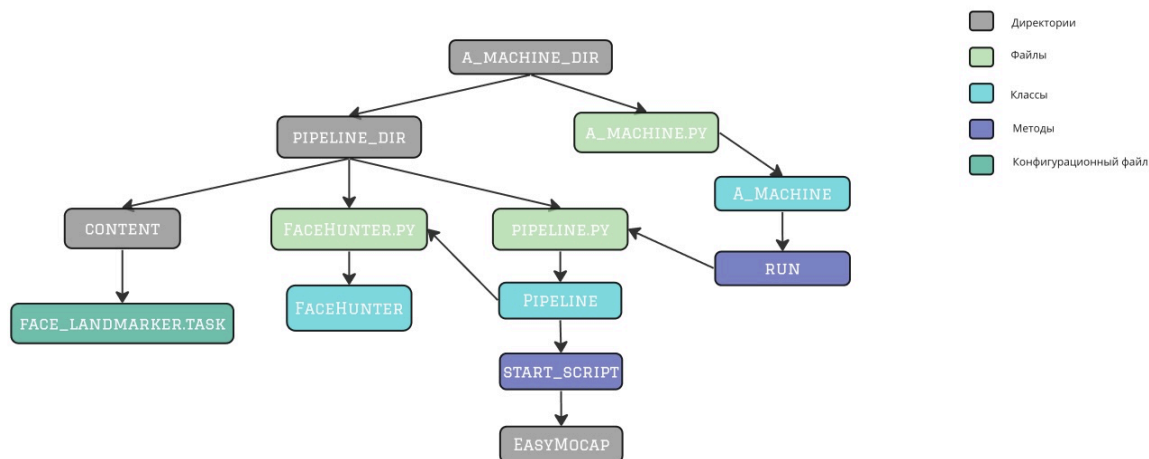


Вычислительный блок подключается к распределительному облачному серверу, откуда он может получать данные для обработки от Пользователей. Получаемые необработанные файлы размещаются в хранилище A_Machine_storage.

Собранные данные при помощи класса Pipeline отправляются либо на обработку мимики лица (класс FaceHunter), или, если режим - создание 3D модели тела человека, то на обработку методами класса Pipeline, который в свою очередь обращается к функциям EasyMocap.

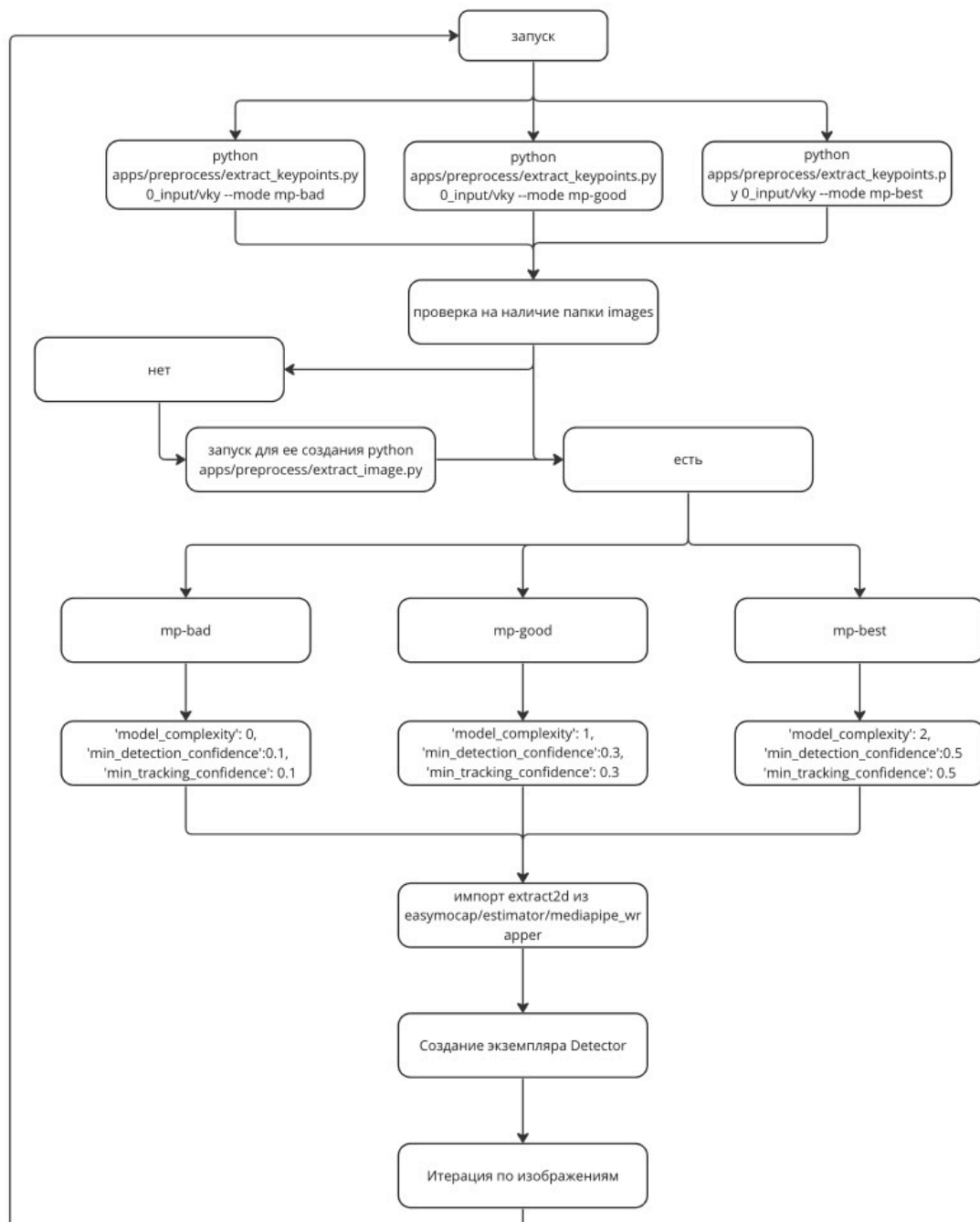
Вне зависимости от режима, результат работы сохраняется в A_Machine_storage, откуда передается обратно на блок Сервер через методы Pipeline.

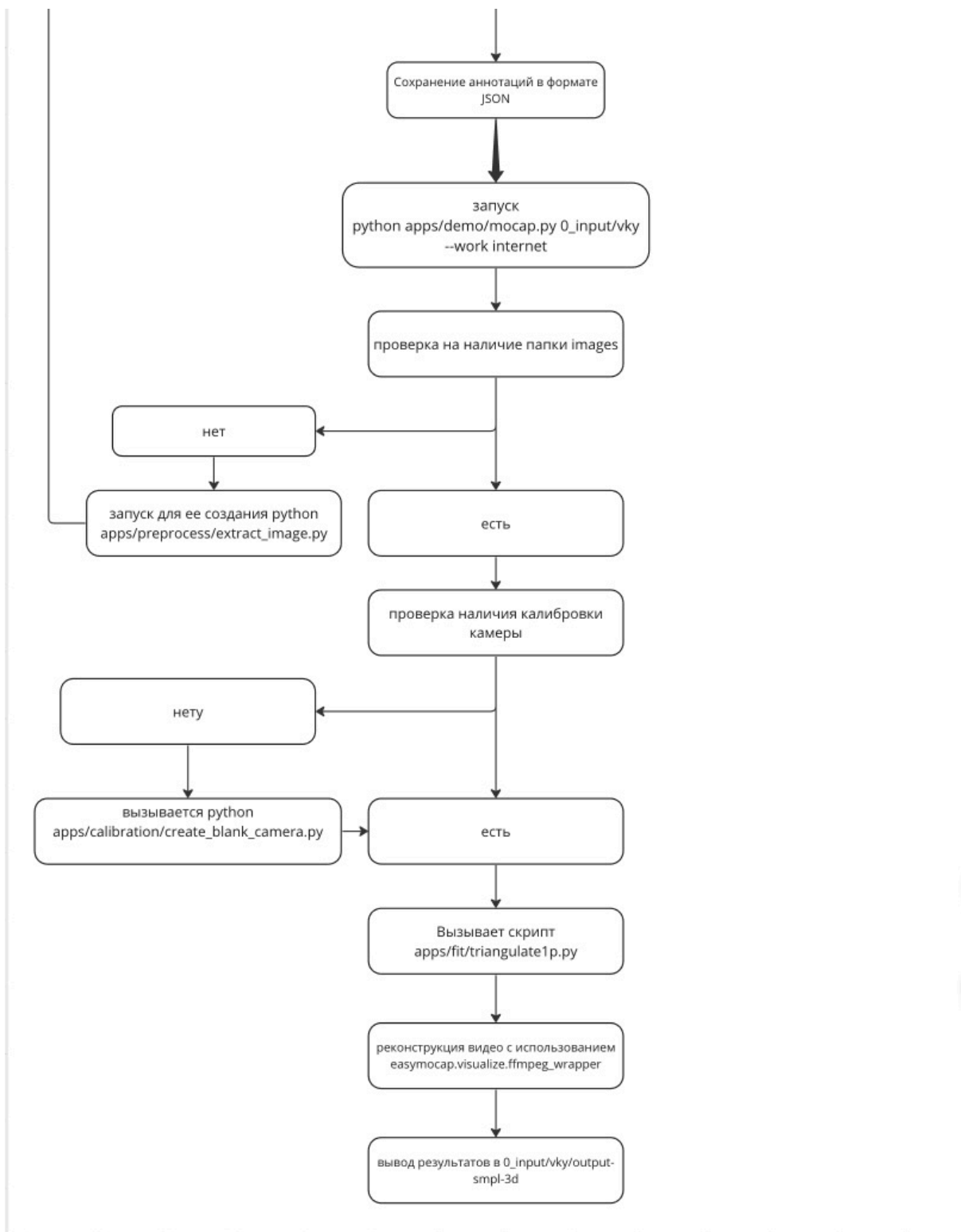
Структура файловой системы Вычислительного блока:



- A_Machine_dir - директория, отвечающая за работу вычислительного блока.
 - A_Machine.py - файл, содержащий в себе верхнеуровневый класс A_Machine, запускающий работу Pipeline.
 - Pipeline_dir:
 - Pipeline.py - файл, содержащий класс Pipeline, который в зависимости от параметров запускает либо FaceHunter, либо из метода start_script работу EasyMocap.
 - FaceHunter.py - файл, содержащий класс FaceHunter, выполняющий считывание мимики лица.
 - Content:
 - Face_Landmarker.task - конфигурационный файл, к которому обращается FaceHunter

Этапы работы EasyMoscap:

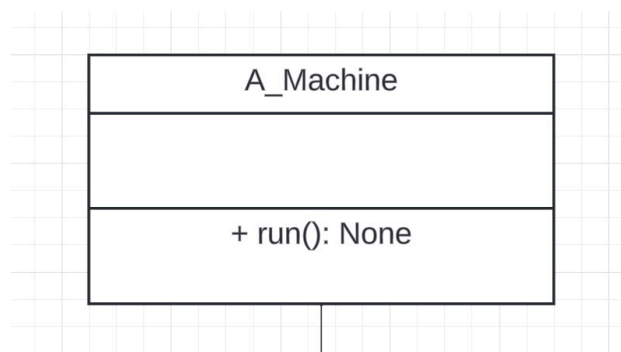




2.3.1 A_Machine

Класс A_Machine - верхнеуровневый класс для управления Вычислительным блоком

2.3.1.1. Архитектура блока



Поля класса A_Machine:
отсутствуют

Методы класса A_Machine:

run() → None

Описание:

Метод, создающий объекты класса Pipeline и запускающий его методы.

2.3.1.2. Импортируемые зависимости/библиотеки/фреймворки

- **time** - предоставляет функции для работы с временем и измерения прошедшего времени.
- **Pipeline** - класс Pipeline для выполнения основной работы на Вычислительном блоке

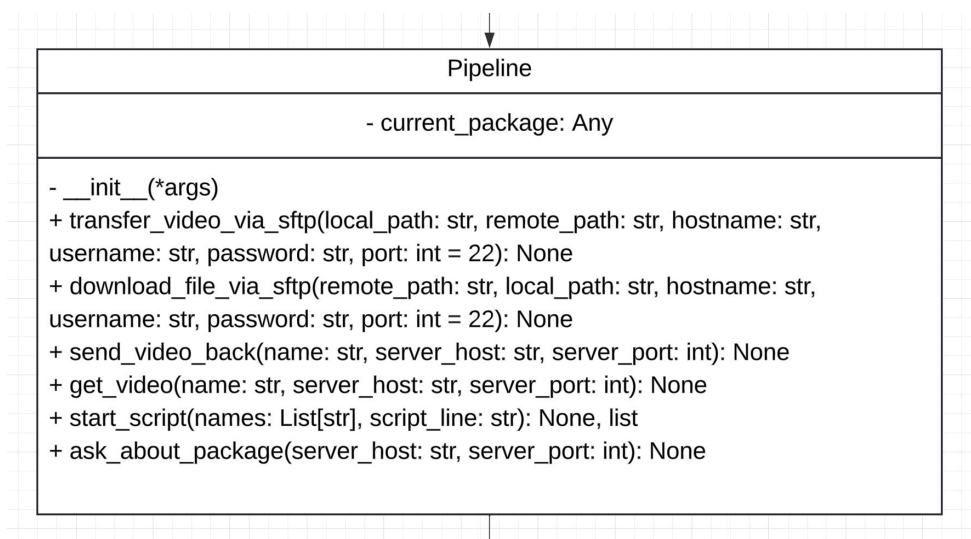
2.3.1.3. Описание работы

Объект класса запускается в работу с помощью команды run(), в исполняемом файле a_machine.py, что в свою очередь запускает весь Вычислительный Блок

2.3.2 Pipeline

Класс Pipeline представляет собой часть системы, отвечающую за управление передачей и обработкой видеоданных. Внутри класса реализованы методы для передачи видеофайлов через протокол SFTP, а также обработки запросов и отправки видеофайлов обратно на сервер. Дополнительно, класс запускает скрипты обработки видео в зависимости от поступающих запросов.

2.3.2.1. Архитектура блока



Поля класса Pipeline:

current_package

Тип: any

Описание: переменная класса, предназначенная для хранения текущего пакета данных, полученного от сервера. Инициализируется значением None в конструкторе класса.

Методы класса Pipeline:

transfer_video_via_sftp(local_path, remote_path, hostname, username, password, port=22) -> None

Описание: Метод для передачи видеофайла на удаленный сервер по протоколу SFTP.

Параметры:

- *local_path* (тип: **str**): Путь к локальному видеофайлу.
- *remote_path* (тип: **str**): Путь на удаленном сервере, куда будет передан видеофайл.
- *hostname* (тип: **str**): Хост сервера SFTP.
- *username* (тип: **str**): Имя пользователя для подключения по SFTP.
- *password* (тип: **str**): Пароль для подключения по SFTP.
- *port* (тип: **int**, по умолчанию: 22): Порт для подключения по SFTP.

download_file_via_sftp(*remote_path*, *local_path*, *hostname*, *username*, *password*, *port*=22) -> **None**

Описание: Метод для загрузки файла с удаленного сервера по протоколу SFTP.

Параметры:

- *remote_path* (тип: **str**): Путь к файлу на удаленном сервере.
- *local_path* (тип: **str**): Путь, по которому будет сохранен файл локально.
- *hostname* (тип: **str**): Хост сервера SFTP.
- *username* (тип: **str**): Имя пользователя для подключения по SFTP.
- *password* (тип: **str**): Пароль для подключения по SFTP.
- *port* (тип: **int**, по умолчанию: 22): Порт для подключения по SFTP.

send_video_back(*name*, *server_host*, *server_port*) -> **None**

Описание: Метод для отправки видеофайла обратно на сервер, включая передачу по SFTP и отправку метаданных.

Параметры:

- *name* (тип: **str**): Название видеофайла.
- *server_host* (тип: **str**): Хост сервера.
- *server_port* (тип: **int**): Порт сервера.

get_video(*name*, *server_host*, *server_port*) -> **None**

Описание: Метод для получения видеофайла с сервера.

Параметры:

- *name* (тип: **str**): Название видеофайла.
- *server_host* (тип: **str**): Хост сервера.
- *server_port* (тип: **int**): Порт сервера.

start_script(*names*, *script_line*) -> **None | list**

Описание: Метод для запуска скриптов обработки видео в зависимости от поступающих запросов.

Параметры:

- *names* (тип: **list[str]**): Список названий видеофайлов.
- *script_line* (тип: **str**): Строка, представляющая команду для выполнения скрипта.

Возвращаемое значение:

- **list** - список новых видеофайлов после обработки скриптами.

ask_about_package(server_host, server_port) -> None

Описание: Метод для обработки запросов от сервера, включая получение метаданных, загрузку видео и запуск скриптов.

Параметры:

- **server_host** (тип: **str**): Хост сервера.
- **server_port** (тип: **int**): Порт сервера.

2.3.2.2. Импортируемые зависимости/библиотеки/фреймворки

- **shutil**: Библиотека для работы с операциями над файлами и директориями.
- **socket**: Модуль для работы с сетевыми сокетами.
- **json**: Библиотека для работы с JSON-данными.
- **threading**: Модуль для создания и управления потоками выполнения.
- **time**: Модуль для работы с временем и временными задержками.
- **os**: Модуль для взаимодействия с операционной системой.
- **pickle**: Модуль для сериализации/десериализации объектов Python.
- **pysftp**: Библиотека для взаимодействия с SFTP-серверами.
- **FaceHunter**, **run** из модуля FaceHunter: Класс и функция для обработки видеофайлов с использованием библиотеки mediapipe.

2.3.2.3. Описание работы

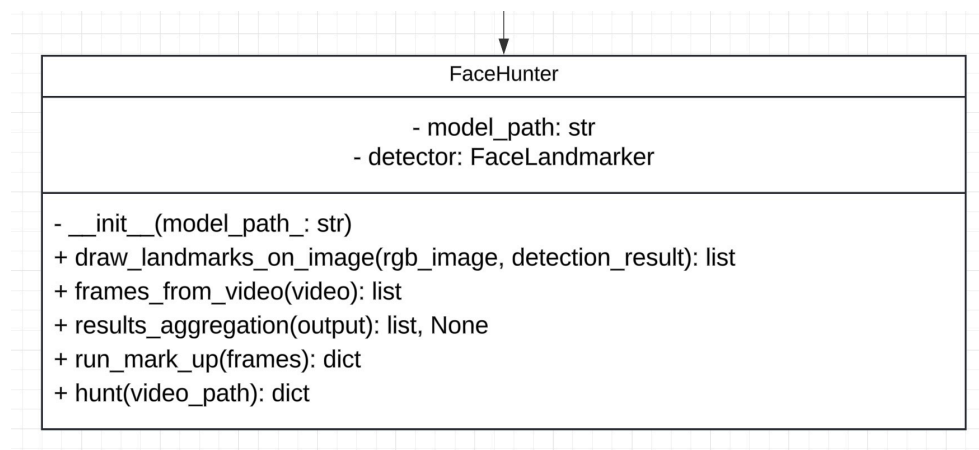
При создании экземпляра класса инициализируется переменная `current_package` значением `None`. Класс предоставляет методы для передачи видеофайлов по протоколу SFTP, обработки запросов от сервера, запуска скриптов обработки видео и других операций.

Основной метод `run` выполняет бесконечный цикл опроса сервера и вызова соответствующих методов в зависимости от поступающих запросов. Класс использует библиотеки для работы с файлами, сокетами, временными задержками, а также сторонние библиотеки для взаимодействия с SFTP-серверами и обработки видео.

2.3.3 FaceHunter

FaceHunter представляет собой класс, разработанный для обнаружения ключевых точек лица в видеопотоке. Класс использует библиотеку `mediapipe`, а точнее, задачу `vision.FaceLandmarker`, для выполнения обнаружения лица и ключевых точек на каждом кадре видео

2.3.3.1. Архитектура блока



Поля класса FaceHunter:

model_path

Тип: str

Описание: Путь к файлу модели для обнаружения ключевых точек лица.

detector

Тип: vision.FaceLandmarker

Описание: Экземпляр класса, используемого для обнаружения ключевых точек лица.

Методы класса FaceHunter:

draw_landmarks_on_image(rgb_image, detection_result) -> np.ndarray

Описание: Метод для визуализации ключевых точек лица на изображении.

Параметры:

- *rgb_image* (тип: **np.ndarray**): Изображение в формате RGB.
- *detection_result* (тип: **vision.FaceLandmarks**): Результат обнаружения ключевых точек лица.

Возвращаемое значение:

- **np.ndarray** - изображение с отмеченными ключевыми точками лица.

frames_from_video(video) -> List[np.ndarray]

Описание: Метод для извлечения кадров из видео.

Параметры:

- *video* (тип: **cv2.VideoCapture**): Объект видео.

Возвращаемое значение:

- **List[np.ndarray]** - список кадров из видео.

results_aggregation(output) -> Union[None, List[Dict[]]]

Описание: Метод для агрегации результатов обнаружения ключевых точек лица.

Параметры:

- *output* (тип: **vision.FaceLandmarks**): Результат обнаружения ключевых точек лица.

Возвращаемое значение:

- **Union[None, List[Dict[]]]** - список словарей с координатами ключевых точек.

run_mark_up(frames) -> dict

Описание: Метод для запуска обнаружения ключевых точек лица на кадрах.

Параметры:

- *frames* (тип: **List[np.ndarray]**): Список кадров из видео.

Возвращаемое значение:

- **Dict[str, Union[List[Dict[]], List[np.ndarray]]]** - словарь с ключами "point4D" и "images", содержащий соответственно координаты ключевых точек и изображения с отмеченными точками.

hunt(video_path) -> dict

Описание: Метод для обработки видео с обнаружением ключевых точек лица.

Параметры:

- *video_path* (тип: **str**): Путь к видеофайлу.

Возвращаемое значение:

- **dict** - словарь с ключами "point4D" и "images", содержащий соответственно координаты ключевых точек и изображения с отмеченными точками.

Внешние функции:

write_images_to_path(images, path) -> bool

Описание: Метод для записи изображений в указанную директорию.

Параметры:

- *images* (тип: **List[np.ndarray]**): Список изображений.
- *path* (тип: **str**): Путь к директории для сохранения изображений.

Возвращаемое значение:

- **bool** - True, если сохранение прошло успешно.

write_dict(*points*, *path*) -> **None**

Описание: Метод для записи словаря с координатами точек в JSON-файл.

Параметры:

- *points* (тип: **List[Dict[str, Union[float, int]]]**): Список словарей с координатами точек.
- *path* (тип: **str**): Путь к JSON-файлу.

images_to_video(*image_folder*, *video_name*='output_video.mp4') -> **None**

Описание: Метод для создания видеоизображения из изображений в указанной директории.

Параметры:

- *image_folder* (тип: **str**): Путь к директории с изображениями.
- *video_name* (тип: **str**, по умолчанию: 'output_video.mp4'): Имя создаваемого видеофайла.

run(*MODEL_PATH*, *VIDEO_PATH*, *OUTPUT_PATH*, *JSON_PATH*, *save*=True) -> **dict**

Описание: Метод для запуска обработки видео с обнаружением ключевых точек лица.

Параметры:

- *MODEL_PATH* (тип: **str**): Путь к файлу модели для обнаружения ключевых точек лица.
- *VIDEO_PATH* (тип: **str**): Путь к видеофайлу.
- *OUTPUT_PATH* (тип: **str**): Путь к директории для сохранения изображений.
- *JSON_PATH* (тип: **str**): Путь к JSON-файлу для сохранения координат точек.
- *save* (тип: **bool**, по умолчанию: True): Флаг, указывающий, сохранять ли результаты обработки.

Возвращаемое значение:

- **dict** - словарь с ключами "point4D" и "images", содержащий соответственно координаты ключевых точек и изображения с отмеченными точками.

2.3.3.2. Импортируемые зависимости/библиотеки/фреймворки

- **json**: Библиотека для работы с JSON-данными.
- **mediapipe**: Библиотека для обработки видео и извлечения информации, включая обнаружение ключевых точек лица.

- **cv2** (OpenCV): Библиотека для обработки изображений и работы с видео.
- **numpy**: Библиотека для выполнения математических операций и работы с массивами данных.
- **matplotlib.pyplot**: Библиотека для визуализации данных и изображений.
- **os**: Модуль для взаимодействия с операционной системой, в данном случае, для создания директорий.
- **ImageSequenceClip** из `moviepy.editor`: Библиотека для работы с видеофайлами, в данном случае, для создания видеоизображения из последовательности изображений.

2.3.3.3. Описание работы

FaceHunter обеспечивает функциональность по обнаружению ключевых точек лица в видеопотоке. Этот класс может использоваться для обработки видеофайлов, извлечения ключевых точек, визуализации результатов и сохранения изображений с отмеченными точками. В результате обработки, класс возвращает словарь, содержащий координаты ключевых точек (`point4D`) и изображения с отмеченными точками (`images`).

3. Приложение к Документации Разработчика

3.1 Общая UML-диаграмма, отражающая связь классов между собой

Доступ по [ссылке](#)