Командный тур

6.1. Легенда

Современные банки в погоне за привлечением клиентов разрабатывают и вводят в эксплуатацию все более комфортные сервисы.

Например, российские банки Сбербанк и Тинькофф-банк вводят в строй банкоматы, способные идентифицировать человека по лицу:

- `https://tass.ru/ekonomika/4402287`
- `https://www.banki.ru/news/lenta/?id=10740603`

Банк "Ак Барс" планирует использовать технологию *Face2Pay* в образовательных учреждениях для оплаты школьниками питания: `https://tass.ru/ekonomika/5607346`.

Не только в России (`https://www.banki.ru/news/lenta/?id=10670764`), но и в других странах мира (`https://www.vesti.ru/doc.html?id=3126288`) разрабатываются системы для оплаты общественного транспорта с помощью технологий распознавания человека по лицу.

Также стремясь увеличить скорость перемещения денежных средств (в частности для межбанковских и международных переводов) банки рассматривают альтернативные платежные системы, основанные на технологии блокчейн:

- `https://www.jpmorgan.com/global/news/digital-coin-payments`
- `http://coinlog.ru/6-mirovyh-bankov-zapuskayut-stejblkoiny-v-obshhej-blokchejn-seti-ibm/`
- `https://bits.media/pyat-bankov-yaponii-obedinilis-dlya-sozdaniya-blokcheyn-resheniy/`
- `https://cryptoratings.ru/news/alfa-bank-podklyuchilsya-k-blokchejn-seti-marco-polo/`

Поскольку перечисленные выше технологии становятся трендом в современном финансовом секторе, то участниками профиля "Программная инженерия финансовых технологий" предлагается разработать прототип программного обеспечения банкомата, проводящего операции в блокчейн сети, и идентифицирующего пользователя по лицу.

При этом разработка модели машинного обучения для распознавания и идентификации человека по лицу остается за рамками данного прототипа. Как и в реальных индустриальных проектах, на стадии прототипа вместо разработки собственной модели будет применяться уже готовое решение, а именно сервис *Microsoft Face API*,

наиболее удовлетворяющий нужды данного проекта.

В общих чертах работа банкомата описывается следующим образом:

- Пользователь идентифицируется банкоматом по лицу.
- В качестве дополнительной меры безопасности банкомат может запросить от пользователя выполнить определенные действия, для проверки, что перед камерой банкомата не "плоское" изображение пользователя и не маникен.
- Поскольку блокчейн сеть авторизует пользовательские операции посредством приватного ключа пользователя, ключ генерируется на основании идентификатора пользователя, возвращенного сервисом идентификации человека по лицу, и PIN-кодом, введенным человеком. Таким образом, происходит защита от перехвата индентификатора человека.
- После получения приватного ключа возможны операции получения баланса, проведение платежей, получения истории платежей.
- Платежи другим пользователям осуществляются по их номеру телефона

Следовательно, чтобы ПО банкомата могло работать необходимо реализовать дополнительные сервисные функции, осуществляемые при регистрации нового пользователя в систему (например, в офисе банка):

- Управление пользователями в сервисе распознования по лицу: добавление и удаление пользователя и изображений его лица.
- Управление соответствями между пользователем и его номером телефона: регистрация и удаление соответствия.
- Для избежания мошенничеств регистрация и удаление соответствия должны подтверждаться авторизованным лицом (администратором в банке).

6.2. Набор заданий

Решение командной задачи разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулирует необходимый функционал, который должен быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволяющих проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагается решать в отдельном этапе (*итерации*). Подзадачи в первом наборе позволяют школьникам продемонстрировать базовое понимание принципов решения данной задачи. Они могут быть решены с использованием тех разработок, котороые были реализованы участниками финала на учебно-тренировочных сборах. Поздадачи второй итерации позволяют реализовать минимально рабочий продукт. Третья итерация состоит из подзадач повышенной сложности, которые требуют от старшеклассников продемонстрировать более глубокое понимание принципов написания контрактов для сети блокчейн, а также использовать знания из олимпиады по программированию для оптимизации работы с видеопотоком.

На каждом этапе решение подзадач проверялось с помощью системы автоматической системы оценивания, которая запускала решение участников с теми или иными параметрами в соответствии с приемочными тестами.

Первая итерация

Участникам необходимо разработать базовый функционал для скриптов `setup.py`, `face-management.py` и `faceid.py`, реализовать соответствующие Ethereum-контракты, тем самым решая следующие подзадачи:

user story (*)	acceptance criteria
US-001 Регистрация контракта	все
US-004 Простое добавление пользователя в сервис индентификации	все
US-013 Получение баланса идентифицированного пользователя	все

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- инициализация аккаунтов тестовой публичной сети Ethereum для тестирования решения в ходе процесса разработки;
- настройка окружения разработки программного обеспечения для использования системы ведения версий *Git*, Python-библиотек для взаимодействия с узлами блокчейн платформы Ethereum и *Microsoft Face API*;
- написание и отладка программ на языке Python, работающих с параметрами командной строки;
- обеспечение отправки JSON-RPC запросов (платежные операции) из программ на языке Python к провайдерам `Web3`;
- написание и отладка Ethereum контрактов на языке Solidity;
- подготовка тестовых данных — видео-файлов для работы с сервисом распознавания человека по лицу;
- получение отдельных кадров из видео-потока;
- обеспечение отправки REST запросов к сервису *Microsoft Face API* из программ на языке Python.

Вторая итерация

Участникам необходимо завершить работу над минимально достаточным прототипом программного обеспечения банкомата, который бы позволял

- идентифицировать человека по лицу;
- управлять соответствиями между аккаунтом пользователя и его телефонным номером;
- отправлять средства другому зарегистрированному в системе пользователю.

Данный функционал покрывался следующими подзадачами:

user story (*)	acceptance criteria
US-002 Вывод владельца контракта регистра соответствий	все
US-003 Изменение владельца контракта регистра соответствий	все
US-006 Получение всех пользователей из сервиса идентификации	все
US-007 Удаление пользователя из сервиса идентификации	все
US-008 Запуск обучения сервиса индентификации	все, кроме AC-008-04
US-010 Идентификация пользователя	все, кроме AC-010-04
US-014 Отправка запроса на регистрацию соответствия	все
US-015 Отправка запроса на удаление соответствия	все
US-016 Отмена запроса на регистрацию или удаление соответствия	все
US-017 Отправка средств	все
US-021 Получение истории платежей	все
US-023 Получение всех запросов на регистрацию соответствий	все
US-024 Получение всех запросов на удаление соответствий	все
US-025 Подтверждение запросов на регистрацию или удаление соответствий	все
US-026 Получение аккаунта по номеру телефона	все

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- взаимодействие с сервисом распознавания человека по лицу для наполнения его необходимыми данными, обучения и выполнения операций распознавания;
- разработка контракта с простейшей моделью разграничения доступа;
- разработка схемы взаимодействия структур данных для контракта с заданными характеристиками;
- обеспечение отправки JSON-RPC запросов (транзакции на вызов методов Ethereum контрактов, получение данных из блокчейн сети) из программ на языке Python к провайдерам `Web3`;
- выстраивание процесса интеграционного тестирования компонент приложения, работающего с сервисом распознавания лица и узлами блокчейн сети.

Третья итерация

Работа участников должна быть направлена на наращивание функционала прототипа программного обеспечения банкомата. В ходе третьей итерации должна была появиться защита от мошенничества при операциях распознавания по лицу и возможность формировать отложенные платежи. После заверешения итерации у участников должно быть 3 Python-скрипта и 2 Ethereum-контракта, закрывающие последнии набор подзадач:

user story (*), все acceptance criteria
US-005 Улучшенное добавление пользователя в сервис индентификации
US-008 Запуск обучения сервиса индентификации
US-009 Обнаружение уже добавленного пользователя
US-010 Идентификация пользователя
US-011 Запрос действий на безопасную идентификацию пользователя
US-012 Безопасная идентификация пользователя
US-018 Генерация сертификата на получение средств
US-019 Использование сертификата на получение средств
US-020 Вернуть средства из неиспользованных сертификатов
US-022 Получение истории платежей, включая использование сертификатов

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- оптимизация работы с видео-потоком для уменьшения количества обрабатываемых кадров;
- проверки истинности информации с использованием цифровой подписи.

6.3. Условия проведения

- В первые день соревнований все члены команд получают ноутбук со следующим набором установленного программного обеспечения:
  - набор Python инструментария Anaconda с установленными библиотеками `web3` (`v5.0.0a3`), `cognitive_face`, `opencv-python`;
  - среды разработки PyCharm и WingIDE;
  - компилятор языка Solidity `solc` (`v0.5.4`);
  - клиент системы ведения версий `git`;
  - Интернет-браузер Chrome.

  Каждый ноутбук имеет возможность выхода в сеть Интернет. Команды могут использовать собственные ноутбуки.
- Команды могут устанавливать дополнительное программное обеспечение, но после согласования с членами жюри.
- Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.
- Участники должны использовать язык программирования Python для написания программ, использующих командную строку. Для написания Ethereum контрактов участники могут использовать любой язык программирования.
- Для работы с сервисом *Microsoft Face API* участникам предоставляется один ключ подписки на команду, а также базовый URL для доступа к REST API. При доступе к сервису с помощью данного ключа действует ограничение на 10 запросов в секунду. Участники одной команды должны сами заботиться о том, чтобы держать ключ в тайне от других команд. Ключ не должен передаваться третьим лицам. Если ресурс ключа подписки (примерно 60000 запросов) полностью используется командой, то организаторы вправе не предоставлять команде другой ключ.

- Участники не могут использовать помощь тренера, сопровождающего лица или привлекать третьих лиц для решения задачи.
- Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются, баллы. Баллы за подзадачи можно получить только в день, закрепленный за конкретным этапом.
- В начале первого дня состязаний участники каждой команды получают доступ к репозиторию на серверах GitLab.com. Каждая команда имеет свой собственный репозиторий. Члены других команд не имеют доступ к чужим репозиториям.
- В течение дня не ведется учет количества изменений, которые команды регистрируют в Git-репозитории.
- В конце каждого дня финального этапа жюри проверяет решение участников на соответствие приемочным тестам для каждой подзадачи, входящей в набор для соответствующей итерации.
- Баллы за все подзадачи, для которых прошло приемочное тестирование, определяют баллы, набранные командой в данный день соревнований.
- Система автоматического тестирования имеет следующую конфигурацию:
  - OS Linux
  - Python `v3.6`
  - Python модули (перечислены) и соответствующие зависимости (не перечислены): `web3` (`v5.0.0a3`), `opencv-python`, `cognitive_face`, `dlib`, `imutils`, `ethereum`
  - `/usr/local/bin/solc` (`v0.5.4`)
  - `/opt/shape_predictor_68_face_landmarks.dat`
- После выставления баллов, командам предоставляется доступ к системе автоматического тестирования, ответственной за проведение приемочных тестов в конкретный день состязаний, так что члены команды могут ознакомиться с логикой проверки и подать апелляцию, если не согласны с корректностью проведения тестов.
- После рассмотрения сути апелляции, жюри вправе провести тестирование вручную и назначить команде баллы за соответствующие подзадачи.
- В начале следующего дня состязаний жюри выдает всем командам свое решение подзадач предыдущего дня, которое команды могут использовать для того, чтобы решать следующий набор подзадач.
- Описанные выше условия могут быть изменены членами жюри. Все изменения в условиях объявляются участникам перед началом каждого дня состязаний.

6.4. Процедура проведения приемочного тестирования и критерии оценки

Для каждого дня соревнований (для каждой итерации) справедлива следующая процедура приемочного тестирования:

- В конце каждого дня финального этапа команды должны сформировать запрос на слияние (*Merge Request*) из своей ветки исходного кода в основную ветку (*master*) в Git-репозитории.
- Команда ответственна за то, чтобы в запросе на слияние не должно быть конфликтов. Запрос на слияние с конфликтами может не рассматриваться жюри для выполнения приемочного тестирования.
- После того, как все команды отправили запросы на слияние, жюри одобряет все запросы и приступает к приемочному тестированию, для тех подзадач, которые входят в соответствующую итерацию. Для этого исходный код приложения команды загружается в систему автоматического тестирования (поддержка которой осуществляется функциональность GitLab CI/CD), где запускаются автоматические тесты на соответствие решения участников требованиям к приемочным тестам (*acceptance criteria*).
- Если все приемочные тесты для данной подзадачи пройдены успешно, команда получает баллы за данную подзадачу. Если хотя бы один тест не проходит, то баллы за данное подзадачу не начисляются.

Приемочные тесты для каждой подзадачи описаны в разделе "Подробное описание подзадач".

Дальше перечислены баллы, которые получает команда за решение подзадач в каждой итерации.

Максимальное количество баллов, которое может набрать команда за решение всех подзадач — 400.

Первая итерация

user story	баллы
US-001 Регистрация контракта	10
US-004 Простое добавление пользователя в сервис индентификации	20
US-013 Получение баланса идентифицированного пользователя	10

Максимальное количество баллов за итерацию — 40.

Вторая итерация

user story	баллы
US-002 Вывод владельца контракта регистра соответствий	10
US-003 Изменение владельца контракта регистра соответствий	10
US-006 Получение всех пользователей из сервиса идентификации	10
US-007 Удаление пользователя из сервиса идентификации	10
US-008 Запуск обучения сервиса индентификации (кроме AC-008-04)	18
US-010 Идентификация пользователя (кроме AC-010-04)	23
US-014 Отправка запроса на регистрацию соответствия	10
US-015 Отправка запроса на удаление соответствия	10
US-016 Отмена запроса на регистрацию или удаление соответствия	15
US-017 Отправка средств	10
US-021 Получение истории платежей	20
US-023 Получение всех запросов на регистрацию соответствий	10
US-024 Получение всех запросов на удаление соответствий	10
US-025 Подтверждение запросов на регистрацию или удаление соответствий	15
US-026 Получение аккаунта по номеру телефона	10

Максимальное количество баллов за итерацию — 191.

Третья итерация

user story	баллы
US-005 Улучшенное добавление пользователя в сервис индентификации	50
US-008 Запуск обучения сервиса индентификации (полностью)	2
US-009 Обнаружение уже добавленного пользователя	20
US-010 Идентификация пользователя (полностью)	2
US-011 Запрос действий на безопасную идентификацию пользователя	5
US-012 Безопасная идентификация пользователя	35
US-018 Генерация сертификата на получение средств	10
US-019 Использование сертификата на получение средств	15
US-020 Вернуть средства из неиспользованных сертификатов	20
US-022 Получение истории платежей, включая использование сертификатов	10

Максимальное количество баллов за итерацию — 169.

Критерии определения команды-победителя командного тура

- Сумма баллов, набранных за решения подзадач командного тура финального этапа, определяет итоговую результативность команды (измеряемую в баллах).
- Команды ранжируются по результативности.
- Команда победитель определяется, как команда с максимальной результативностью.

6.5. Подробное описание подзадач

Решение представляет из себя набор python скриптов (компонент), каждый из который ответственен за определенный функционал.

Управление скриптами происходит через параметры командной строки и через конфигурационные файлы. Параметры командной строки описываются тдельно в разделах, где описывается функциональность каждого скрипта.

Конфигурационные файлы - файлы в формате JSON, должны читаться из текущей директории, где происходит вызов скрипта и могут быть двух типов:

- конфигурационный файл с настройками для работы с *Microsoft Face API*
- конфигурационный файл с настройками для работы с Ethereum узлом

Конфигурационный файл с настройками для MS Face API

Имя: `faceapi.json`

Содержит следующую информацию:

- `key` — ключ подписки для использования сервиса
- `serviceUrl` — URL для доступа к сервису
- `groupId` – группа пользователей (в терминах *MS Face API*), с которой работает в данный момент скрипт

Пример файла:

```
{"key": "563879b61984550e40cbbe8d3039523c",
 "serviceUrl": "https://westeurope.api.cognitive.microsoft.com/face/v1.0/",
 "groupId": "fintech-01"}
```

Конфигурационный файл с настройками для работы с узлом Ethereum

Имя: `network.json`

Содержит следующую информацию:

- `rpcUrl` — URL для доступа к узлу по JSON RPC;
- `privKey` — приватный ключ аккаунта для подписи транзакций, отправляемых на узел Ethereum, и для локальных вызовов;
- `gasPriceUrl` — URL для доступа к сервису, предоставляющему цену за газ. Данный сервис возвращает JSON, из которого нужно взять значение (в *gwei*) из поля `fast`:

  ```
  {"block_time":19.91, "fast":10.0, "instant":25.0, "block_number":7240426,
   "standard":5.0, "health":true, "slow":3.0}
  ```

- `defaultGasPrice` — цена за газ в wei, использующаяся, если нельзя получить цену за газ из сервиса.

Пример файла:

```
{"rpcUrl": "https://sokol.poa.network",
 "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
 "gasPriceUrl": "https://gasprice.poa.network/",
 "defaultGasPrice": 2000000000}
```

Если это явно не прописано в описании функционала соответствующей компоненты, она не должна оставлять после себя никаких файлов-данных. Аналогично с

получением данных - чтение данных из файла должно происходить только в нескольких случаях:

- для получения настроек для работы с *MS Face API*;
- для получения видеопотока, эмулирующего камеру;
- для получения настроек для работы с Ethereum узлом;
- для получения Application Binary Interface контракта или его байткода;
- или если это явно указано в описании компоненты.

в остальных случаях данные должны быть получены из сервиса *MS Face API* или блокчейн сети.

Все компоненты, работающие с сервисом *MS Face API*, должны уметь обрабатывать ситуации, когда:

- недоступно подключение к сервису (сообщение об ошибке ''`No connection to MS Face API provider`'');
- используется некорректный ключ подписки (сообщение об ошибке ''`Incorrect subscription key`'').

Компоненты, отправляющие транзакции в блокчейн, должны уметь обрабатывать ситуации, когда:

- недоступно подключение к узлу Ethereum (сообщение об ошибке ''`No connection to node`'');
- на счету аккаунта, от чьего имени выполняется транзакция, недостаточно средств, чтобы покрыть все затраты по использованному газу (сообщение об ошибке ''`No enough funds to send transaction`'');
- транзакция может не включаться в блок продолжительное время в зависимости от нагрузки на сеть (сообщение об ошибке ''`Transaction is not validated too long`'').

При возникновении данных событий в терминал должно выдаваться сообщение понятное пользователю, а не stack trace.

Возможны также другие конфигурационные файлы, необходимые для работы тех или иных подсистем решения. Данные файлы будут отдельно описываться в соответствующих разделах.

Настройка сервиса KYC

Развертывание и настройка в блокечейн сети регистра соответствий аккаунтов и телефонных номеров (Know Your Customer, KYC) выполняется компанией производителем программного обеспечения (тем, кто разрабатывает данный сервис). Тоже самое относится к обработчику сертификатов на получение средств. Предполагается, что в потенциальные пользователи сервиса (пользователи, регистрирующие соответствия, владельцы платежных терминалов) могут ознакомиться с опубликованным в открытом доступе исходным кодом контрактов, принять решение на основе этого, доверять ли этому разработчику и начать пользоваться сервисом.

Фиксация контрактов в блокчейн гарантирует, что приозводитель не сможет изменить логику работы приложения позднее, т.е. условия участия в системе, с которыми ознакомились пользователи, не поменяется.

Функционал сервиса KYC должен быть доступен сразу, минуя фазы настройки, когда контракт зарегистрирован в блокчейн, но еще находится в нерабочем состоянии. Поэтому первичная настройка базовых параметров по максимуму должна выполняться вовремя инициализации контракта во время его регистрации (deploy).

Имя скрипта
`setup.py <command> [options]`

US-001 Регистрация контракта

Использование скрипта
`$ setup.py --deploy`
Подключается к узлу Ethereum и регистрирует следующие контракты • регистр соответствий аккаунтов и телефонных номеров в сети блокчейн. • обработчик сертификатов на получение средств. На терминал выводятся адреса зарегистрированных контрактов. В текущей рабочей директории создается JSON файл `registrar.json`, содержащий адреса зарегистрированных контрактов.

Пример вызова команды (AC-001-01)

`$ cat registrar.json`
`cat: registrar.json: No such file or directory`
`$ setup.py --deploy`
`KYC Registrar: 0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd`
`Payment Handler: 0x95426f2bC716022fCF1dEf006dbC4bB81f5B5164`
`$ cat registrar.json`
`{"registrar": {"address": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd", "startBlock": 123456}, "payments": {"address": "0x95426f2bC716022fCF1dEf006dbC4bB81f5B5164", "startBlock": 123457}}`

Комментарий: В блокчейн сеть отправляются транзакции для регистрации и первичной настройке контрактов. Транзакции успешно включены в один или несколько блоков. Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Контракты по адресу, выведенным в терминале, созданы одной из отправленных транзакций, и могут быть просмотрены с помощью браузера блоков.

Пример вызова команды (AC-001-02)

```
$ cat registrar.json
{"registrar": {"address": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd", "
startBlock": 123456}, "payments": {"address": "0x95426f2bC716022fCF1dEf00
6dbC4bB81f5B5164", "startBlock": 123457}}
$ setup.py --deploy
KYC Registrar: 0x23B40E5bCd06D819ba81A09e0340Ec06460d2b7D
Payment Handler: 0xE797A1da01eb0F951E0E400f9343De9d17A06bac
$ cat registrar.json
{"registrar": {"address": "0x23B40E5bCd06D819ba81A09e0340Ec06460d2b7D", "
startBlock": 456123}, "payments": {"address": "0xE797A1da01eb0F951E0E400f
9343De9d17A06bac", "startBlock": 456125}}
```

Комментарий: В блокчейн сеть отправляются транзакции для регистрации и первичной настройке контрактов. Транзакции успешно включены в один или несколько блоков. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

US-002 Вывод владельца контракта регистра соответствий

Использование скрипта

```
$ setup.py --owner registrar
```

Подключается к узлу Ethereum и получает адрес аккаунта, имеющего полномочия выполнять действия по подверждению запросов на регистрацию и удалению соответствий аккаунтов и телефонных номеров в сети блокчейн. На терминал выводится адрес аккаунта.

Пример вызова команды (AC-002-01)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045
d85a470",
$ setup.py --deploy
KYC Registrar: 0x9FdddF5bf10c65221da0a78ADAFec1D8E9EF0A7D
Payment Handler: 0xD79A8FDB771Ea12359270aD7020bcCB328C9f5f7
$ setup.py --owner registrar
Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6
```

Комментарий: Транзакции в блокчейн сеть не отправляются.

Пример вызова команды (AC-002-02)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "64e604787cbf194841e7b68d7cd28786f6c9a0a3ab9f8b0a0e87cb438
7ab0107",
$ cat registrar.json
{"registrar": {"address": "0x9FdddF5bf10c65221da0a78ADAFec1D8E9EF0A7D", "
startBlock": 234156}, "payments": {"address": "0xD79A8FDB771Ea12359270aD7
020bcCB328C9f5f7", "startBlock": 451247}}
$ setup.py --owner registrar
Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6
```

Комментарий: Транзакции в блокчейн сеть не отправляются.

US-003 Изменение владельца контракта регистра соответствий

Использование скрипта

```
$ setup.py --chown registrar <address>
```

Подключается к узлу Ethereum и отправляет транзакцию на изменение аккаунта, имеющего полномочия выполнять действия по подверждению запросов на регистрацию и удалению соответствий аккаунтов и телефонных номеров в сети блокчейн. Только аккаунт, в текущее время обладающий полномочиями на выполнение вышеуказанных действий, имеет возможность производить данное изменение.
На терминал выводится адрес нового аккаунта.

Пример вызова команды (AC-003-01)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045
d85a470",
$ setup.py --owner registrar
Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6
$ setup.py --chown registrar 0x6455f1445c72ba9460c6f6ab364d3935a0ad4559
New admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559
$ setup.py --owner registrar
Admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559
```

Комментарий: В блокчейн сеть отправляется транзакция для изменения аккаунта. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-003-02)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045
d85a470",
$ setup.py --owner registrar
Admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559
$ setup.py --chown registrar 0x11460ff94ca4212d9d02b5bea1766dd099e0a9df
Request cannot be executed
$ setup.py --owner registrar
Admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559
```

Комментарий: Транзакции в блокчейн сеть не отправляются.

Пример вызова команды (AC-003-03)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ setup.py --chown registrar 0x11460ff94ca4212d9d02b5bea1766dd099e0a9df
New admin account: 0x11460ff94cA4212d9D02b5bEA1766Dd099E0A9DF
```

Комментарий: В блокчейн сеть отправляется транзакция для изменения аккаунта. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-003-04)

`$ cat network.json | python -mjson.tool | grep privKey`
`"privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",`
`$ setup.py --owner registrar`
`Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6`
`$ setup.py --chown registrar 0x6455f1445c72ba9460c6f6ab364d3935a0ad4559`
`New admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559`
`$ setup.py --owner registrar`
`Admin account: 0x6455F1445c72BA9460C6f6AB364d3935a0AD4559`

Комментарий: Если из транзакции, которая была отправлена в результате команды `setup.py -chown`, извлечь поле `input` и отправить его в новой транзакции снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка, поскольку новый аккаунт, обладающий полномочиями, был изменен транзакцией, посланной командо `setup.py -chown`. Следовательно, изменение аккаунта с полномочиями выполнять действия по подверждению запросов на регистрацию и удалению соответствий аккаунтов и телефонных номеров не происходит. Статус можно подтвердить для данной транзакции в браузере блоков.

Подотовка сервиса идентификации

Сервис идентификации человека по лицу перед полноценной работой требует предварительной настройки. Первое, что должно быть сделано - в сервис необходимо добавить лица людей, которых в дальнейшем необходимо идентифицировать. Поскольку система автоматического тестирования не может работать с камерой, то изображения человека будут передаваться через видео-файл, передаваемый в сервис через параметры командной строки.

Администратор свервиса должен иметь возможность просматривать список добавленных пользователей, удалять пользователя (и его изображения) из системы.

Как только необходимое количество лиц зарегистрировано в сервисе, администратор может запустить обучение нейронной сети.

Имя скрипта
`face-management.py <command> [options]`

US-004 Простое добавление пользователя в сервис индентификации

Сервис должен быть устроен так, что добавление изображений человека в систему происходит анонимно, т.е. имя при добавлении не указывается.

Использование скрипта
`$ face-management.py --simple-add <path to video file>`
При использовании команды простого добавления пользователя из видеопотока извлекается 5 кадров с изображением лица человека. При этом подразумевается, что все кадры в видео принадлежат одному и тому же человеку. Если в видео-потоке недостаточно кадров с изображением человека, то обработка такого видео должно приводить к ошибке.

Пример вызова команды (AC-004-01)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01
" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000
0000000000000000000000000000"
{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou
nd.\r\nParameter name: personGroupId"}}
$ face-management.py --simple-add /path/to/video.avi
5 frames extracted
PersonId: 419e345a-e6d6-4d9c-953d-667787b8d52e
FaceIds
=======
e27558b9-812d-41c3-b114-8e434b8f4602
44c350f2-6653-4616-a1b7-e0fe9b481b6b
f945a3be-4b20-4049-b080-4142a55e4f93
855ab7c2-9bb3-49ed-8cac-1366c0274b08
9c4af288-54cd-4375-8eef-f8c29ed56685
```

Комментарий: Требуемый `personGroupId` не существовал до этого в сервисе *Microsoft Face API*. После добавления `personGroupId` добавляется новый `personId`, с которым ассоциируется 5 изображений лица (`persistedFaceId`).

Пример вызова команды (AC-004-02)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Conte
nt-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000
0000000000000000"
[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
```

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*. В данную группу добавляется новый `personId`, с которым ассоциируется 5 изображений лица (`persistedFaceId`).

Пример вызова команды (AC-004-03)

```
face-management.py --simple-add /path/to/video2.avi
Video does not contain any face
```

Комментарий: Выдается ошибка при попытке обработать видео, в котором либо нет кадров с лицом пользователя, либо в видео содержится меньше 5 кадров. Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-004-04)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 52865cde-3af8-443d-b260-9319c2cb1788
FaceIds
=======
cdb6227e-7453-4057-b4fa-79660914e597
6976d3c2-dee5-4f24-8950-f38ff10c70ad
fae15e55-6639-42a4-a954-731c33310e41
15092567-5765-49ed-ac63-94bc5fa08d17
a77f1f0a-aa95-4bd1-9826-6b453aec42b2
$ face-management.py --simple-add /path/to/video31.avi
5 frames extracted
PersonId: 9fa0a99b-8e76-474d-8223-dea217c2c19b
FaceIds
=======
b552ef11-a162-4a7d-9047-ccfc84a07043
90c0815a-ecce-45c6-8107-ced7ef29a249
fde35dba-505d-4a62-ac5a-c6ae4c89128e
6c6910b4-0ab5-4eb4-9e53-95b1929f9867
fdb9d352-65b0-41a2-a1be-03ea5b543160
$ face-management.py --simple-add /path/to/video41.avi
5 frames extracted
PersonId: f290ecb9-bfab-46f7-b623-45140d730628
FaceIds
=======
e5735ecd-ca09-4fd4-bfd3-8ace67702ab0
9e1bbdee-5981-4f6b-aba5-03be57e5e910
3120ef58-8d53-4558-8b84-784ba338f621
8fceb9c7-f029-4326-9703-6749005674fa
8ea02a3b-7dc0-455a-858c-67251b0ca3b4
```

Комментарий: Несколько добавлений пользователя проходят успешно. Для каждого видео добавляется новый `personId` добавляется.

US-005 Улучшенное добавление пользователя в сервис индентификации

Наличие в наборе изображений одного и того же человека, но у которых есть различия в мимике, положении головы и освещенности, влияет на качество обучения нейронной сети сервиса, поэтому при сборе данных для сервиса идентификации важно собирать разные изображения.

Использование скрипта

```
$ face-management.py --add <path to video file 1> [ <path to video file 2
> [ <path to video file 3> [ <path to video file 4> [ <path to video file
 5> ] ] ] ]
```

Команда ожидает 5 видео файлов. Требования к каждому из видео файлов:

- В первом по списку видеофайле лицо человека должно быть неподвижно (допускаются небольшие повороты)
- Во втором по списку видеофайле должны быть зафиксированы наклоны головы влево и вправо (*roll*)
- В третьем по списку видеофайле должны быть зафиксированы повороты головы влево и вправо (*yaw*)
- В четвертом видеофайле должен быть зафиксирован открытый рот
- В пятом - должно быть зафиксировано закрытие глаз

Пример вызова команды (AC-005-01)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01
" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000
0000000000000000000000000000"
{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou
nd.\r\nParameter name: personGroupId"}}
$ face-management.py --add /path/to/video1.avi
5 frames extracted
PersonId: ddaa7036-cab0-4d8f-9b36-18f20e294c51
FaceIds
=======
5754a049-0de7-4ee5-99ba-45d0d3398645
56c0aa34-50c4-4d0b-bcf0-c86e9c9dec52
f626c3d7-1126-401c-b125-16b6c65e6ed8
6d617030-941d-4f6c-9d05-b714b4e2b504
1087c13c-09d7-4053-8bc6-9381c48081e2
```

Комментарий: Требуемый `personGroupId` не существовал до этого в сервисе *Microsoft Face API*. После добавления `personGroupId` добавляется новый `personId`, с которым ассоциируется 5 изображений лица (`persistedFaceId`). Изображение лица человека не должно значительно отличаться от кадра к кадру (допускаются повороты до 5 градусов).

Пример вызова команды (AC-005-02)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Conte
nt-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000
0000000000000000"
[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]
$ face-management.py --add /path/to/video1.avi
5 frames extracted
PersonId: 6dc70d27-3bad-400a-983f-fec6591cff6b
FaceIds
=======
08609f0a-55ee-46ed-899b-5f7d66f62cce
2bb6d3a9-1c52-4a70-ac62-44881f2aed29
4309d3b7-5250-47e3-bc7e-f3d1da8badd1
76b6a516-c378-4d86-b4cc-03b60b5c2d2c
85f2af43-ca0e-4dd1-a400-9780d7b7a8f5
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 5 изображений лица (`persistedFaceId`). Изображение лица человека не должно значительно отличаться от кадра к кадру (допускаются повороты до 5 градусов).

Пример вызова команды (AC-005-03)

```
$ face-management.py --add /path/to/video2.avi
Base video does not follow requirements
```

Комментарий: Изображение лица человека в видео значительно отличаться от кадра к кадру (повороты больше чем на 5 градусов либо есть открый рот, либо есть закрытые глаза). Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-005-04)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi
10 frames extracted
PersonId: 8d0b7195-c172-4d1a-8588-063f3e64c0c0
FaceIds
=======
08609f0a-55ee-46ed-899b-5f7d66f62cce
2bb6d3a9-1c52-4a70-ac62-44881f2aed29
4309d3b7-5250-47e3-bc7e-f3d1da8badd1
76b6a516-c378-4d86-b4cc-03b60b5c2d2c
85f2af43-ca0e-4dd1-a400-9780d7b7a8f5
300c161f-7154-43c4-80bc-34d1fd9ffb1d
4f57f616-23ea-4be7-b977-8e9b14a192b4
52c9b5c4-6cb7-4aae-b9d0-206d6db90510
8f9a5e7f-c2c4-4d92-908b-6f220b4d3b32
d45f9324-a845-4db6-abc6-88214726bdcc
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 10 изображений лица (`persistedFaceId`). Помимо 5 изображений лиц из первого видео, добавляется 5 - из второго.
Лицо человека во втором видео должно наклонятся из крайнего положения влево к крайнему положению вправо. Зафиксирован максимальный наклон в каждую из сторон не меньше 30 градусов. Из второго видео выбрано 5 кадров с наклонами головы из следующего списка: 30 градусов влево, 15 градусов влево, 0 градусов, 15 градусов вправо, 30 градусов вправо. Допускается отклонение от перечисленных значений +/- 3 градуса.

Пример вызова команды (AC-005-05)

```
$ face-management.py --add /path/to/video1.avi /path/to/video3.avi
Roll video does not follow requirements
```

Комментарий: Во втором по счету видео лицо человека не доходит до крайних положений. Т.е. нельзя зафиксировать максимальный наклон в каждую из сторон от 30 градусов и более. Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-005-06)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi
15 frames extracted
PersonId: a25ddbf2-aabf-41c7-b9a9-d69f1c055761
FaceIds
=======
0ca5925b-55eb-44fd-9624-39fbaa33a5c5
31091841-2e3c-43c2-a14c-497fd8137d25
3f46e69d-33d7-4474-97a1-230046dc9cfa
f1752749-f602-4151-b3bb-7cec627de3df
f556db9a-55cd-4542-bf0f-a5848376ba66
08609f0a-55ee-46ed-899b-5f7d66f62cce
2bb6d3a9-1c52-4a70-ac62-44881f2aed29
4309d3b7-5250-47e3-bc7e-f3d1da8badd1
76b6a516-c378-4d86-b4cc-03b60b5c2d2c
85f2af43-ca0e-4dd1-a400-9780d7b7a8f5
300c161f-7154-43c4-80bc-34d1fd9ffb1d
4f57f616-23ea-4be7-b977-8e9b14a192b4
52c9b5c4-6cb7-4aae-b9d0-206d6db90510
8f9a5e7f-c2c4-4d92-908b-6f220b4d3b32
d45f9324-a845-4db6-abc6-88214726bdcc
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 15 изображений лица (`persistedFaceId`). Помимо 10 изображений лиц из первого и второго видео, добавляется 5 - из третьего. Лицо человека в третьем видео должно поворачиваться из крайнего положения влево к крайнему положению вправо. Зафиксирован максимальный поворот в каждую из сторон не меньше 20 градусов. Из третьего видео выбрано 5 кадров с наклонами головы из следующего списка: 20 градусов влево, 10 градусов влево, 0 градусов, 10 градусов вправо, 20 градусов вправо. Допускается отклонение от перечисленных значени +/- 3 градуса.

Пример вызова команды (AC-005-07)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video4.avi
Yaw video does not follow requirements
```

Комментарий: Во третьем по счету видео лицо человека не доходит до крайних положений. Т.е. нельзя зафиксировать максимальный поворот в каждую из сторон от 20 градусов и более. Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-005-08)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi
16 frames extracted
PersonId: e3152bc7-431c-4215-a19b-d128f2768438
FaceIds
=======
100d7ff0-0e95-4998-a385-d65a69cbbab4
1ca71243-1370-4c7f-b457-ac53ade0b59a
25f51952-880c-435e-9afb-3ca8b2b981d4
cf17ee9e-140b-444b-a87a-35209a631aa1
f0ae9fd1-81bd-4fcc-853c-107e228be383
2307cf52-687b-4d51-9861-728b6297aa7d
34ac19bb-4018-498b-bb13-02098aaac75c
7e01b585-78b1-42a0-8e48-4b1ef199dde7
99943bc7-f92e-49fb-91b4-06b123343982
ae8b5917-ee88-4499-b262-8bfbfda687ca
8195743c-ae84-48fe-b381-1eb73c82f5c1
87ad1c82-0ffd-4b39-8b72-fa6323257802
9cb6fc25-f32e-49d7-b6de-352bf2d911fc
bff40657-1ea4-4667-b2aa-d8b480f527be
d4b57970-6390-4c23-a366-0c51fdc17f9b
1f41144e-eee5-4210-8a7d-ffa4adc3ba21
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 16 изображений лица (`persistedFaceId`). Помимо 15 изображений лиц из первого-третьего видео, добавляется 1 изображение - из четвертого.

На серии кадров человека в четвертом видео рот человека должен быть открыт. Один из кадров с открытым ртом отправляется в *Microsoft Face API*.

Пример вызова команды (AC-005-09)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video5.avi
Video to detect open mouth does not follow requirements
```

Комментарий: Во четвертом по счету видео не найдены кадры с открытым ртом. Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-005-10)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video5.avi
18 frames extracted
PersonId: 588e76bc-a0ce-4066-94d2-96620d105401
FaceIds
=======
220dd1ea-3392-466b-ab50-fbb465a513ff
4977b87d-3c36-45d7-b827-8907aa66b918
29e7d8fd-f065-41b6-9fd0-b191fb3dde78
d0e269f1-199a-459e-9fbf-1c24e015fdec
94d0887a-4e0a-4b86-9e6e-8f6fdd45f1ec
12938747-8530-4bcc-a811-b890aa852175
2d486b65-ed15-4313-836c-2cd4ca7e02cb
9a1ab5a7-9e46-4566-8d85-54bbcd021307
7a6f7afd-a1f6-4d7a-9e6f-c5e0daf1501a
105dfcaa-a25d-45ff-ba38-48de04a735e6
090707ff-91b8-4647-afbd-a80f1976c10d
65f53802-dfa0-46a0-b9e5-c4780f41d580
8c485998-ea43-439b-a84e-606433232133
7b0166db-f7a2-477c-b3f6-bbf3833ea770
191f33fc-501f-4d1a-8b5f-a4f9917f311a
44c79f24-5f87-48c6-af1f-f85e0fca521e
cdd4e100-217d-4163-8f51-4c8f82282f46
e417cb0c-ad18-47eb-8ffa-d850f3aaa8ec
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 18 изображений лица (`persistedFaceId`). Помимо 16 изображений лиц из первого-четвертого видео, добавляется два изображения - из пятого.

На серии кадров человека в пятом видео был закрыт сначала левый глаз, потом правый. По одному кадру с каждым из закрытых глаз отправляется в *Microsoft Face API*.

Пример вызова команды (AC-005-11)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video6.avi
18 frames extracted
PersonId: 5d5d6e92-f8f0-47cf-8e9b-b4455092603e
FaceIds
=======
0e5edd8e-7756-470e-a26c-f54ab70a5524
73b4caea-c85c-4594-9153-198351937d94
2ae3c850-190a-4cd0-afd2-8efa341767ba
adb7744e-b2fb-4a58-b3a8-f22c3143a8cd
7e0f87d6-14ad-4ec6-971f-e4c2771cc267
47c8c307-1f90-473c-b48c-792d5b5a1241
33e35a22-358a-4847-bcdd-4a5bdfd5eb69
2c7b951b-126a-4eec-9e22-1d9d65ece9e3
bcaf7f6a-1c44-476e-a51f-9dde2744ade0
1c9c0134-ada0-47c2-848c-d4424b232f04
b79945e1-de47-4011-98f2-e7e8f0d9f3ef
f972b6ac-078b-4e3d-a779-4cde8ea28f36
fbeb7aa1-f66e-4412-ad07-b61103e8af0b
9c762e64-075b-4fa4-8e39-0d83df86428b
9794e349-e4e4-44ad-951c-610b4890b16e
2e93da04-8f31-4a27-bd9e-5768335447a5
b9227512-dfa8-4a17-94e2-be520b8975a9
b67c1f86-2e3d-45a2-a7fb-57bbbff9cd8c
```

Комментарий: В требуемую `personGroupId` добавляется новый `personId`, с которым ассоциируется 18 изображений лица (`persistedFaceId`). Помимо 16 изображений лиц из первого-четвертого видео, добавляется два изображения - из пятого.
На серии кадров человека в пятом видео был закрыт сначала правый глаз потом левый. По одному кадру с каждым из закрытых глаз отправляется в *Microsoft Face API*.

Пример вызова команды (AC-005-12)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video7.avi
Video to detect closed eyes does not follow requirements
```

Комментарий: Во пятом по счету видео не найдены кадры с закрытыми глазами. Причем видео считается не удовлетворяющим требованиям, если только один глаз был закрыт. Группа с `personGroupId` не создается, новый `personId` не добавляется.

Пример вызова команды (AC-005-13)

```
$ face-management.py --add /path/to/video11.avi /path/to/video12.avi /pat
h/to/video13.avi /path/to/video14.avi /path/to/video15.avi
Roll video does not follow requirements
```

Комментарий: В каком-то видео не найдены кадры, удовлетворяющие требованиям. Группа с `personGroupId` не создается, новый `personId` не добавляется.

US-006 Получение всех пользователей из сервиса идентификации

Администратор сервиса может получить список всех добавленных пользовате-

лей.

Использование скрипта
`$ face-management.py --list`

Пример вызова команды (AC-006-01)

`$ cat faceapi.json | python -mjson.tool | grep groupId`
`"groupId": "fintech-01",`
`$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 00000000000000000000000000000000"`
`{"error":{"code":"PersonGroupNotFound","message":"Person group is not found.\r\nParameter name: personGroupId"}}`
`$ face-management.py --list`
`The group does not exist`

Комментарий: Требуемый `personGroupId` не существовует в сервисе *Microsoft Face API* ни до, ни после выполнения команды.

Пример вызова команды (AC-006-02)

`$ cat faceapi.json | python -mjson.tool | grep groupId`
`"groupId": "fintech-01",`
`$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Content-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000000000000000000"`
`[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]`
`$ face-management.py --list`
`Persons IDs:`
`27dadf08-bc60-4a29-82a7-7d21ea7f40af`
`b8cf9c2f-a606-4f21-851d-26e0a0dc8a74`
`bf4806de-8c4b-4a12-8495-002f43dba797`
`ff79486f-15ac-43be-9c6c-b2840f8c8d22`

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*.

Пример вызова команды (AC-006-03)

`$ cat faceapi.json | python -mjson.tool | grep groupId`
`"groupId": "fintech-01",`
`$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Content-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000000000000000000"`
`[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]`
`$ face-management.py --list`
`No persons found`

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*, но в ней нет пользователей.

US-007 Удаление пользователя из сервиса идентификации

Администратор сервиса может удалить пользователя сервиса по его идентификатору.

Использование скрипта
`$ face-management.py --del <person id>`

Пример вызова команды (AC-007-01)

`$ cat faceapi.json | python -mjson.tool | grep groupId`
`    "groupId": "fintech-01",`
`$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01 " -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000 00000000000000000000000000000000"`
`{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou nd.\r\nParameter name: personGroupId"}}`
`$ face-management.py --del 27dadf08-bc60-4a29-82a7-7d21ea7f40af`
`The group does not exist`

Комментарий: Требуемый `personGroupId` не существовует в сервисе *Microsoft Face API* ни до, ни после выполнения команды.

Пример вызова команды (AC-007-02)

`$ cat faceapi.json | python -mjson.tool | grep groupId`
`    "groupId": "fintech-01",`
`$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Conte nt-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000 0000000000000000"`
`[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]`
`$ face-management.py --del 27dadf08-bc60-4a29-82a7-7d21ea7f40af`
`Person deleted`

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*. Пользователь с заданным ID удаляется из сервиса.

Пример вызова команды (AC-007-03)
`$ face-management.py --del bdaf190a-4805-4e2c-95af-1afa8b2623df` `The person does not exist`
Комментарий: Пользователь с данным ID не существет в требуемой `personGroupId`.

US-008 Запуск обучения сервиса индентификации

Администратор сервиса может запустить обучение нейронной сети сервиса *Microsoft Face API* для возможности дальнейшего распознавания человека по лицу.

Обучение должно запускаться только если до этого происходило добавление или удаление нового человека.

Использование скрипта
`$ face-management.py --train`

Пример вызова команды (AC-008-01)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01
" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000
0000000000000000000000000000"
{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou
nd.\r\nParameter name: personGroupId"}}
$ face-management.py --train
There is nothing to train
```

Комментарий: Требуемый `personGroupId` не существовует в сервисе *Microsoft Face API* ни до, ни после выполнения команды. Тренировка сервиса не запускается.

Пример вызова команды (AC-008-02)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Conte
nt-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000
0000000000000000"
[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]
$ face-management.py --train
There is nothing to train
```

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*, но в группе нет ни одного добавленного пользователя. Тренировка сервиса не запускается.

Пример вызова команды (AC-008-03)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups" -H "Conte
nt-Type: application/json" -H "OcApim-Subscription-Key: 00000000000000000
0000000000000000"
[{"personGroupId":"fintech-01","name":"fintech-01","userData":null}]
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
```

Комментарий: Требуемый `personGroupId` существует в *Microsoft Face API*. Поскольку запуску команды тренировки сервиса предшествует команда добавления пользователя, то обучение запускается.

Пример вызова команды (АС-008-04)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video6.avi
18 frames extracted
PersonId: 5d5d6e92-f8f0-47cf-8e9b-b4455092603e
FaceIds
=======
0e5edd8e-7756-470e-a26c-f54ab70a5524
73b4caea-c85c-4594-9153-198351937d94
2ae3c850-190a-4cd0-afd2-8efa341767ba
adb7744e-b2fb-4a58-b3a8-f22c3143a8cd
7e0f87d6-14ad-4ec6-971f-e4c2771cc267
47c8c307-1f90-473c-b48c-792d5b5a1241
33e35a22-358a-4847-bcdd-4a5bdfd5eb69
2c7b951b-126a-4eec-9e22-1d9d65ece9e3
bcaf7f6a-1c44-476e-a51f-9dde2744ade0
1c9c0134-ada0-47c2-848c-d4424b232f04
b79945e1-de47-4011-98f2-e7e8f0d9f3ef
f972b6ac-078b-4e3d-a779-4cde8ea28f36
fbeb7aa1-f66e-4412-ad07-b61103e8af0b
9c762e64-075b-4fa4-8e39-0d83df86428b
9794e349-e4e4-44ad-951c-610b4890b16e
2e93da04-8f31-4a27-bd9e-5768335447a5
b9227512-dfa8-4a17-94e2-be520b8975a9
b67c1f86-2e3d-45a2-a7fb-57bbbff9cd8c
$ face-management.py --train
Training successfully started
```

Комментарий: Поскольку запуску команды тренировки сервиса предшествует команда добавления пользователя, то обучение запускается.

Пример вызова команды (АС-008-05)

```
$ face-management.py --del 27dadf08-bc60-4a29-82a7-7d21ea7f40af
Person deleted
$ face-management.py --train
Training successfully started
```

Комментарий: Поскольку запуску команды тренировки сервиса предшествует команда удаления пользователя, то обучение запускается.

Пример вызова команды (АС-008-06)

```
$ face-management.py --train
Training successfully started
$ face-management.py --del bdaf190a-4805-4e2c-95af-1afa8b2623df
The person does not exist
$ face-management.py --train
Already trained
```

Комментарий: Поскольку после предыдущего запуска команды тренировки сервиса изменений в списке пользователей не происходило, то обучение не запускается.

US-009 Обнаружение уже добавленного пользователя

Администратор сервиса не сможет добавить пользователя в сервис *Microsoft Face API*, если изображения лица данного пользователя уже были добавлены в систему,

и эти изображения были использованы для обучения сервиса.

Пример вызова команды (АС-009-01)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
$ face-management.py --simple-add /path/to/video21.avi
The same person already exists.
```

Комментарий: Первое и второе видео содержат кадры лица одного и того же человека. Идентификация человека на втором видео происходит успешно, поскольку пять разных кадров из видео указывают на одного и того же человека с высокой степенью (не менее 50%) уверенности определения. Добавление пользователя в систему не происходит.

Пример вызова команды (АС-009-02)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video6.avi
18 frames extracted
PersonId: 5d5d6e92-f8f0-47cf-8e9b-b4455092603e
FaceIds
=======
0e5edd8e-7756-470e-a26c-f54ab70a5524
73b4caea-c85c-4594-9153-198351937d94
2ae3c850-190a-4cd0-afd2-8efa341767ba
adb7744e-b2fb-4a58-b3a8-f22c3143a8cd
7e0f87d6-14ad-4ec6-971f-e4c2771cc267
47c8c307-1f90-473c-b48c-792d5b5a1241
33e35a22-358a-4847-bcdd-4a5bdfd5eb69
2c7b951b-126a-4eec-9e22-1d9d65ece9e3
bcaf7f6a-1c44-476e-a51f-9dde2744ade0
1c9c0134-ada0-47c2-848c-d4424b232f04
b79945e1-de47-4011-98f2-e7e8f0d9f3ef
f972b6ac-078b-4e3d-a779-4cde8ea28f36
fbeb7aa1-f66e-4412-ad07-b61103e8af0b
9c762e64-075b-4fa4-8e39-0d83df86428b
9794e349-e4e4-44ad-951c-610b4890b16e
2e93da04-8f31-4a27-bd9e-5768335447a5
b9227512-dfa8-4a17-94e2-be520b8975a9
b67c1f86-2e3d-45a2-a7fb-57bbbff9cd8c
$ face-management.py --train
Training successfully started
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video6.avi
The same person already exists.
```

Комментарий: Поскольку для добавления пользователя использовалось одно и то же видео, то повторное добавление пользователя в систему не происходит.

Пример вызова команды (АС-009-03)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
$ face-management.py --simple-add /path/to/video22.avi
5 frames extracted
PersonId: 31e1fc90-84ff-498a-833c-1730aa00a310
FaceIds
=======
21a12afe-6aab-4593-ac77-d20d2bea7e8b
e2f4e1d8-ba2d-4c13-80b7-791f10949143
8c366814-3e8a-441e-86f2-c9edf967c04e
8510de4b-9a70-4d62-823d-471107f838da
c159bd2e-2f71-4c0d-8c85-179d76d96953
```

Комментарий: Первое и второе видео содержат кадры лица разных людей. Человек из второго видео не был добавлен до этого в сервис *Microsoft Face API*. Кадры лица человека из второго видео добавляются в сервис.

Пример вызова команды (AC-009-04)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 52865cde-3af8-443d-b260-9319c2cb1788
FaceIds
=======
cdb6227e-7453-4057-b4fa-79660914e597
6976d3c2-dee5-4f24-8950-f38ff10c70ad
fae15e55-6639-42a4-a954-731c33310e41
15092567-5765-49ed-ac63-94bc5fa08d17
a77f1f0a-aa95-4bd1-9826-6b453aec42b2
$ face-management.py --simple-add /path/to/video31.avi
5 frames extracted
PersonId: 9fa0a99b-8e76-474d-8223-dea217c2c19b
FaceIds
=======
b552ef11-a162-4a7d-9047-ccfc84a07043
90c0815a-ecce-45c6-8107-ced7ef29a249
fde35dba-505d-4a62-ac5a-c6ae4c89128e
6c6910b4-0ab5-4eb4-9e53-95b1929f9867
fdb9d352-65b0-41a2-a1be-03ea5b543160
$ face-management.py --simple-add /path/to/video41.avi
5 frames extracted
PersonId: f290ecb9-bfab-46f7-b623-45140d730628
FaceIds
=======
e5735ecd-ca09-4fd4-bfd3-8ace67702ab0
9e1bbdee-5981-4f6b-aba5-03be57e5e910
3120ef58-8d53-4558-8b84-784ba338f621
8fceb9c7-f029-4326-9703-6749005674fa
8ea02a3b-7dc0-455a-858c-67251b0ca3b4
$ face-management.py --train
Training successfully started
$ face-management.py --simple-add /path/to/video22.avi
5 frames extracted
PersonId: 31e1fc90-84ff-498a-833c-1730aa00a310
FaceIds
=======
21a12afe-6aab-4593-ac77-d20d2bea7e8b
e2f4e1d8-ba2d-4c13-80b7-791f10949143
8c366814-3e8a-441e-86f2-c9edf967c04e
8510de4b-9a70-4d62-823d-471107f838da
c159bd2e-2f71-4c0d-8c85-179d76d96953
```

Комментарий: Даже если обучение сервиса не происходило возможно добавлять несколько разных пользователей в сервис.

Взаимодействие с пользователем

US-010 Идентификация пользователя

Для идентификации пользователя, компонента использует кадры с лицом из

видео-потока и отправляет их в сервис *Microsoft Face API*.

Использование скрипта

```
$ faceid.py --find <path to video file>
```

В случае, когда в текущей директории нет файла `actions.json`, то запускается простой (небезопасный) способ аутентификации. При этом из видеопотока извлекается 5 кадров с изображением лица человека. Подразумевается, что все кадры в видео принадлежат одному и тому же человеку. Идентификация человека происходит успешно, если кадры из видео указывают на одного и того же человека с высокой степенью (не менее 50%) уверенности определения.
Если в видео-потоке недостаточно кадров с изображением человека или на изображении невозможно определить лицо, то обработка такого видео должно приводить к ошибке.
После успешной идентификации в текущей директории создается (если файл уже существовал, то он пересоздается) файл `person.json`, в котором указывается идентификатор, возвращенный сервисом *Microsoft Face API*. Пример файла: `{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}`

Пример вызова команды (AC-010-01)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --find /path/to/video21.avi
37da04e7-f471-49c7-a54c-a08f05950fc5 identified
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
```

Комментарий: Первое и второе видео содержат кадры лица одного и того же человека. Идентификация человека на втором видео происходит успешно, поскольку пять разных кадров из видео указывают на одного и того же человека с высокой степенью (не менее 50%) уверенности определения. В текущей директории создан файл `person.json`

Пример вызова команды (АС-010-02)

```
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
$ face-management.py --simple-add /path/to/video2.avi
5 frames extracted
PersonId: d9e01953-7bc1-4079-96af-3f3a26cf4b1d
FaceIds
=======
3c8195bb-49e7-4b45-b7ae-d44e60310599
d01523ce-cd83-4605-8298-afd4bb8d9e81
c0facdb1-704f-44f2-b76f-4e7298c476be
dcfa8e7b-f8d4-4567-80c5-55f3c5e13d85
a6872fc7-e45b-467f-b9a7-e18935d057da
$ cat person.json
{"id": "33fc4c4a-911a-4ab0-9535-f588d47c3a60"}
$ faceid.py --find /path/to/video21.avi
The service is not ready
$ cat person.json
cat: person.json: No such file or directory
```

Комментарий: Первое и третье видео содержат кадры лица одного и того же человека. Но поскольку после добавления нового пользователя не происходило повторное обучение сервиса, идентификация человека невозможна. Существовавший в текущей директории файл `person.json` удаляется.

Пример вызова команды (АС-010-03)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01
" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000
0000000000000000000000000000"
{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou
nd.\r\nParameter name: personGroupId"}}
$ faceid.py --find /path/to/video21.avi
The service is not ready
$ cat person.json
cat: person.json: No such file or directory
```

Комментарий: Поскольку не существует группы на момент запуска команды идентификации пользователя, то идентификация человека невозможна. Файл `person.json` не создается.

Пример вызова команды (AC-010-04)

```
$ face-management.py --add /path/to/video1.avi /path/to/video2.avi /path/
to/video3.avi /path/to/video4.avi /path/to/video5.avi
18 frames extracted
PersonId: 5d5d6e92-f8f0-47cf-8e9b-b4455092603e
FaceIds
=======
0e5edd8e-7756-470e-a26c-f54ab70a5524
73b4caea-c85c-4594-9153-198351937d94
2ae3c850-190a-4cd0-afd2-8efa341767ba
adb7744e-b2fb-4a58-b3a8-f22c3143a8cd
7e0f87d6-14ad-4ec6-971f-e4c2771cc267
47c8c307-1f90-473c-b48c-792d5b5a1241
33e35a22-358a-4847-bcdd-4a5bdfd5eb69
2c7b951b-126a-4eec-9e22-1d9d65ece9e3
bcaf7f6a-1c44-476e-a51f-9dde2744ade0
1c9c0134-ada0-47c2-848c-d4424b232f04
b79945e1-de47-4011-98f2-e7e8f0d9f3ef
f972b6ac-078b-4e3d-a779-4cde8ea28f36
fbeb7aa1-f66e-4412-ad07-b61103e8af0b
9c762e64-075b-4fa4-8e39-0d83df86428b
9794e349-e4e4-44ad-951c-610b4890b16e
2e93da04-8f31-4a27-bd9e-5768335447a5
b9227512-dfa8-4a17-94e2-be520b8975a9
b67c1f86-2e3d-45a2-a7fb-57bbbff9cd8c
$ face-management.py --train
Training successfully started
$ faceid.py --find /path/to/video22.avi
The person was not found
$ cat person.json
cat: person.json: No such file or directory
```

Комментарий: Видео, использовавшееся при добавлении человека, и видео, использующееся при распознавании, содержат кадры лица разных людей. Человек, чье лицо изображено на видео, использующееся при распознновании, не был добавлен до этого в сервис *Microsoft Face API*. Кадры лица человека из второго видео не добавляются в сервис *Microsoft Face API*. Файл `person.json` не создается.

Пример вызова команды (AC-010-05)

```
$ cat faceapi.json | python -mjson.tool | grep groupId
    "groupId": "fintech-01",
$ curl -X GET "https://<datacenter url>/face/v1.0/persongroups/fintech-01
" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 0000
0000000000000000000000000000"
{"error":{"code":"PersonGroupNotFound","message":"Person group is not fou
nd.\r\nParameter name: personGroupId"}}
$ faceid.py --find /path/to/video100.avi
The video does not follow requirements
$ cat person.json
cat: person.json: No such file or directory
$ face-management.py --simple-add /path/to/video1.avi
5 frames extracted
PersonId: 37da04e7-f471-49c7-a54c-a08f05950fc5
FaceIds
=======
1d499868-3d01-487c-8bab-626dc562e4e8
27dadf08-bc60-4a29-82a7-7d21ea7f40af
b8cf9c2f-a606-4f21-851d-26e0a0dc8a74
bf4806de-8c4b-4a12-8495-002f43dba797
ff79486f-15ac-43be-9c6c-b2840f8c8d22
$ face-management.py --train
Training successfully started
$ faceid.py --find /path/to/video100.avi
The video does not follow requirements
$ cat person.json
cat: person.json: No such file or directory
```

Комментарий: Видео, использующееся при распознавании, либо содержит меньше 5 кадров, либо не содержит кадров с лицом пользователя. Файл `person.json` не создается.

Пример вызова команды (AC-010-06)

```
$ faceid.py --find /path/to/video101.avi
The person was not found
$ cat person.json
cat: person.json: No such file or directory
```

Комментарий: На видео, использующееся при распознавании, как минимум один из пяти разных кадров указывает на человека, отличающегося от человека в других кадрах. Степень уверенности определения не менее 50%. Файл `person.json` не создается.

US-011 Запроса действий на безопасную идентификацию пользователя

Для обеспечения безопасной идентфикации сервис может сформировать запрос к пользователю на выполнение определенных действий. Совершив данные действия пользователь продемонстрирует, что является живым человеком. Таким образом достигается дополнительная безопасность.

Использование скрипта
`$ faceid.py --actions`
Команда генерирует JSON `actions.json`, содержащий описание набора действий, которые должен совершить пользователь. Действия выбираются из следующего списка: • YawRight - поворот головы направо на 15 градусов • YawLeft - поворот головы налево на 15 градусов • RollRight - наклон головы вправо на 20 градусов • RollLeft - наклон головы влево на 20 градусов • CloseRightEye - прищурить правый глаз на 2 секунды • CloseLeftEye - прищурить левый глаз на 2 секунды • OpenMouth - открыть широко рот В сформированных действиях не должно быть 2 поворота и/или наклона. Всего действий должно быть 3 или 4.

Пример вызова команды (AC-011-01)

`$ faceid.py --actions`
`$ cat actions.json`
`{"actions": ["OpenMouth", "YawLeft", "CloseLeftEye"]}`

Комментарий: Сформирован файл `actions.json` с тремя действиями. В сформированных действиях - один запрос на открытие рта, один запрос на закрытие глаза, один запрос на изменение положения головы.

Пример вызова команды (AC-011-02)

`$ faceid.py --actions`
`$ cat actions.json`
`{"actions": ["CloseRightEye", "RollRight", "OpenMouth", "CloseLeftEye"]}`

Комментарий: Сформирован файл `actions.json` с тремя действиями. В сформированных действиях - один запрос на открытие рта, два запроса (разные глаза) на закрытие глаза, один запрос на изменение положения головы.

Пример вызова команды (AC-011-03)

`$ faceid.py --actions`
`$ cat actions.json`
`{"actions": ["CloseLeftEye", "CloseRightEye", "RollRight"]}`

Комментарий: Сформирован файл `actions.json` с тремя действиями. В сформированных действиях - два запроса на закрытие глаз (разные глаза), один запрос на изменение положения головы.

US-012 Безопасная идентификация пользователя

Должен быть action JSON, сохраняет PersonId в person JSON.

Использование скрипта
`$ faceid.py --find /path/to/video.avi`

US-013 Получение баланса идентифицированного пользователя

После идентификации пользователь может получить баланс на соответствующем

данному пользователю аккаунте в сети блокчейн.

Использование скрипта

```
$ faceid.py <PIN code>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется запрос на баланс аккаунта, полученного из приватного ключа.
Приватный ключ генерируется по правилу:

$$K = keccak256(keccak256(keccak256(keccak256(keccak256(''), I, P_1), I, P_2), I, P_3), I, P_4)$$

где '' - 'пустая' последовательность байт, I - это идентификатор, который возвращает система распознавания по лицу, приведенный к длине в 16 байт, а (P_1, P_2, P_3, P_4) - четыре цифры PIN-кода, где каждая цифра представлена целым числом длиной 1 байт, P_1 - цифра самого старшего разряда в PIN-коде (первая цифра), а P_4 - цифра самого младшего разряда в PIN-коде (последняя цифра).
Например, идентификатору `37da04e7-f471-49c7-a54c-a08f05950fc5` при применении PIN-кода `1234` соответствует приватный ключ `6be7217f318a6409ba8e87e42ce600e14153647a816f7f2d43d244d1a00ed3df`.
При выводе баланса должно быть автомтическое масштабирование суммы - нормализация к одному из возможных значений: *poa*, *finney*, *szabo*, *gwei*, *mwei*, *kwei*, *wei*. Происходит следующим образом:

- Используется минимально возможное нормализованное значение, чья целая часть больше ноля;
- Дробная часть записывается с округлением до 10^{-6};
- Завершающие нули в дробной части, полученные полсе округления, не записываются (например, 1.3, но не 1.300000).

Пример вызова команды (AC-013-01)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance 4590
Your balance is 2.5 poa
```

Комментарий: Баланс пользователя получен от узла блокчейн сети, доступ к которому получен через JSON RPC.

Пример вызова команды (AC-013-02)

```
$ cat person.json
{"id": "a9d0f1d5-1359-43ca-bcc3-cc7c1e314b86"}
$ faceid.py --balance 1864
Your balance is 84.000138 szabo
```

Комментарий: Баланс пользователя получен от узла блокчейн сети, доступ к которому получен через JSON RPC.

Пример вызова команды (AC-013-03)

`$ cat person.json`
`{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}`
`$ faceid.py --balance 6879`
`Your balance is 0 poa`

Комментарий: Неправильно задан PIN-код, поэтому баланс запрашивается у несуществующего аккаунта.

Пример вызова команды (AC-013-04)

`$ cat person.json`
`cat: person.json: No such file or directory`
`$ faceid.py --balance 6879`
`ID is not found`

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`.

US-014 Отправка запроса на регистрацию соответствия

После идентификации пользователь может отправить запрос на регистрацию соответствия между телефоном и своим аккаунтом.

Использование скрипта
`$ faceid.py --add <pin code> <phone number>`
Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется транзакция к контракту регистра соответствий с запросом регистрации. В терминал выводится хэш транзакции, в рамках которой в контракт добавлен запрос регистрации.

Пример вызова команды (AC-014-01)

`$ cat network.json`
`{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.poa.network/", "defaultGasPrice": 2000000000}`
`$ cat person.json`
`{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}`
`$ faceid.py --add 4590 +79991234567`
`Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede84bca5e91517e8e5bf1cc69`

Комментарий: В блокчейн сеть отправляется транзакция с запросом регистрации соответствия аккаунта, ассоциированного с аккаунтом, указанным в файле `person.json` и номером телефона, указанным в параметре командной строки. При обработке запроса, контракт производить событие (`event`) `RegistrationRequest`, в с указанием аккаунта, отправившего запрос: `event RegistrationRequest(address indexed sender);` Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-014-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ faceid.py --add 4590 +79991234567
Registration request already sent
$ faceid.py --add 4590 +79991239812
Registration request already sent
```

Комментарий: Повторный запрос на регистрацию соответствия аккаунта не отправляется в блокчейн сеть, поскольку данный аккаунт уже посылал запрос на регистрацию соответствий. Первичный запрос еще не был обработан. В терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-014-03)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --add 4590 +79991234567
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-014-04)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 1234 +79991234567
No funds to send the request
```

Комментарий: Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда на аккаунте нет средств для оплаты комиссии на обработку транзакции, в терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-014-05)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --add 4590 +79122229016
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-014-06)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ faceid.py --add 4590 +79122229016
Seems that the contract address is not the registrar contract
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

Пример вызова команды (AC-014-07)

```
$ faceid.py --add 4590 +79122
Incorrect phone number
$ faceid.py --add 4590 adad
Incorrect phone number
$ faceid.py --add 4590 +73431231543543534653
Incorrect phone number
$ faceid.py --add 4590
Incorrect phone number
```

Комментарий: Выдается сообщение об ошибке, если номер телефона указан некорректно. Корректный номер содержит 11 цифр и начинается со знака +.

Пример вызова команды (AC-014-08)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ faceid.py --find /path/to/video23.avi
168f2e61-089f-88a9-53b2-3b3d0c497704 identified
$ faceid.py --add 5981 +79037518950
Registration request sent by 0x6a8a46c4e005b9e8bea97aa58c5839787b5689e9e8
785ea9e38330395582f78f
```

Комментарий: В блокчейн сеть отправляется транзакция с запросом регистрации соответствия аккаунта даже если в контракте регистра уже учтены запросы на регистрацию соответствий других аккаунтов. При обработке запроса, контракт производит событие (`event`) `RegistrationRequest`.

Пример вызова команды (AC-014-09)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ faceid.py --add 5981 +79037518950
Registration request sent by 0x6a8a46c4e005b9e8bea97aa58c5839787b5689e9e8
785ea9e38330395582f78f
```

Комментарий: В блокчейн сеть отправляется транзакция с запросом регистрации соответствия аккаунта. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-014-10)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x73aad4ff595a8813cc7d440d244545017c77098528
f010a7caaa7d74c382f6c5
```

Комментарий: Если из транзакции с идентификатором `0x73aad4ff595a8813cc7d440d244545017c77098528f010a7caaa7d74c382f6c5` извлечь поле `input` и отправить его в новой транзакции с того же аккаунта снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка, поскольку такой запрос регистрации соответствия уже был послан в контракт. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-014-11)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
```

Комментарий: Если из транзакции с идентификатором `0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede84bca5e91517e8e5bf1cc69` извлечь поле `input`, изменить в нем те байты, которые кодируют номер телефона так, чтобы передаваемый номер телефона содержал количество цифр отличное от 11, либо содержал буквы, и отправить получившийся набор байт в новой транзакции с того же аккаунта снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка, поскольку такой номер телефона некорректный. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-014-12)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x73aad4ff595a8813cc7d440d244545017c77098528
f010a7caaa7d74c382f6c5
```

Комментарий: Если из транзакции с идентификатором `0x73aad4ff595a8813cc7d440d244545017c77098528f010a7caaa7d74c382f6c5` извлечь поле `input` и отправить его в новой транзакции с аккаунта, отличающегося от `from` в упомянутой выше транзакции, на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - успешно, поскольку такой изначальный запрос регистрации соответствия еще не был подтвержден. Статус транзакции можно подтвердить для данной транзакции в браузере блоков.

US-015 Отправка запроса на удаление соответствия

После идентификации пользователь может отправить запрос на удаление соответствия между телефоном и своим аккаунтом.

Использование скрипта

```
$ faceid.py --del <pin code>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется транзакция к контракту регистра соответствий с запросом удаления. В терминал выводится хэш транзакции, в рамках которой в контракт добавлен запрос удаления.

Пример вызова команды (AC-015-01)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
```

Комментарий: В блокчейн сеть отправляется транзакция с запросом удаления ранее зарегистрированного соответствия аккаунта, ассоциированного с аккаунтом, указанным в файле `person.json` и номера телефона. При обработке запроса, контракт производить событие (`event`) `UnregistrationRequest`, в с указанием аккаунта, отправившего запрос:
`event UnregistrationRequest(address indexed sender);`
Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-015-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
$ faceid.py --del 4590
Unregistration request already sent
```

Комментарий: Повторный запрос на удаление соответствия аккаунта не отправляется в блокчейн сеть, поскольку данный аккаунт уже посылал запрос на удаление соответствий. Первичный запрос еще не был обработан. В терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-015-03)

```
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79220012534
Registration request sent by 0xc95d677eb6f3fcb55e08274ae0eed1970391e637f1
062426a3406fd7d4cfcfcb
$ faceid.py --del 6104
Account is not registered yet
```

Комментарий: Запрос на удаление соответствия аккаунта не отправляется в блокчейн сеть, поскольку данный аккаунт не зарегистрирован в регистре соответствий. В терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-015-04)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --del 4590
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-015-05)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 1234
No funds to send the request
```

Комментарий: Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда на аккаунте нет средств для оплаты комиссии на обработку транзакции, в терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-015-06)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --del 4590
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-015-07)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ faceid.py --del 4590
Seems that the contract address is not the registrar contract.
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

Пример вызова команды (AC-015-08)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
$ faceid.py --find /path/to/video23.avi
168f2e61-089f-88a9-53b2-3b3d0c497704 identified
$ faceid.py --del 5981
Unregistration request sent by 0xda0e2a124e6d37080b539f1bb0dc4c698b8025b6
2dcc56d75efc1115d87381aa
```

Комментарий: В блокчейн сеть отправляется транзакция с запросом удаления ранее зарегистрированного соответствия аккаунта даже если в контракте регистра уже учтены запросы на удаление соответствий других аккаунтов. При обработке запроса, контракт производить событие (`event`) `UnregistrationRequest`.

Пример вызова команды (AC-015-09)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ faceid.py --del 5981
Unregistration request sent by 0xda0e2a124e6d37080b539f1bb0dc4c698b8025b6
2dcc56d75efc1115d87381aa
```

Комментарий: В блокчейн сеть отправляется транзакция с запросом удаления соответствия аккаунта. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-015-10)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
```

Комментарий: Если из транзакции с идентификатором `0x58aff6987a78c9109ff3b99b0e1cf333faba313bc8002d4b333653be45d7e1d8` извлечь поле `input` и отправить его в новой транзакции с того же аккаунта снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка, поскольку такой запрос удаления соответствия уже был послан в контракт. Статус можно подтвердить для данной транзакции в браузере блоков.

US-016 Отмена запроса на регистрацию или удаление соответствия

После идентификации пользователь может отправить отмену для запроса на добавление или удаление соответствия между телефоном и своим аккаунтом.

Использование скрипта

```
$ faceid.py --cancel <pin code>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется транзакция к контракту регистра соответствий на отмену запроса добавления или удаления соответствия. В терминал выводится хэш транзакции, в рамках которой просиходит отмена.

Пример вызова команды (AC-016-01)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ faceid.py --cancel 4590
Registration canceled by 0x99a7e5f28b653d63b5c0bbddebc91678530ee461e4fa21
c9ab2f281d28b8da5e
```

Комментарий: В блокчейн сеть отправляется транзакция с отменой запроса регистрации соответствия аккаунта, ассоциированного с аккаунтом, указанным в файле `person.json`. При обработке запроса, контракт производить событие (`event`) `RegistrationCanceled`, в с указанием аккаунта, отправившего запрос:

```
event RegistrationCanceled(address indexed sender);
```

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-016-02)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x20d2ac2a28641786ba03eff53facea687b35e08e68
0a1b71922c6fc1ed1f2735
$ faceid.py --cancel 4590
Registration canceled by 0x566674f71911cd3b7cbc2fd66353711d31aaa96f269440
d186374a017f53f324
```

Комментарий:

Пример вызова команды (AC-016-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
$ faceid.py --cancel 4590
Unregistration canceled by 0x84a9548edfa9ce5d05bb7c88702873196f7537e06061
9d4238e7f23ba8dfe3bd
```

Комментарий: В блокчейн сеть отправляется транзакция с отменой запроса удаления ранее зарегистрированного соответствия аккаунта, ассоциированного с аккаунтом, указанным в файле `person.json`. При обработке запроса, контракт производить событие (`event`) `UnregistrationCanceled`, в с указанием аккаунта, отправившего запрос:

```
event UnregistrationCanceled(address indexed sender);
```

Пример вызова команды (AC-016-04)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0xbf08d8a2d7c7780024ba6e6dbe12d3d5a5a747ea
3071fca63d8a2ede50d3472a
$ faceid.py --cancel 4590
Registration canceled by 0x2946d6e84f7bc52619746d71da01ef04a31855a8deb6db
ea29310ab3b0f6201e
```

Комментарий:

Пример вызова команды (AC-016-05)

```
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ faceid.py --cancel 4590
Registration canceled by 0x99a7e5f28b653d63b5c0bbddebc91678530ee461e4fa21
c9ab2f281d28b8da5e
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x4ae610dabaaddd16fe0097641e0e78b5423f891083
bb81266b228a49db54e2ff
```

Комментарий: После отмены запроса регистрации соответствия, в блокчейн сеть снова можно отправить запрос на регистрацию соответствия для того же аккаунта.

Пример вызова команды (AC-016-06)

```
$ faceid.py --del 4590
Unregistration request sent by 0x58aff6987a78c9109ff3b99b0e1cf333faba313b
c8002d4b333653be45d7e1d8
$ faceid.py --cancel 4590
Unregistration canceled by 0x84a9548edfa9ce5d05bb7c88702873196f7537e06061
9d4238e7f23ba8dfe3bd
$ faceid.py --del 4590
Unregistration request sent by 0x03a8233da62730a9e5ac69d78c3fc3cf02b04ecb
995f5a9f9eb3b17c01692824
```

Комментарий: После отмены запроса на удаление соответствия, в блокчейн сеть снова можно отправить отмену запроса удаления ранее зарегистрированного соответствия аккаунта.

Пример вызова команды (AC-016-07)

```
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x4ae610dabaaddd16fe0097641e0e78b5423f891083
bb81266b228a49db54e2ff
$ faceid.py --cancel 4590
Registration canceled by 0x3d61f45ffa5ac772fc53d36fcc89bacaee425152d65c7f
d9a506ba40aa0ed6e1
$ faceid.py --cancel 4590
No requests found
```

Комментарий: Выдается ошибка, если нет активных запросов на регистрацию или удаления соответствия.

Пример вызова команды (AC-016-08)

```
$ faceid.py --del 4590
Unregistration request sent by 0x03a8233da62730a9e5ac69d78c3fc3cf02b04ecb
995f5a9f9eb3b17c01692824
$ faceid.py --cancel 4590
Unregistration canceled by 0x41c2cd6cce43fddcd3c1489e1b2f40ad13b635bb0fce
22f2ea66a97f15b6c84b
$ faceid.py --cancel 4590
No requests found
```

Комментарий: Выдается ошибка, если нет активных запросов на регистрацию или удаления соответствия.

Пример вызова команды (AC-016-09)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --cancel 4590
Registration canceled by 0x41c2cd6cce43fddcd3c1489e1b2f40ad13b635bb0fce22
f2ea66a97f15b6c84b
$ faceid.py --find /path/to/video23.avi
168f2e61-089f-88a9-53b2-3b3d0c497704 identified
$ faceid.py --cancel 5981
No requests found
```

Комментарий: Выдается ошибка, если нет активных запросов на регистрацию или удаления соответствия.

Пример вызова команды (AC-016-10)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --cancel 4590
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-016-11)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --cancel 1234
No funds to send the request
```

Комментарий: Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда на аккаунте нет средств для оплаты комиссии на обработку транзакции, в терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-016-12)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --cancel 4590
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-016-13)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ faceid.py --cancel 4590
Seems that the contract address is not the registrar contract.
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

Пример вызова команды (AC-016-14)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ faceid.py --cancel 4590
Registration canceled by 0x41c2cd6cce43fddcd3c1489e1b2f40ad13b635bb0fce22
f2ea66a97f15b6c84b
```

Комментарий: В блокчейн сеть отправляется транзакция с отменой запросом регистрации или удаления соответствия аккаунта. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

US-017 Отправка средств

После идентификации пользователь может отправить часть средств, которые числятся на его балансе другому пользователю системы, указав его номер телефона.

Использование скрипта

```
$ faceid.py --send <pin code> <phone number> <value>
```

Через RPC узел блокчейн сети происходит обращение к контракту регистрации соответствий на получение аккаунта соответствующего данному номеру телефона. После этого используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя, в блокчейн сеть отправляется транзакция на перевод средств с баланса пользователя на аккаунт пользователя, ассоциированного с номером телефона.

Пример вызова команды (AC-017-01)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance 1234
Your balance is 500 finney
$ faceid.py --send 1234 +79873344556 10000000000000000
Payment of 10 finney to +79873344556 scheduled
Transaction Hash: 0x27c9181caeb55d37e1105fa1a8648db7fe50f79064b98e56b8e85
4e3abb43728
$ faceid.py --balance 1234
Your balance is 489.860605 finney
```

Комментарий: Средства успешно доставлены.

Пример вызова команды (AC-017-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance 1234
Your balance is 90 finney
$ faceid.py --send 1234 +79873344556 10000000000000000
No funds to send the payment
$ faceid.py --balance 1234
Your balance is 90 finney
```

Комментарий: Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-017-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --send 1234 +79873312356 10000000000000000
No account with the phone number +79873312356
```

Комментарий: Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-017-04)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --send 1114 +79873312356 10000000000000000
No funds to send the payment
```

Комментарий: Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда на аккаунте нет средств для перевода, в терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-017-05)

```
$ faceid.py --send 1234 +79122 10000000000000000
Incorrect phone number
$ faceid.py --send 1234 adad 10000000000000000
Incorrect phone number
$ faceid.py --send 1234 +73431231543543534653 10000000000000000
Incorrect phone number
```

Комментарий: Выдается сообщение об ошибке, если номер телефона указан некорректно. Корректный номер содержит 11 цифр и начинается со знака +.

US-018 Генерация сертификата на получение средств

После идентификации пользователь может отправить запрос на создание сертификата на получение определенного количества средств. Созданный сертификат впоследствии может быть использован любым пользователем до истечения срока действия. Сертификат имеет следующий формат: первые 32 байта - цифровой идентификатор сертификата, следующие 65 байт - цифровая подпись цифрового идентификатора (`r`, `s`, `v`).

Использование скрипта

```
$ faceid.py --gift <pin code> <value> <expire date>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя, в блокчейн сеть отправляется транзакция к контракту управления сертификатами на создание нового сертификата на указанную сумму в `wei`, действующий до указанной даты в формате `HH:MM DD.MM.YYYY`. В терминал выводится созданный цифровой сертификат.

Пример вызова команды (AC-018-01)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 500 finney
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
aee79c36a8aff107f836a382d175b2e7cd86c34dae48a3288eb38b72da955d609e1c1f49a
a566e305bf444120af2f65923315026b0a03bcde4b139571752c0421a368cc1dc7171c6e8
08ba1fcb4cd7f3c034c64853dbd9a91bfc12ef9eece1e21c
$ faceid.py --balance
Your balance is 489.860605 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на создание сертификата. При обработке запроса, контракт производит событие (`event`) `CertificateCreated` с указанием идентификатора сертификата внутри контракта:

```
event CertificateCreated(bytes32 indexed id);
```

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителя уменьшился на сумму указанную в сертификате.

Пример вызова команды (AC-018-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 90 finney
$ faceid.py --gift 4590 100000000000000000 "15:40 08.03.2019"
No funds to create a certificate
$ faceid.py --balance
Your balance is 90 finney
```

Комментарий: На балансе пользователя недостаточно средств чтобы создать сертификат с указанной суммой. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не уменьшается. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-018-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ date
Wed Oct  21 07:28:00 MSK 2015
$ faceid.py --gift 4590 1000 "09:00 26.10.1985"
Expiration date is invalid
```

Комментарий: Указанная дата годности сертификата уже истекла. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не уменьшается. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-018-04)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --gift 1234 1000 "15:40 08.03.2019"
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-018-05)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --gift 1234 1000 "15:40 08.03.2019"
No funds to create a certificate
```

Комментарий: Указан неверный пинкод. Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда на аккаунте нет средств для оплаты транзакции, в терминал выводится сообщение об ошибке. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-018-06)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 485.992209 finney
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a907cab99926f4af694a7053ac2253
149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59ff8fbcf6d86a8d179e6e74d103f9
e7e423630454b06ffe5dd2806e3bc0551971675619b31a1b
$ faceid.py --balance
Your balance is 475.852814 poa
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
73bd8c3b60a59a3ccb37764f87e62dafab374d3c4885edd21192f060b83a47e616dcc4fd2
2eac78fdc3e023fa60ac7863a4111f245556b3966b2d960b104c53f5d7a2c67b787abdd86
2826708737d425a5d0e774d055a996cc853591113afc881c
$ faceid.py --balance
Your balance is 465.713419 poa
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --balance
Your balance is 1.5 poa
$ faceid.py --gift 6104 10000000000000000 "15:40 08.03.2019"
b6f4176f9fe77211717fb3f7f41995b29cc04bd888d24eaa8d398fd57b8fc2153c352e75a
dbbb101570a58056ae00a0c6fb4c176f7e91f3489ae4bc1104f857e551bd96f1f7223ad9e
735f1544cc0eb89f2ed703dc4243cd1a2d96a47b91d0f41c
$ faceid.py --balance
Your balance is 1.48983 poa
```

Комментарий: В блокчейн сеть отправляется несколько транзакции на создание сертификатов с нескольких аккаунтов. Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителей уменьшился на суммы указанные в сертификатах.

Пример вызова команды (AC-018-07)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 500 finney
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
c265d15e7a89cab7a9e30a3287d0be6d8b2ab3e635ede29129514002b5bf4cdf7f880a32d
623da84c78d89b7814d5a0d41fc249d55902c7835c0cd2fa978cd077b4483d7d828a63d1c
bec57e5ebc65fb262c7d3c2fc3f3c7afc0b599fee1371b1c
$ faceid.py --balance
Your balance is 489.860605 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на создание сертификата. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-018-08)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта управления сертификатами. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-018-09)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
Seems that the contract address is not the certificates contract.
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту управления сертификатами.

US-019 Использование сертификата на получение средств

После идентификации пользователь может отправить запрос на получение средств с уже созданного сертификата. Сертификат может быть использован только один раз и только до истечения указанного при создании срока годности.

Использование скрипта

```
$ faceid.py --receive <pin code> <certificate>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется транзакция к контракту управления сертификатами на получение средств с сертификата.

Пример вызова команды (AC-019-01)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 500 finney
$ faceid.py --receive 4590 aee79c36a8aff107f836a382d175b2e7cd86c34dae48a3
288eb38b72da955d609e1c1f49aa566e305bf444120af2f65923315026b0a03bcde4b1395
71752c0421a368cc1dc7171c6e808ba1fcb4cd7f3c034c64853dbd9a91bfc12ef9eece1e2
1c
Received funds from the certificate
$ faceid.py --balance
Your balance is 510 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на получение средств. При обработке запроса, контракт производит событие (`event`) `CertificateUsed` с указанием идентификатора сертификата внутри контракта:

```
event CertificateUsed(bytes32 indexed id);
```

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителя увеличился на сумму указанную в сертификате.

Пример вызова команды (AC-019-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 510 finney
$ faceid.py --receive 4590 aee79c36a8aff107f836a382d175b2e7cd86c34dae48a3
288eb38b72da955d609e1c1f49aa566e305bf444120af2f65923315026b0a03bcde4b1395
71752c0421a368cc1dc7171c6e808ba1fcb4cd7f3c034c64853dbd9a91bfc12ef9eece1e2
1c
Cannot receive funds from the certificate
$ faceid.py --balance
Your balance is 510 finney
```

Комментарий: Нельзя использовать сертификат, так как он уже был использован ранее. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-019-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --receive 4590 e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a90
7cab99926f4af694a7053ac2253149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59f
f8fbcf6d86a8d179e6e74d103f9e7e423630454b06ffe5dd2806e3bc0551971675619b31a
1b
Cannot receive funds from the certificate
```

Комментарий: Нельзя использовать сертификат, так как указанная дата годности сертификата уже истекла. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-019-04)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --receive 1234 e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a90
7cab99926f4af694a7053ac2253149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59f
f8fbcf6d86a8d179e6e74d103f9e7e423630454b06ffe5dd2806e3bc0551971675619b31a
1b
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-019-05)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --receive 4590 0123456789abcdef0123456789abcdef0123456789abcd
ef0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456
789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
01
Cannot receive funds from the certificate
```

Комментарий: Указан несуществующий сертификат, который невозможно использовать. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-019-06)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --gift 4590 1000000000000000 "15:40 08.03.2019"
e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a907cab99926f4af694a7053ac2253
149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59ff8fbcf6d86a8d179e6e74d103f9
e7e423630454b06ffe5dd2806e3bc0551971675619b31a1b
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
73bd8c3b60a59a3ccb37764f87e62dafab374d3c4885edd21192f060b83a47e616dcc4fd2
2eac78fdc3e023fa60ac7863a4111f245556b3966b2d960b104c53f5d7a2c67b787abdd86
2826708737d425a5d0e774d055a996cc853591113afc881c
$ faceid.py --balance
Your balance is 465.713419 poa
$ faceid.py --receive 4590 73bd8c3b60a59a3ccb37764f87e62dafab374d3c4885ed
d21192f060b83a47e616dcc4fd22eac78fdc3e023fa60ac7863a4111f245556b3966b2d96
0b104c53f5d7a2c67b787abdd862826708737d425a5d0e774d055a996cc853591113afc88
1c
Received funds from the certificate
$ faceid.py --balance
Your balance is 475.713419 poa
$ faceid.py --receive 4590 e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a90
7cab99926f4af694a7053ac2253149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59f
f8fbcf6d86a8d179e6e74d103f9e7e423630454b06ffe5dd2806e3bc0551971675619b31a
1b
Received funds from the certificate
$ faceid.py --balance
Your balance is 476.713419 poa
```

Комментарий: В блокчейн сеть отправляется несколько транзакции на создание сертификатов с нескольких аккаунтов, а затем несколько транзакций на получение средств. Для проведения транзакций выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителей корректно изменяется на суммы указанные в сертификатах.

Пример вызова команды (AC-019-07)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 489.860605 finney
$ faceid.py --receive 4590 c265d15e7a89cab7a9e30a3287d0be6d8b2ab3e635ede2
9129514002b5bf4cdf7f880a32d623da84c78d89b7814d5a0d41fc249d55902c7835c0cd2
fa978cd077b4483d7d828a63d1cbec57e5ebc65fb262c7d3c2fc3f3c7afc0b599fee1371b
1c
Received funds from the certificate
$ faceid.py --balance
Your balance is 499.860605 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на получение средств с сертификата. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-019-08)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --receive 4590 c265d15e7a89cab7a9e30a3287d0be6d8b2ab3e635ede2
9129514002b5bf4cdf7f880a32d623da84c78d89b7814d5a0d41fc249d55902c7835c0cd2
fa978cd077b4483d7d828a63d1cbec57e5ebc65fb262c7d3c2fc3f3c7afc0b599fee1371b
1c
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта управления контрактами. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-019-09)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ faceid.py --receive 4590 c265d15e7a89cab7a9e30a3287d0be6d8b2ab3e635ede2
9129514002b5bf4cdf7f880a32d623da84c78d89b7814d5a0d41fc249d55902c7835c0cd2
fa978cd077b4483d7d828a63d1cbec57e5ebc65fb262c7d3c2fc3f3c7afc0b599fee1371b
1c
Seems that the contract address is not the certificates contract.
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту управления сертификатами.

Пример вызова команды (AC-019-10)

```
$ faceid.py --balance
Your balance is 500 finney
$ faceid.py --receive 4590 aee79c36a8aff107f836a382d175b2e7cd86c34dae48a3
288eb38b72da955d609e1c1f49aa566e305bf444120af2f65923315026b0a03bcde4b1395
71752c0421a368cc1dc7171c6e808ba1fcb4cd7f3c034c64853dbd9a91bfc12ef9eece1e2
1c
Received funds from the certificate
$ faceid.py --balance
Your balance is 510 finney
```

Комментарий: Если из транзакции, которая была отправлена в результате команды `faceid.py -receive`, извлечь поле input и отправить его в новой транзакции снова в поле input на адрес контракта управления сертификатами, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка по причине того, что сертификат уже был использован ранее. Следовательно, повторно сертификат не должен быть использован.

US-020 Вернуть средства из неиспользованных сертификатов

После идентификации пользователь может отправить запрос на возврат средств с уже созданного сертификата после истечения срока годности. Возврат может быть осуществлен только один раз и только с аккаунта создателя сертификата.

Использование скрипта

```
$ faceid.py --withdraw <pin code>
```

Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется транзакция к контракту управления сертификатами на возврат средств с сертификата.

Пример вызова команды (AC-020-01)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 500 finney
$ faceid.py --withdraw 4590
Withdrew funds from the certificate
$ faceid.py --balance
Your balance is 510 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на возврат средств. При обработке запроса, контракт производит событие (`event`) `CertificateWithdrew` с указанием идентификатора сертификата внутри контракта:

```
event CertificateWithdrew(bytes32 indexed id);
```

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителя увеличился на сумму указанную в сертификате.

Пример вызова команды (AC-020-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 510 finney
$ faceid.py --withdraw 4590
Cannot withdraw funds from the certificate
$ faceid.py --balance
Your balance is 510 finney
```

Комментарий: Нельзя вернуть средства, так как аналогичный запрос уже был выполнен. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-020-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --withdraw 4590
Cannot withdraw funds from the certificate
```

Комментарий: Нельзя вернуть средства, так как указанная дата годности сертификата еще не истекла. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-020-04)

```
$ cat person.json
cat: person.json: No such file or directory
$ faceid.py --withdraw 1234
ID is not found
```

Комментарий: Выдается ошибка, если в текущей директории не существует файл `person.json`. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-020-05)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --withdraw 4590
Cannot withdraw funds from the certificate
```

Комментарий: Указан несуществующий сертификат, который невозможно использовать. Транзакция в блокчейн сеть не отправляется. Баланс пользователя не изменяется. В терминал выводится сообщение об ошибке.

Пример вызова команды (AC-020-06)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --gift 4590 1000000000000000 "15:40 08.03.2019"
e4473f3c3e3f3803f5c640a93b44b9a35968df4e7d4a907cab99926f4af694a7053ac2253
149e7a5212f816cd367a471acc97c67cf38b7b3cf0d59ff8fbcf6d86a8d179e6e74d103f9
e7e423630454b06ffe5dd2806e3bc0551971675619b31a1b
$ faceid.py --gift 4590 10000000000000000 "15:40 08.03.2019"
73bd8c3b60a59a3ccb37764f87e62dafab374d3c4885edd21192f060b83a47e616dcc4fd2
2eac78fdc3e023fa60ac7863a4111f245556b3966b2d960b104c53f5d7a2c67b787abdd86
2826708737d425a5d0e774d055a996cc853591113afc881c
$ faceid.py --balance
Your balance is 465.713419 poa
$ faceid.py --withdraw 4590
Withdrew funds from the certificate
$ faceid.py --balance
Your balance is 476.713419 poa
```

Комментарий: В блокчейн сеть отправляется несколько транзакции на создание сертификатов с нескольких аккаунтов, а затем транзакция на возврат средств. Для проведения транзакций выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`. Баланс отправителей корректно изменяется на суммы указанные в сертификатах.

Пример вызова команды (AC-020-07)

```
$ cat network.json | python -mjson.tool | grep gasPriceUrl
    "gasPriceUrl": "https://gasprice.poa.network/",
$ curl https://gasprice.poa.network/
curl: (6) Could not resolve host: gasprice.poa.network
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --balance
Your balance is 489.860605 finney
$ faceid.py --withdraw 4590
Withdrew funds from the certificate
$ faceid.py --balance
Your balance is 499.860605 finney
```

Комментарий: В блокчейн сеть отправляется транзакция на получение средств с сертификата. Транзакция успешно верифицирована и включена в блок. Для проведения транзакции выбрана цена из значения `defaultGasPrice` из файла `network.json`.

Пример вызова команды (AC-020-08)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ faceid.py --withdrew 4590
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта управления контрактами. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-020-09)

`$ cat registrar.json`
`{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B40E5bCd0a1da01eb0F951x", "startBlock": 456125}}`
`$ faceid.py --withdrew 4590`
`Seems that the contract address is not the certificates contract.`

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту управления сертификатами.

Пример вызова команды (AC-020-10)

`$ faceid.py --balance`
`Your balance is 500 finney`
`$ faceid.py --withdrew 4590`
`Withdrew funds from the certificate`
`$ faceid.py --balance`
`Your balance is 510 finney`

Комментарий: Если из транзакции, которая была отправлена в результате команды `faceid.py -receive`, извлечь поле input и отправить его в новой транзакции снова в поле input на адрес контракта управления сертификатами, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка по причине того, что возврат средств уже был осуществлен ранее. Следовательно, повторно возврат не должен быть произведен.

US-021 Получение истории платежей

После индентификации пользователь может получить список платежей, связанных с аккаунтом данного пользователя.

Использование скрипта
`$ faceid.py --ops 1234`
Используя идентификатор, содержащийся в `person.json`, и PIN-код скрипт генерирует приватный ключ пользователя. В блокчейн сеть отправляется запрос на баланс аккаунта, полученного из приватного ключа. История платежей отображается с момента первого подтверждения регистрации соответствия аккаунта, для которого отображаются платежи. В истории Сумма в каждой строчке, обозначающей платеж, всегда отображается в роа с точностью до 6 знака.

Пример вызова команды (AC-021-01)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --ops 1234
Operations:
13:01:42 04.03.2019 FROM: +79418552734 2.5 poa
18:31:52 08.03.2019 FROM: +79019594479 0.5 poa
18:32:02 08.03.2019   TO: +79418552734 0.15 poa
18:33:37 08.03.2019 FROM: +79114505724 0.3 poa
18:34:17 08.03.2019   TO: +79286975842 1.3 poa
$ faceid.py --balance 1234
Your balance is 2.349641 poa
```

Комментарий: Поскольку на данном аккаунте был ненулевой баланс до подтверждения, поэтому итоговый баланс аккаунта больше суммы всех входящих платежей за вычетом всех исходящих платежей.

Пример вызова команды (AC-021-02)

```
$ cat person.json
{"id": "81e1dbe6-e0e0-4cf6-af4f-ff8341833b55"}
$ faceid.py --ops 6801
No operations found
$ faceid.py --balance 1234
Your balance is 598458 finney
```

Комментарий: Поскольку на данном аккаунте не было операций, то выводится сообщение об ошибке.

Пример вызова команды (AC-021-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --ops 1234
Operations:
13:01:42 04.03.2019 FROM: +79010451275 2.5 poa
18:31:52 08.03.2019 FROM: +79019594479 0.5 poa
18:32:02 08.03.2019   TO: +79010451275 0.15 poa
18:33:37 08.03.2019 FROM: +79114505724 0.3 poa
18:34:17 08.03.2019   TO: +79286975842 1.3 poa
$ kyc.py --list del
0x32276b955E7dCBeA1C97fe8f06053E760E739e8d: +79010451275
$ kyc.py --confirm 0x32276b955E7dCBeA1C97fe8f06053E760E739e8d
Confirmed by 0xbdf07588e6652aaee28b29156c2ae266c9ba46e253ddd99b466cce31f7
7c51a3
$ faceid.py --ops 1234
Operations:
13:01:42 04.03.2019 FROM: +79010451275 2.5 poa
18:31:52 08.03.2019 FROM: +79019594479 0.5 poa
18:32:02 08.03.2019   TO: +79010451275 0.15 poa
18:33:37 08.03.2019 FROM: +79114505724 0.3 poa
18:34:17 08.03.2019   TO: +79286975842 1.3 poa
```

Комментарий: Несмотря на то, что соответствие между аккаунтом и телефонным номером было удалено, в списке платежей выводится номер телефона, соответствовавший аккаунту в момент проведения конкретных платежей.

Пример вызова команды (AC-021-04)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --ops 1234
Operations:
13:01:42 04.03.2019 FROM: +79010451275 2.5 poa
18:31:52 08.03.2019 FROM: +79019594479 0.5 poa
18:32:02 08.03.2019   TO: +79010451275 0.15 poa
18:33:37 08.03.2019 FROM: +79114505724 0.3 poa
18:34:17 08.03.2019   TO: +79286975842 1.3 poa
$ kyc.py --list del
0xEF8eedf35C2D212bf70389ecA193622e833C3652: +79010451275
$ kyc.py --confirm 0xEF8eedf35C2D212bf70389ecA193622e833C3652
Confirmed by 0xbdf07588e6652aaee28b29156c2ae266c9ba46e253ddd99b466cce31f7
7c51a3
$ cat person.json
{"id": "81e1dbe6-e0e0-4cf6-af4f-ff8341833b55"}
$ faceid.py --add 6801 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ kyc.py --confirm 0xEF8eedf35C2D212bf70389ecA193622e833C3652
Confirmed by 0xbdf07588e6652aaee28b29156c2ae266c9ba46e253ddd99b466cce31f7
7c51a3
$ faceid.py --send 6801 +79873344556 15026871000000000000
Payment of 10 finney to +79873344556 scheduled
Transaction Hash: 0x27c9181caeb55d37e1105fa1a8648db7fe50f79064b98e56b8e85
4e3abb43728
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --ops 1234
Operations:
13:01:42 04.03.2019 FROM: +79010451275 2.5 poa
18:31:52 08.03.2019 FROM: +79019594479 0.5 poa
18:32:02 08.03.2019   TO: +79010451275 0.15 poa
18:33:37 08.03.2019 FROM: +79114505724 0.3 poa
18:34:17 08.03.2019   TO: +79286975842 1.3 poa
18:39:34 08.03.2019 FROM: +79991234567 1.502687 poa
```

Комментарий: Несмотря на то, что соответствие между аккаунтом и телефонным номером было изменено, в списке платежей выводятся оба номера телефона, соответствовавшие аккаунту в момент проведения конкретных платежей.

Пример вызова команды (AC-021-05)

```
$ cat person.json
{"id": "81e1dbe6-e0e0-4cf6-af4f-ff8341833b55"}
$ faceid.py --ops 1010
No operations found
```

Комментарий: Поскольку ситуация, когда неправильный приватный ключ сформирован из-за некорретного PIN-кода, неотличима от ситуации, когда нет операций ассоциированных с аккаунтом, в терминал выводится сообщение об ошибке.

Пример вызова команды (AC-021-06)

`$ cat registrar.json`
`cat: registrar.json: No such file or directory`
`$ faceid.py --ops 6801`
`No contract address`

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла registrar.json, содержащего адрес контракта регистра соответствий.

Пример вызова команды (AC-021-07)

`$ cat registrar.json`
`{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B40E5bCd0a1da01eb0F951x", "startBlock": 456125}}`
`$ faceid.py --ops 6801`
`Seems that the contract address is not the certificates contract.`

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле registrar.json, не принадлежит контракту регистра соответствий.

US-022 Получение истории платежей, включая использование сертификатов

Использование скрипта
`$ faceid.py --opsall 1234`

Администрирование сервиса KYC

Управление сервисом соответствий блокчейн аккаунтов и номеров телефонов происходить с использованием отдельной компоненты. Эта компонента позволяет администратору сервиса подтерждать запросы на регистрацию соответствий и запросы на удаление соответствий.

Имя скрипта
`kyc.py <command> [options]`

В зависимости от команды, через RPC узел блокчейн сети будет отправляться либо запрос на информацию, либо транзакция к контракту регистра соответствий.

US-023 Получение всех запросов на регистрацию соответствий

Любой пользователь сервиса, может получить список всех запросов на регистрацию соответствий.

Использование скрипта
`$ kyc.py --list add`
Через RPC узел блокчейн сети отправляется запрос к контракту, адрес которого указан в поле `registrar` файла `registrar.json`, на получение всех неотмененных запросов на регистрацию соответствий. Запросы выводятся в формате `отправитель запроса: номер телефона`

Пример вызова команды (AC-023-01)

```
$ setup.py --deploy
KYC Registrar: 0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd
Payment Handler: 0x95426f2bC716022fCF1dEf006dbC4bB81f5B5164
$ cat registrar.json
{"registrar": {"address": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd", "
startBlock": 123456}, "payments": {"address": "0x95426f2bC716022fCF1dEf00
6dbC4bB81f5B5164", "startBlock": 123457}}
$ kyc.py --list add
No KYC registration requests found
```

Комментарий: Поскольку в контракте, адрес которого указан в файле `registrar.json`, не было зарегистрировано ни одного запроса регистрации соответствия аккаунта, то выдается соответствующее сообщение. Транзакции в сеть не отправляются.

Пример вызова команды (AC-023-02)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list add
No KYC registration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79220012534
Registration request sent by 0xc95d677eb6f3fcb55e08274ae0eed1970391e637f1
062426a3406fd7d4cfcfcb
$ rm person.json
$ kyc.py --list add
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79220012534
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Все два запроса на регистрацию соответствия, отправленные в контракт, адрес которого указан в файле `registrar.json`, отображены в выводе команды `kyc.py -list`. Сортировка вывода происходит по номеру телефона. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-023-03)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list add
No KYC registration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79991234567
Registration request sent by 0xc95d677eb6f3fcb55e08274ae0eed1970391e637f1
062426a3406fd7d4cfcfcb
$ rm person.json
$ kyc.py --list add
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79991234567
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Два запроса на регистрацию соответствия, отправленные в контракт, адрес которого указан в файле `registrar.json`, приняты контрактом даже если они - для регистрации одного и того же номера телефона. Оба запроса отображены в выводе команды `kyc.py -list`. Сортировка вывода происходит сначала по номеру телефона, затем по адресу аккаунта. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-023-04)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list add
No KYC registration requests found
$ cat person.json
{"id": "5069b18c-9f6d-42f9-aa5f-a5f6b924d87e"}
$ faceid.py --add 1096 +79125812224
Registration request sent by 0x01bef0afddd3ebc79e80c0385bf5e0bfea871b97d9
37c74462a3221094b44c1c
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79220012534
Registration request sent by 0xc95d677eb6f3fcb55e08274ae0eed1970391e637f1
062426a3406fd7d4cfcfcb
$ cat person.json
{"id": "5069b18c-9f6d-42f9-aa5f-a5f6b924d87e"}
$ faceid.py --cancel 1096
Registration canceled by 0x99a7e5f28b653d63b5c0bbddebc91678530ee461e4fa21
c9ab2f281d28b8da5e
$ rm person.json
$ kyc.py --list add
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79220012534
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Только два запроса на регистрацию соответствия, отправленные в контракт, адрес которого указан в файле `registrar.json`, отображены в выводе команды `kyc.py -list`. Отмененный запрос не отображается. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-023-05)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list add
No KYC registration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79220012534
Registration request sent by 0xc95d677eb6f3fcb55e08274ae0eed1970391e637f1
062426a3406fd7d4cfcfcb
$ faceid.py --cancel 6104
Registration canceled by 0x99a7e5f28b653d63b5c0bbddebc91678530ee461e4fa21
c9ab2f281d28b8da5e
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --cancel 4590
Registration canceled by 0xc90366c708b1c1e8462da76682338d5262a785d592918e
d915e1292ac0d6178a
$ rm person.json
$ kyc.py --list add
No KYC registration requests found
```

Комментарий: Поскольку все запросы на регистрацию соответствий, отправленные в контракт, адрес которого указан в файле `registrar.json`, были отменены, то в выводе команды `kyc.py -list` выдается соответствующее сообщение. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-023-06)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list add
No KYC registration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xa7f3239715ff731a3d6fc477b18e35b9b0a9e1ede8
4bca5e91517e8e5bf1cc69
$ faceid.py --find /path/to/video23.avi
168f2e61-089f-88a9-53b2-3b3d0c497704 identified
$ faceid.py --del 5981
Unregistration request sent by 0xda0e2a124e6d37080b539f1bb0dc4c698b8025b6
2dcc56d75efc1115d87381aa
$ rm person.json
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Только запросы на регистрацию соответствий, оптравленные в контракт, адрес которого указан в файле `registrar.json`, отображаются в выводе команды `kyc.py -list`. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-023-07)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ kyc.py --list add
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-023-08)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ kyc.py --list add
Seems that the contract address is not the registrar contract
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

US-024 Получение всех запросов на удаление соответствий

Любой пользователь сервиса, может получить список всех запросов на удаление соответствий.

Использование скрипта

```
$ kyc.py --list del
```

Через RPC узел блокчейн сети отправляется запрос к контракту, адрес которого указан в поле `registrar` файла `registrar.json`, на получение всех неотмененных запросов на удаление соответствий. Запросы выводятся в формате `отправитель запроса: номер телефона`

Пример вызова команды (AC-024-01)

```
$ setup.py --deploy
KYC Registrar: 0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd
Payment Handler: 0x95426f2bC716022fCF1dEf006dbC4bB81f5B5164
$ cat registrar.json
{"registrar": {"address": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd", "
startBlock": 123456}, "payments": {"address": "0x95426f2bC716022fCF1dEf00
6dbC4bB81f5B5164", "startBlock": 123457}}
$ kyc.py --list del
No KYC unregistration requests found
```

Комментарий: Поскольку в контракте, адрес которого указан в файле `registrar.json`, не было зарегистрировано ни одного запроса удаления соответствия аккаунта, то выдается соответствующее сообщение. Транзакции в сеть не отправляются.

Пример вызова команды (AC-024-02)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list del
No KYC unregistration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x64e604787cbf194841e7b68d7cd28786f6c9a0a3
ab9f8b0a0e87cb4387ab0107
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --del 6104
Unregistration request sent by 0xac09810740600c31fa69f9db79ed6fc3e3281f75
8a950fe1fb254a3a3ae571b6
$ rm person.json
$ kyc.py --list del
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79220012534
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Все два запроса на удаление соответствия, отправленные в контракт, адрес которого указан в файле `registrar.json`, отображены в выводе команды `kyc.py -list`. Сортировка вывода происходит по номеру телефона. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-024-03)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list del
No KYC unregistration requests found
$ cat person.json
{"id": "5069b18c-9f6d-42f9-aa5f-a5f6b924d87e"}
$ faceid.py --del 1096
Unregistration request sent by 0x631897788617bc63fb0292c44f57a6c3824e4264
bc88fb4e3c08b910f3f417fc
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x7dbef654fd5c6145f82b72cd5dbbfc6d48fd276b
8a7a4371a61d55985b9d6d8b
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --del 6104
Unregistration request sent by 0xf65e3ff0de05ca77aa8c820bd528facc139ee4b0
986f7c350f3198928bd2c72b
$ cat person.json
{"id": "5069b18c-9f6d-42f9-aa5f-a5f6b924d87e"}
$ faceid.py --cancel 1096
Unregistration canceled by 0x24036ccf201a67256250eabe66d3f9fd72f9c4d022f2
25a8ee964be060dcb993
$ rm person.json
$ kyc.py --list del
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79220012534
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Только два запроса на удаление соответствия, отправленные в контракт, адрес которого указан в файле `registrar.json`, отображены в выводе команды `kyc.py -list`. Отмененный запрос не отображается. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-024-04)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list del
No KYC unregistration requests found
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --del 4590
Unregistration request sent by 0x535306ee4b42c92aecd0e71fca98572064f049c2
babb2769faa3bbd87d67ec2d
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --del 6104
Unregistration request sent by 0x425a3d65e6c6cbbf01507814af1ac7512f93cceb
8411381e1d5fb7adbfd44881
$ faceid.py --cancel 6104
Unregistration canceled by 0xd772c1d26a64e4114af99655ec353dc3dba1b12ba483
c0fc852e01d0432d3aa1
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --cancel 4590
Unregistration canceled by 0xe7fcf89c34605f23590ff58513f1080dc375dd06cc5e
256320151057827a258a
$ rm person.json
$ kyc.py --list del
No KYC unregistration requests found
```

Комментарий: Поскольку все запросы на удаление соответствий, оптравленные в контракт, адрес которого указан в файле `registrar.json`, были отменены, то в выводе команды `kyc.py -list` выдается соответствующее сообщение. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-024-05)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --list del
No KYC unregistration requests found
$ faceid.py --find /path/to/video23.avi
168f2e61-089f-88a9-53b2-3b3d0c497704 identified
$ faceid.py --del 5981
Unregistration request sent by 0xa413d075ec804ca282f1637fedbe82ecda199209
7d14e4d900d254e0f5fe523d
$ faceid.py --find /path/to/video24.avi
37da04e7-f471-49c7-a54c-a08f05950fc5 identified
$ faceid.py --add 4590 +79991234567
Registration request sent by 0x2f5ccb30fa919306cd2899a81c7670a9b1be09e8c1
5a0b78b0fbbc777d15e97b
$ rm person.json
$ kyc.py --list del
0x7F3faAe3f2238439d6ea5A320caB969aC62b68e3: +79870194581
```

Комментарий: Только запросы на удаление соответствий, оптравленные в контракт, адрес которого указан в файле `registrar.json`, отображаются в выводе команды `kyc.py -list`. Транзакции этой командой в сеть не отправляются.

Пример вызова команды (AC-024-06)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ kyc.py --list del
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-024-07)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ kyc.py --list del
Seems that the contract address is not the registrar contract
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

US-025 Подтверждение запросов на регистрацию или удаление соответствий

Только администратор KYC сервиса имеет полномочия подтверждать запросы на регистрацию или удаление соответствий аккаунтов пользователей их телефонным номерам.

Использование скрипта

```
$ kyc.py --confirm <address>
```

В блокчейн сеть отправляется транзакция к контракту регистра соответствий. Контракт проверяет обладает ли отправитель транзакции полномочиями по подтверждению запросов, после чего происходит изменение статуса запроса:

- если запрос был на регистрацию соответствия между аккаунтом и телефонным номером, то это соответствие становится доступным для других пользователей;
- если запрос был на удаление соответствия, то соответствие перестает существовать - по номеру телефона, который указывался в соответствии, больше нельзя будет получить адрес аккаунта.

Пример вызова команды (AC-025-01)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045
d85a470",
$ setup.py --owner registrar
Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Confirmed by 0x9960ed3041b9945289da338fa273462c820f58427408d57789c0e1400a
d5c9bb
$ kyc.py --list add
No KYC registration requests found
```

Комментарий: Поскольку аккаунт, чей приватный ключ указан в `network.json`, обладает полномочиями на подтверждение запросов на регистрацию соответствий, посланная транзакция включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков.

При подтверждении контракт производит событие (`event`) `RegistrationConfirmed`, в с указанием аккаунта, отправившего запрос:
`event RegistrationConfirmed(address indexed sender);`

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-025-02)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "6fadbaf86d2aa9c9e34a85df385a2c9afa9509e09756634e72bbcf94a
5ceb213",
$ setup.py --owner registrar
Admin account: 0x5e4d710a4995bFA3F6560effcad36C10ebac998C
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Failed but included in 0x9960ed3041b9945289da338fa273462c820f58427408d577
89c0e1400ad5c9bb
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
```

Комментарий: Поскольку аккаунт, чей приватный ключ указан в `network.json`, не обладает полномочиями на подтверждение запросов на регистрацию соответствий, посланная транзакция включается в блок, но статус ее исполнения - ошибка. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-03)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045
d85a470",
$ setup.py --owner registrar
Admin account: 0x9cce34F7aB185c7ABA1b7C8140d620B4BDA941d6
$ kyc.py --list del
0x32276b955E7dCBeA1C97fe8f06053E760E739e8d: +79010451275
$ kyc.py --confirm 0x32276b955E7dCBeA1C97fe8f06053E760E739e8d
Confirmed by 0xbdf07588e6652aaee28b29156c2ae266c9ba46e253ddd99b466cce31f7
7c51a3
$ kyc.py --list del
No KYC unregistration requests found
```

Комментарий: Поскольку аккаунт, чей приватный ключ указан в `network.json`, обладает полномочиями на подтверждение запросов на удаление соответствий, посланная транзакция включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков.

При подтверждении контракт производит событие (`event`) `UnregistrationConfirmed`, в с указанием аккаунта, отправившего запрос:

```
event UnregistrationConfirmed(address indexed sender);
```

Для проведения транзакции выбрана цена из значения `fast`, возвращенного сервисом `https://gasprice.poa.network`.

Пример вызова команды (AC-025-04)

```
$ cat network.json | python -mjson.tool | grep privKey
    "privKey": "6fadbaf86d2aa9c9e34a85df385a2c9afa9509e09756634e72bbcf94a
5ceb213",
$ setup.py --owner registrar
Admin account: 0x5e4d710a4995bFA3F6560effcad36C10ebac998C
$ kyc.py --list del
0x32276b955E7dCBeA1C97fe8f06053E760E739e8d: +79010451275
$ kyc.py --confirm 0x32276b955E7dCBeA1C97fe8f06053E760E739e8d
Failed but included in 0x360f73c63c063fae5d1e9ead454a4836ec470b54de0960f0
5909b7f8c7c140ba
$ kyc.py --list del
0x32276b955E7dCBeA1C97fe8f06053E760E739e8d: +79010451275
```

Комментарий: Поскольку аккаунт, чей приватный ключ указан в `network.json`, не обладает полномочиями на подтверждение запросов на удаление соответствий, посланная транзакция включается в блок, но статус ее исполнения - ошибка. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-05)

```
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ faceid.py --cancel 6104
Registration canceled by 0x26a975632f70533c9512e57b565b8ce8e2befee6b08dfb
c27c4a897a20e2c1d6
$ kyc.py --list add
No KYC registration requests found
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Failed but included in 0xd534ca6cd7b201fa32702a1b5b6a38881f8a63e70f9d3e1a
8a953d61c6524405
```

Комментарий: Транзакция, посланная в блокчейн сеть, на подвтерждение запроса регистрации соответствия включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков. Поскольку успешного подтверждения запрос теряет актуальность, то его нельзя в дальнейшем отменить.

Пример вызова команды (AC-025-06)

```
$ kyc.py --list del
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ faceid.py --cancel 6104
Unregistration canceled by 0xcfe93e00cbaaf37bad5f0b2ff3ab98072b4f3a2656e1
48b3bcccb553f2fa678f
$ kyc.py --list del
No KYC unregistration requests found
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Failed but included in 0x890f693d9ea1b595bb4a33efee4a21d60a63ffd2ec40c378
83b92f82fc5da52b
```

Комментарий: Аккаунт, чей приватный ключ указан в `network.json`, обладает полномочиями на подтверждение запросов на удаление соответствий. А поскольку запрос на удаление соответствия был отменен, посланная транзакция на подтерждение удаления включается в блок, но статус ее исполнения - ошибка. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-07)

```
$ kyc.py --list del
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Confirmed by 0x7b0ac78d4efde931c083b037c1c7f91b4cc8f27a159c1264ae53aa473f
3e4e6a
$ faceid.py --cancel 6104
No requests found
```

Комментарий: Транзакция, посланная в блокчейн сеть, на подвтерждение запроса удаления соответствия включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков. Поскольку успешного подтверждения запрос теряет актуальность, то его нельзя в дальнейшем отменить.

Пример вызова команды (AC-025-08)

```
$ kyc.py --list add
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Confirmed by 0x2b2a51e6e75ece6beb4c5ed9d51d381a6766e4929f34c8c545b0c0f22a
890e3d
$ faceid.py --del 6104
Unregistration request sent by 0x5a10742dc65b7a28255fe7b4d2b9f290786892e1
e437654e2ccb53bed1bfe01a
```

Комментарий: Транзакция, посланная в блокчейн сеть, на подвтерждение запроса регистрации соответствия включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков. Теперь для установленного соответствия есть возможность отправить запрос на его удаление.

Пример вызова команды (AC-025-09)

```
$ kyc.py --list del
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0xFCE1151f31065913F124917E0F2Ba5a6e29D6426
Confirmed by 0xe3cb0ca89f217511efd4af6caea80048f9921bac3b9063844e65469cf5
15f138
$ faceid.py --add 6104 +79991234567
Registration request sent by 0x41c2cc0c9e11501a3eec0956812949b1b9b2d14d9d
c7118df2ef1aac71f18767
```

Комментарий: Транзакция, посланная в блокчейн сеть, на подвтерждение запроса удаления соответствия включается в блок со статусом успешного исполнения. Статус можно подтвердить для данной транзакции в браузере блоков. Поскольку у данного аккаунта больше нет актуального соответствия, то есть возможность отправить запрос на создание нового соответствия.

Пример вызова команды (AC-025-10)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79991234567
Such phone number already registered
```

Комментарий: При запросе на регистрацию соответствия выводится сообщение об ошибке, поскольку такой номер телефона уже зарегистрирован в системе. Транзакция в блокчейн не отправляется.

Пример вызова команды (AC-025-11)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
```

Комментарий: Если из транзакции с идентификатором `0xb004fbd84afb38927636fa378c0a62a86a02b00862b6ff80fef4d6e948c0571d` извлечь поле `input` и отправить его в поле `input` новой транзакции с аккаунта, который отличется от аккаунта, отправившего эту транзакцию, на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка, поскольку запрашиваемый в данных телефонный номер уже зарегистрирован в системе. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-12)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ cat person.json
{"id": "229b3307-fe57-4ddc-b7e7-ac9f1de7abda"}
$ faceid.py --add 3928 +79991234567
Registration request sent by 0x0330c0d84464f890f3c81afbba7a3a05e0e864ec2e
31676dedcb075b2bf855a0
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
```

Комментарий: Если из транзакции, которая была отправлена в результате команды `kyc.py -confirm`, извлечь поле `input` и отправить его в новой транзакции снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка по причине того, что такое соответствие уже подтверждено. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-13)

```
$ kyc.py --list add
0xFf8E5C14d240d76EAe66aEF1A3a99895e08a0c3F: +79271173070
0xa898240Ba2D5C0C2537BC187CaC5B70843EAC8dD: +79397411449
0x60861DE06626f60d89f5795AD950B502CE00A8B0: +79017422129
$ kyc.py --list del
0x5bAD5c60781111094C247F81792eDDE9bb38818A: +79220012534
0xFCE1151f31065913F124917E0F2Ba5a6e29D6426: +79991234567
$ kyc.py --confirm 0x60861DE06626f60d89f5795AD950B502CE00A8B1
Failed but included in 0x18e28784ed3043bccd713a5f99352b74c3ebc4ca469216a6
86040f025571a184
```

Комментарий: Поскольку запрос, который пытается подтвердиться, не существует, то посланная транзакция включается в блок, но статус ее исполнения - ошибка. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-14)

```
$ cat person.json
{"id": "da04e377-47f1-c749-54ca-0fc5a08f0595"}
$ faceid.py --add 6104 +79220012534
Registration request sent by 0xa65d03add95baf22e482ebcd3423aa9bf9d8b0ec3f
ca2831960080b84536d360
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x05dee4192aaad4acb149f9615e8a72a761fc8cdb1d8b4e9c00fe66665f
5a8184
$ faceid.py --del 6104
Unregistration request sent by 0x1b6edb65b6c2611a7285cc668f4110f574a7bc3b
0d28690b40a16f141eada39a
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x1c6c2dd4b412e3c82ad1f10cef9a1a4112bc1a128dc4d12242d781aa03
ccbf2c
```

Комментарий: Если из транзакции с идентификатором `0x1b6edb65b6c2611a7285cc668f4110f574a7bc3b0d28690b40a16f141eada39a`, извлечь поле `input` и отправить его в новой транзакции снова в поле `input` на адрес контракта регистра соответствий, то эта транзакция будет включена в блок, но статус ее исполнения будет - ошибка поскольку данный аккаунт больше не зарегистрирован в регистре соответствий. Статус можно подтвердить для данной транзакции в браузере блоков.

Пример вызова команды (AC-025-15)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ kyc.py --confirm 0x60861DE06626f60d89f5795AD950B502CE00A8B0
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-025-16)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ kyc.py --confirm 0x60861DE06626f60d89f5795AD950B502CE00A8B0
Seems that the contract address is not the registrar contract
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

Пример вызова команды (AC-025-17)

```
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --confirm 0x60861DE06626f60d89f5795AD950B502CE00A8B0
No admin account found
```

Комментарий: Выдается сообщение об ошибке, если в `network.json` нет приватного ключа аккаунта, от имени которого выполнялось бы подтверждение.

US-026 Получение аккаунта по номеру телефона

Любой пользователь по номеру телефона может получить аккаунт соответствующий номеру телефона, если такое соответствие было зарегистрировано.

Использование скрипта

```
$ kyc.py --get <phone number>
```

Через RPC узел блокчейн сети отправляется запрос к контракту, адрес которого указан в поле `registrar` файла `registrar.json`, на получение соответствия. Если соответствие не зарегистрировано, контракт возвращает 0x0000000000000000000000000000000000000000.

Пример вызова команды (AC-026-01)

```
$ setup.py --deploy
KYC Registrar: 0x23B40E5bCd06D819ba81A09e0340Ec06460d2b7D
Payment Handler: 0xE797A1da01eb0F951E0E400f9343De9d17A06bac
$ kyc.py --get +79017422129
Correspondence not found
```

Комментарий: Поскольку в контракте не было зарегистированных соответствий, то соответствующее сообщение выводится на экран.

Пример вызова команды (AC-026-02)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
$ cat person.json
{"id": "229b3307-fe57-4ddc-b7e7-ac9f1de7abda"}
$ faceid.py --add 3928 +79418552734
Registration request sent by 0x0330c0d84464f890f3c81afbba7a3a05e0e864ec2e
31676dedcb075b2bf855a0
$ kyc.py --confirm 0xdCaBFD5c76D3717567568710a735717bf0C792De
Confirmed by 0x840881693295f1213bc621234bc18aa1727c90c9e1416799df22e18f9c
7c029b
$ rm person.rm
$ cat network.json
{"rpcUrl": "https://sokol.poa.network", "gasPriceUrl": "https://gasprice.
poa.network/", "defaultGasPrice": 2000000000}
$ kyc.py --get +79991234567
Registered correspondence: 0x5bAD5c60781111094C247F81792eDDE9bb38818A
```

Комментарий: Для одно из зарегистированных соответствий выводится аккаунт, с которым связан указанных номер телефона. Транзакция в сеть не отправляется.

Пример вызова команды (AC-026-03)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
$ kyc.py --get +79991234567
Registered correspondence: 0x5bAD5c60781111094C247F81792eDDE9bb38818A
$ faceid.py --del 4590
Unregistration request sent by 0xc7d00ca82740de2eef417cfa47c2970dbdec49b2
a6d15c64e85f83cf97939ca5
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x0697183ae8eedd43b56f90cfa450ea6e3ac27fb69f1056ee33ccc5a958
7bcd2a
$ kyc.py --get +79991234567
Correspondence not found
```

Комментарий: В первом случае, соответствие зарегистрировано и поэтому вызов команды позволяет получить аккаунт по номеру телефона. К моменты вызова команды второй раз, соответствие удалено, поэтому выводится сообщение об ошибке.

Пример вызова команды (AC-026-04)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --get +79991234567
Correspondence not found
```

Комментарий: Хотя регистрация соответствия только была запрошена, оно не было подтверждено, поэтому выводится сообщение об ошибке.

Пример вызова команды (AC-026-05)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
$ faceid.py --del 4590
Unregistration request sent by 0xc7d00ca82740de2eef417cfa47c2970dbdec49b2
a6d15c64e85f83cf97939ca5
$ kyc.py --get +79991234567
Registered correspondence: 0x5bAD5c60781111094C247F81792eDDE9bb38818A
```

Комментарий: Поскольку удаление соответствия не было было подтверждено, поэтому выводится информация об аккаунте.

Пример вызова команды (AC-026-06)

```
$ cat person.json
{"id": "37da04e7-f471-49c7-a54c-a08f05950fc5"}
$ faceid.py --add 4590 +79991234567
Registration request sent by 0xb004fbd84afb38927636fa378c0a62a86a02b00862
b6ff80fef4d6e948c0571d
$ cat person.json
{"id": "229b3307-fe57-4ddc-b7e7-ac9f1de7abda"}
$ faceid.py --add 3928 +79991234567
Registration request sent by 0x0330c0d84464f890f3c81afbba7a3a05e0e864ec2e
31676dedcb075b2bf855a0
$ kyc.py --confirm 0x5bAD5c60781111094C247F81792eDDE9bb38818A
Confirmed by 0x06a79e67d636d828aa58884bc7fed897698bb40922db4eadf708235cd9
c2de2c
$ kyc.py --get +79991234567
Registered correspondence: 0x5bAD5c60781111094C247F81792eDDE9bb38818A
```

Комментарий: Регистрация соответствия первого аккаунта была подтверждена, регистрация второго аккаунта не была подтверждена, поэтому выводится информация об аккаунте, отправившем первый запрос на регистрацию.

Пример вызова команды (AC-026-07)

```
$ cat registrar.json
cat: registrar.json: No such file or directory
$ kyc.py --get +79991234567
No contract address
```

Комментарий: Выдается сообщение об ошибке, если в текущей директории нет файла `registrar.json`, содержащего адрес контракта регистра соответствий. Транзакция в блокчейн сеть не отправляется.

Пример вызова команды (AC-026-08)

```
$ cat registrar.json
{"registrar": {"address": "0x340Ec06460d9b2b7D23B40E5bCd0a81A09e06D81", "
startBlock": 456123}, "payments": {"address": "0x81A09e06D81797AE2b7D23B4
0E5bCd0a1da01eb0F951x", "startBlock": 456125}}
$ kyc.py --get +79991234567
Seems that the contract address is not the registrar contract
```

Комментарий: Выдается сообщение об ошибке, если в адрес контракта, указанного в файле `registrar.json`, не принадлежит контракту регистра соответствий.

6.6. Решение

./api.py

```
from config import get
import cognitive_face as cf
from detector_util import FileLike, image_to_jpeg
from output_util import handle_error, print_error

KEY = get('faceapi.key')
cf.Key.set(KEY)
```

```
BASE_URL = get('faceapi.serviceUrl')
cf.BaseUrl.set(BASE_URL)

CONFIDENCE_THRESHOLD = 0.5

GROUP_ID = get('faceapi.groupId')


def create_group_if_not_exists():
    try:
        cf.person_group.create(GROUP_ID)
    except cf.CognitiveFaceException:
        pass


def list_groups():
    return handle_error(cf.person_group.lists())


def list_people_ids():
    try:
        return list(map(lambda person: person['personId'], cf.person.lists(GROUP_ID)))
    except cf.CognitiveFaceException as err:
        if err.code == 'PersonGroupNotFound':
            print_error('The group does not exist')


def is_group_up_to_date():
    try:
        response1 = cf.person_group.get_status(GROUP_ID)
        response2 = cf.person_group.get(GROUP_ID)

        return response1['status'] == 'succeeded' and response2['userData'] == 'ok'
    except cf.CognitiveFaceException:
        return False


def check_same_person(face_id):
    try:
        response = cf.face.identify([face_id], person_group_id=GROUP_ID,
            threshold=CONFIDENCE_THRESHOLD)
        candidates = response[0]['candidates']

        return len(candidates) != 0
    except cf.CognitiveFaceException:
        return False


def identify_person_id(faces_ids):
    if not is_group_up_to_date():
        print_error('The service is not ready')

    responses = cf.face.identify(faces_ids, person_group_id=GROUP_ID,
        threshold=CONFIDENCE_THRESHOLD)
    person_id = ''
    for response in responses:
        candidates = response['candidates']
        if len(candidates) != 1 or person_id not in ['', candidates[0]['personId']]:
            return None
        person_id = candidates[0]['personId']
```

```
    return person_id


def identify_person_id_old(image):
    response = cf.face.detect(FileLike(image_to_jpeg(image)), True, False)
    if len(response) != 1:
        print_error('There should be exactly 1 person on image')
    face_id = response[0]['faceId']
    response = cf.face.identify([face_id], person_group_id=GROUP_ID,
        threshold=CONFIDENCE_THRESHOLD)
    candidates = response[0]['candidates']

    print(candidates)

    if len(candidates) == 0:
        print_error('Cannot identify face')

    return candidates[0]['personId']


# Return face if there is exactly one face on the image
def detect(image):
    faces = cf.face.detect(FileLike(image_to_jpeg(image)), True, True, 'headPose')
    if len(faces) == 1:
        return faces[0]

    return None


def mark_group(status):
    cf.person_group.update(GROUP_ID, user_data=status)


def create_person(name, data=None):
    try:
        response = cf.person.create(GROUP_ID, name, data)
        mark_group('dirty')
        return response['personId']
    except cf.CognitiveFaceException as err:
        if err.code == 'PersonGroupNotFound':
            print_error('The group does not exist')


def delete_person(person_id):
    try:
        cf.person.delete(GROUP_ID, person_id)
        mark_group('dirty')
    except cf.CognitiveFaceException as err:
        if err.code == 'PersonGroupNotFound':
            print_error('The group does not exist')
        if err.code == 'PersonNotFound':
            print_error('The person does not exist')


def add_face(image, person_id):
    return cf.person.add_face(FileLike(image_to_jpeg(image)),
        GROUP_ID, person_id)['persistedFaceId']


def update_person_data(person_id, name, address):
```

```
    cf.person.update(GROUP_ID, person_id, name, address)
    print('Updated person %s data, name %s, address %s' % (person_id, name, address))


def get_person_address(person_id):
    response = cf.person.get(GROUP_ID, person_id)
    return response['userData']


def train():
    try:
        if is_group_up_to_date():
            print_error('Already trained')
        if len(cf.person.lists(GROUP_ID)) == 0:
            print_error('There is nothing to train')
        cf.person_group.train(GROUP_ID)
        mark_group('ok')
    except cf.CognitiveFaceException as err:
        if err.code == 'PersonGroupNotFound':
            print_error('There is nothing to train')


# Returns 0 <= confidence <= 1
def verify_person(person_id, face_id):
    response = cf.face.verify(face_id, person_group_id=GROUP_ID, person_id=person_id)
    return response['confidence']
```

./blockchain.py

```
import requests

from time import sleep
from ethereum.utils import privtoaddr
from checkers import check_pin_code
from config import get
from subprocess import check_output
import re
from web3 import Web3

from output_util import print_error

RPC_URL = get('network.rpcUrl')

web3 = Web3(Web3.HTTPProvider(RPC_URL))

try:
    PRIVATE_KEY = bytes.fromhex(get('network.privKey'))
except:
    PRIVATE_KEY = bytes(1)

DEFAULT_GAS = 100000

EMPTY_ADDRESS = '0x0000000000000000000000000000000000000000'

try:
    GAS_PRICE = int(1000000000 * requests.get(get('network.gasPriceUrl')).json()['fast'])
except requests.exceptions.RequestException:
    GAS_PRICE = get('network.defaultGasPrice')

REGISTRAR_ADDRESS = get('registrar.registrar.address')
```

```
CERTIFICATES_ADDRESS = get('registrar.payments.address')

compiled_registrar = check_output(["solc", "--optimize", "--bin", "--abi",
    "./Registrar.sol"]).decode()
compiled_registrar_abi = re.findall("ABI \\n(.*?)\\n", compiled_registrar)[0]
compiled_registrar_bytecode = re.findall("Binary: \\n(.*?)\\n", compiled_registrar)[0]

compiled_certificates = check_output(["solc", "--optimize", "--bin", "--abi",
    "Certificates.sol"]).decode()
compiled_certificates_abi = re.findall("ABI \\n(.*?)\\n", compiled_certificates)[0]
compiled_certificates_bytecode = re.findall("Binary: \\n(.*?)\\n",
    compiled_certificates)[0]

registrar = web3.eth.contract(address=REGISTRAR_ADDRESS,
                              abi=compiled_registrar_abi)
certificates = web3.eth.contract(address=CERTIFICATES_ADDRESS,
                                 abi=compiled_certificates_abi)


def check_registrar():
    if REGISTRAR_ADDRESS is None:
        print_error('No contract address')

    b1 = web3.eth.getCode(REGISTRAR_ADDRESS).hex()[2:]
    b2 = compiled_registrar_bytecode
    if b1 != b2[-len(b1):]:
        print_error('Seems that the contract address is not the registrar contract')


def check_certificates():
    if CERTIFICATES_ADDRESS is None:
        print_error('No contract address')

    b1 = web3.eth.getCode(CERTIFICATES_ADDRESS).hex()[2:]
    b2 = compiled_certificates_bytecode
    if b1 != b2[-len(b1):]:
        print_error('Seems that the contract address is not the certificates contract')


def private_key_to_address(private_key):
    return web3.toChecksumAddress(privtoaddr(private_key).hex())


def build_and_send_tx(private_key, to='', data='', value=0, nonce=None,
    gas_estimation=True, message=None):
    address = private_key_to_address(private_key)
    if nonce is None:
        nonce = web3.eth.getTransactionCount(address)
    tx = {
        'from': address,
        'to': to,
        'nonce': nonce,
        'data': data,
        'value': value,
        'gasPrice': GAS_PRICE
    }

    tx['gas'] = web3.eth.estimateGas(tx) if gas_estimation else DEFAULT_GAS

    signed = web3.eth.account.signTransaction(tx, private_key)
```

```
    tx_hash = web3.eth.sendRawTransaction(signed.rawTransaction)

    if message is not None:
        print(message)

    return wait_tx_receipt(tx_hash)


def encode_int(x, base=10):
    if type(x) == str:
        x = int(x, base)
    return x.to_bytes(32, byteorder='big')


# Waits until transaction is mined
def wait_tx_receipt(tx_hash, sleep_interval=0.5):
    while True:
        tx_receipt = web3.eth.getTransactionReceipt(tx_hash)
        if tx_receipt:
            return tx_receipt
        sleep(sleep_interval)


def get_owner_nonce():
    return web3.eth.getTransactionCount(private_key_to_address(PRIVATE_KEY))


# Deploys compiled contract, returns its address
def deploy_contract(contract_name, nonce):
    bytecode = ''
    if contract_name == 'registrar':
        bytecode = compiled_registrar_bytecode
    elif contract_name == 'certificates':
        bytecode = compiled_certificates_bytecode

    tx_receipt = build_and_send_tx(PRIVATE_KEY, data=bytecode, nonce=nonce)

    return {"address": tx_receipt['contractAddress'],
        "startBlock": tx_receipt['blockNumber']}


def get_owner():
    return registrar.functions.owner().call()


def get_requests(filter_type):
    requests = []

    head_addr = registrar.functions.headAddr().call()
    tail_addr = registrar.functions.tailAddr().call()
    node = registrar.functions.requests(head_addr).call()
    while node[1] != tail_addr:
        address = node[1]
        node = registrar.functions.requests(address).call()
        if filter_type == node[2]:
            requests += [(node[3], address)]
    return requests
```

```
def get_reg_requests():
    return get_requests(3)


def get_del_requests():
    return get_requests(4)


def change_owner(new_owner):
    if get_owner() != private_key_to_address(PRIVATE_KEY):
        print_error('Request cannot be executed')

    build_and_send_tx(PRIVATE_KEY, REGISTRAR_ADDRESS,
        web3.keccak(text='transferOwnership(address)')[:4]
        + encode_int(new_owner[2:], 16))


def get_request_type(address):
    return registrar.functions.requests(address).call()[2]


def register_request(user_private_key, phone_number):
    try:
        tx_receipt = build_and_send_tx(user_private_key, REGISTRAR_ADDRESS,
            web3.keccak(text='registerRequest(uint256)')[:4] + encode_int(phone_number))

        return tx_receipt['transactionHash'].hex()
    except ValueError as err:
        if err.args[0]['code'] == -32016:
            request_type = get_request_type(private_key_to_address(user_private_key))
            if request_type == 3:
                print_error('Registration request already sent')
            else:
                print_error('Such phone number already registered')
        if err.args[0]['code'] == -32010:
            print_error('No funds to send the request')


def delete_request(user_private_key):
    request_type = get_request_type(private_key_to_address(user_private_key))
    try:
        tx_receipt = build_and_send_tx(user_private_key, REGISTRAR_ADDRESS,
        web3.keccak(text='deleteRequest()')[:4])

        return tx_receipt['transactionHash'].hex()
    except ValueError as err:
        if err.args[0]['code'] == -32016:
            if request_type == 4:
                print_error('Unregistration request already sent')
            else:
                print_error('Account is not registered yet')
        if err.args[0]['code'] == -32010:
            print_error('No funds to send the request')


def cancel_request(user_private_key):
    request_type = get_request_type(private_key_to_address(user_private_key))
    try:
        tx_receipt = build_and_send_tx(user_private_key, REGISTRAR_ADDRESS,
            web3.keccak(text='cancelRequest()')[:4])
```

```
        return request_type, tx_receipt['transactionHash'].hex()
    except ValueError as err:
        if err.args[0]['code'] == -32016:
            if request_type == 0:
                print_error('No requests found')
        if err.args[0]['code'] == -32010:
            print_error('No funds to send the request')


def confirm(address):
    if get('network.privKey') is None:
        print_error('No admin account found')

    try:

        tx_receipt = build_and_send_tx(PRIVATE_KEY,
            REGISTRAR_ADDRESS, web3.keccak(text='confirm(address)')[:4]
            + encode_int(address[2:], 16), gas_estimation=False)

        return tx_receipt['status'], tx_receipt['transactionHash'].hex()
    except ValueError as err:
        if err.args[0]['code'] == -32010:
            print_error('No funds to confirm the request')


def create_approval(user_private_key, value, time_to_expire):
    try:
        tx_receipt = build_and_send_tx(user_private_key, CERTIFICATES_ADDRESS,
                                       web3.keccak(text='approve(uint256)')[:4]
                                       + encode_int(time_to_expire), value)

        id_hex = tx_receipt['logs'][0]['topics'][1]

        message = web3.eth.account.signHash(id_hex, user_private_key)

        return id_hex.hex()[2:] + message['signature'].hex()[2:]
    except ValueError as err:
        if err.args[0]['code'] == -32010:
            print_error('No funds to create a certificate')


def use_approval(user_private_key, message):
    id = message[:64]
    r = message[64:128]
    s = message[128:192]
    v = message[192:]

    build_and_send_tx(user_private_key, CERTIFICATES_ADDRESS,
                      web3.keccak(text='useApproval(bytes32,uint8,bytes32,bytes32)')[:4]
                      + bytes.fromhex(id)
                      + encode_int(v, 16)
                      + bytes.fromhex(r)
                      + bytes.fromhex(s))

    return id


def cancel_approvals(user_private_key):
    build_and_send_tx(user_private_key, CERTIFICATES_ADDRESS,
```

```
                    web3.keccak(text='cancel()')[:4])


def send_transaction(user_private_key, to, value):
    try:
        addr = phone_to_address(to)
        if addr is None:
            print_error('No account with the phone number +%s' % to)
        tx_receipt = build_and_send_tx(user_private_key, to=addr, value=value,
            message='Payment of %s to +%s scheduled' % (normalize_value(value), to))
        return tx_receipt['transactionHash'].hex()
    except ValueError as err:
        if err.args[0]['code'] == -32010:
            print_error('No funds to send the payment')


def pin_code_to_private_key(pin_code):
    pin_code = check_pin_code(pin_code)
    person_id = get('person.id')
    if person_id is None:
        print_error('ID is not found')
    return get_private_key(person_id, pin_code)


# Evaluate private key by id and pin_code
def get_private_key(person_id, pin_code):
    id = bytes.fromhex(person_id.replace('-', ''))

    a = web3.keccak(bytes(0))
    b = web3.keccak(a + id + bytes([int(pin_code[0])]))
    c = web3.keccak(b + id + bytes([int(pin_code[1])]))
    d = web3.keccak(c + id + bytes([int(pin_code[2])]))
    return web3.keccak(d + id + bytes([int(pin_code[3])]))


def normalize_value(value, custom_currency=None):
    if custom_currency is not None:
        return ('%.6f' % web3.fromWei(value, custom_currency)).rstrip('0').rstrip('.') \
               + ' %s' % (custom_currency if custom_currency != 'ether' else 'poa')

    for currency in ['ether', 'finney', 'szabo', 'gwei', 'mwei', 'kwei', 'wei']:
        if web3.fromWei(value, currency) >= 1:
            return ('%.6f' % web3.fromWei(value, currency)).rstrip('0').rstrip('.') \
                   + ' %s' % (currency if currency != 'ether' else 'poa')
    return '0 poa'


def phone_to_address(phone):
    address = registrar.functions.db(int(phone)).call()
    if address == EMPTY_ADDRESS:
        return None
    return address


def address_to_phone(address, block='latest'):
    phone =
        registrar.functions.db_rev(web3.toChecksumAddress(address)).call(block_identifier=block)

    if phone == 0:
        return 'UNKNOWN'
```

```
    return '+%d' % phone


def get_balance(address):
    return web3.eth.getBalance(address)


def check_address(address):
    if not web3.isChecksumAddress(address):
        print_error('Wrong address format')
    return address


def get_tx_list(address):
    return requests.get(
        'https://blockscout.com/poa/sokol/api?module=account&action=txlist&address=%s\
        &startblock=%s' % (
            address, get('registrar.registrar.startBlock'))
    ).json()['result']
```

./checkers.py

```
from output_util import print_error
from datetime import datetime as dt


def check_pin_code(pin_code):
    if len(pin_code) != 4 or not pin_code.isdecimal():
        print_error('Pin code should have length 4 and contain only digits')
    return pin_code


def check_name(name):
    if len(name) > 128 or not name.isalpha():
        print_error('Name should have length at most 128 and contain only letters')
    return name


def check_phone_number(phone_number):
    if len(phone_number) != 12 or not phone_number[1:].isdecimal()
        or phone_number[0] != '+':
        print_error('Incorrect phone number')
    return phone_number[1:]


def check_date(date_str):
    try:
        date = dt.strptime(date_str, '%H:%M %d.%m.%Y')
    except ValueError:
        print_error('Expiration date is invalid')

    delta = int((date - dt.now()).total_seconds())
    if delta <= 0:
        print_error('Expiration date is invalid')
    return delta


def check_ticket(ticket):
    return ticket
```

```
def check_message(message):
    if len(message) != 140:
        print_error('Message should have length 140')
    return message
```

./config.py

```
import json

# key format a.b.c => ./a.json -> "b" -> "c"
import os


def get(key):
    try:
        keys = key.split('.')
        config = json.load(open('./%s.json' % keys[0]))
        cur = config
        for k in keys[1:]:
            cur = cur[k]

        return cur
    except (IOError, json.JSONDecodeError, KeyError):
        return None


def set(key, value):
    keys = key.split('.')
    try:
        config = json.load(open('./%s.json' % keys[0]))
    except (IOError, json.JSONDecodeError, KeyError):
        config = dict()

    cur = config
    for k in keys[1:-1]:
        cur = cur[k]
    cur[keys[-1]] = value
    open('./%s.json' % keys[0], 'w+').write(json.dumps(config))


def clear(key):
    try:
        os.remove('./%s.json' % key)
    except OSError:
        pass
```

./contracts/Certificates.sol

```
pragma solidity ^0.5.2;


contract Certificates {
    event CertificateCreated(bytes32 indexed id);
    event CertificateUsed(bytes32 indexed id);
    event CertificateWithdrew(bytes32 indexed id);

    struct Approval {
        address sender;
```

```
        uint value;
        uint expire_time;
        bytes32 prev;
        bytes32 next;
    }

    mapping(address => uint) public lastN;

    // sha(sender, id) => approval
    mapping(bytes32 => Approval) public approvals;

    // address => last approval ticket
    mapping(address => bytes32) public lastTicket;

    function approve(uint time) public payable {
        require(msg.value > 0);

        bytes32 prevId = lastTicket[msg.sender];
        if (prevId == bytes32(0))
            prevId = keccak256(abi.encodePacked(msg.sender, uint(0)));

        uint n = ++lastN[msg.sender];

        bytes32 id = keccak256(abi.encodePacked(msg.sender, n));
        lastTicket[msg.sender] = id;

        approvals[prevId].next = id;
        approvals[id] = Approval(msg.sender, msg.value, now + time, prevId, bytes32(0));

        emit CertificateCreated(id);
    }

    function cancel() public {
        bytes32 id = keccak256(abi.encodePacked(msg.sender, uint(0)));
        while (approvals[id].next != bytes32(0)) {
            id = approvals[id].next;
            if (now > approvals[id].expire_time)
                cancelApproval(id);
        }
    }

    function cancelApproval(bytes32 ticket) public {
        require(approvals[ticket].value > 0);
        require(approvals[ticket].sender == msg.sender);
        require(now > approvals[ticket].expire_time);

        approvals[approvals[ticket].prev].next = approvals[ticket].next;
        if (approvals[ticket].next != bytes32(0))
            approvals[approvals[ticket].next].prev = approvals[ticket].prev;
        else
            lastTicket[msg.sender] = approvals[ticket].prev;

        msg.sender.transfer(approvals[ticket].value);

        delete approvals[ticket];

        emit CertificateWithdrew(ticket);
    }

    function useApproval(bytes32 ticket, uint8 v, bytes32 r, bytes32 s) public {
```

```
        require(approvals[ticket].value > 0);
        require(ecrecover(ticket, v, r, s) == approvals[ticket].sender);
        require(now <= approvals[ticket].expire_time);

        approvals[approvals[ticket].prev].next = approvals[ticket].next;
        if (approvals[ticket].next != bytes32(0))
            approvals[approvals[ticket].next].prev = approvals[ticket].prev;
        else
            lastTicket[msg.sender] = approvals[ticket].prev;

        msg.sender.transfer(approvals[ticket].value);

        delete approvals[ticket];

        emit CertificateUsed(ticket);
    }
}
```

./contracts/Registrar.sol

```
pragma solidity ^0.5.2;


contract Registrar {
    event Register(uint indexed phone_number, address addr);
    event Unregister(uint indexed phone_number);
    event RegistrationRequest(address indexed sender);
    event UnregistrationRequest(address indexed sender);
    event RegistrationCanceled(address indexed sender);
    event UnregistrationCanceled(address indexed sender);
    event RegistrationConfirmed(address indexed sender);
    event UnregistrationConfirmed(address indexed sender);

    struct Account {
        uint phone;
        address addr;
    }

    enum NodeType {
        NONE, HEAD, TAIL, REG, DEL
    }

    struct Node {
        address prev;
        address next;
        NodeType t;
        uint data;
    }

    // Double linked list
    address public headAddr = address(0);
    address public tailAddr = address(2 ** 160 - 1);
    // Sender => Request node
    mapping(address => Node) public requests;

    address public owner;

    // Phone number => Address
    mapping(uint => address) public db;
```

```
    // Address => Phone number
    mapping(address => uint) public db_rev;

    constructor() public {
        owner = msg.sender;
        // head
        requests[headAddr] = Node(headAddr, tailAddr, NodeType.HEAD, 0);
        // tail
        requests[tailAddr] = Node(headAddr, tailAddr, NodeType.TAIL, 0);
    }

    // Only contract owner can call these methods
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _owner) onlyOwner public {
        owner = _owner;
    }


    function registerRequest(uint phone) public {
        require(phone <= 1e12);
        require(db[phone] == address(0));
        require(requests[msg.sender].t == NodeType.NONE);

        requests[msg.sender].t = NodeType.REG;
        requests[msg.sender].data = phone;

        requests[msg.sender].next = tailAddr;
        requests[requests[tailAddr].prev].next = msg.sender;
        requests[msg.sender].prev = requests[tailAddr].prev;
        requests[tailAddr].prev = msg.sender;

        emit RegistrationRequest(msg.sender);
    }

    function deleteRequest() public {
        require(db_rev[msg.sender] != 0);
        require(requests[msg.sender].t == NodeType.NONE);

        requests[msg.sender].t = NodeType.DEL;

        requests[msg.sender].next = tailAddr;
        requests[requests[tailAddr].prev].next = msg.sender;
        requests[msg.sender].prev = requests[tailAddr].prev;
        requests[tailAddr].prev = msg.sender;

        emit UnregistrationRequest(msg.sender);
    }

    function confirm(address addr) onlyOwner public {
        require(requests[addr].t == NodeType.REG || requests[addr].t == NodeType.DEL);

        if (requests[addr].t == NodeType.REG) {
            db_rev[addr] = requests[addr].data;
            db[db_rev[addr]] = addr;
            emit RegistrationConfirmed(addr);
        }
```

```
        else {
            delete db[db_rev[addr]];
            delete db_rev[addr];
            emit UnregistrationConfirmed(addr);
        }
        requests[addr].t = NodeType.NONE;
        requests[requests[addr].prev].next = requests[addr].next;
        requests[requests[addr].next].prev = requests[addr].prev;
    }

    function cancelRequest() public {
        require(requests[msg.sender].t != NodeType.NONE);

        if (requests[msg.sender].t == NodeType.REG)
            emit RegistrationCanceled(msg.sender);
        if (requests[msg.sender].t == NodeType.DEL)
            emit UnregistrationCanceled(msg.sender);

        requests[msg.sender].t = NodeType.NONE;
        requests[requests[msg.sender].prev].next = requests[msg.sender].next;
        requests[requests[msg.sender].next].prev = requests[msg.sender].prev;
    }
}
```

./detector_util.py

```
import cv2
import os
import dlib

from output_util import print_error

PREDICTOR_PATH = '/opt/shape_predictor_68_face_landmarks.dat'
    if os.path.exists('/opt/shape_predictor_68_face_landmarks.dat')
    else './shape_predictor_68_face_landmarks.dat'
EYE_THRESHOLD = 0.22
MOUTH_THRESHOLD = 0.3

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)


class FileLike:
    def __init__(self, image):
        self.image = image

    def read(self):
        return self.image


def shape_to_list(shape):
    coords = [None] * 68

    for i in range(0, 68):
        coords[i] = (shape.part(i).x, shape.part(i).y)

    return coords


def right_eye_open(points):
```

```
    eye = points[36:42]
    return aspect_ratio(eye) > EYE_THRESHOLD


def left_eye_open(points):
    eye = points[42:48]
    return aspect_ratio(eye) > EYE_THRESHOLD


def mouth_open(points):
    mouth = [points[60], points[61], points[63], points[64], points[65], points[67]]
    return aspect_ratio(mouth) > MOUTH_THRESHOLD


def aspect_ratio(eye):
    return (dist(eye[1], eye[5]) + dist(eye[2], eye[4])) / (2 * dist(eye[0], eye[3]))


def dist(a, b):
    return ((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) ** 0.5


def landmarks(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    dets, scores, idx = detector.run(gray, 0, -1)

    if len(dets) == 0:
        return None

    return shape_to_list(predictor(image, dets[0]))


def image_to_jpeg(image):
    status, encoded_image = cv2.imencode('.jpeg', image)
    if not status:
        print_error('Failed to encode frame into jpeg')
    return encoded_image.tostring()
```

./face_conditions.py

```
import cv2

from detector_util import landmarks, left_eye_open, right_eye_open, mouth_open

DEFAULT_ANGLE_DELTA = 3


class FaceConditions:
    def __init__(self, label, yaw=None, roll=None, delta=DEFAULT_ANGLE_DELTA,
                 left_eye=None, right_eye=None, mouth=None):
        self.label = label
        self.completed = False

        self.yaw = yaw
        self.yaw_delta = delta
        self.roll = roll
        self.roll_delta = delta
        self.left_eye = left_eye
        self.right_eye = right_eye
```

```
        self.mouth = mouth

    def check_face(self, image, face_api_response=None):
        if self.completed:
            return

        if self.yaw is not None and self.roll is not None and face_api_response is not None:
            yaw_value = face_api_response['faceAttributes']['headPose']['yaw']
            roll_value = face_api_response['faceAttributes']['headPose']['roll']

            if self.yaw is not None and (yaw_value < self.yaw - self.yaw_delta or yaw_value > self.ya
                return
            if self.roll is not None and (
                    roll_value < self.roll - self.roll_delta or roll_value > self.roll + self.roll_de
                return

        points = landmarks(image)
        if points is None:
            return False

        if self.left_eye is not None and self.left_eye != left_eye_open(points):
            return
        if self.right_eye is not None and self.right_eye != right_eye_open(points):
            return
        if self.mouth is not None and self.mouth != mouth_open(points):
            return

        self.completed = True

        return True
```

./faceid.py

```
from argparse import ArgumentParser
from datetime import datetime

from blockchain import register_request, delete_request, \
    cancel_request, get_balance, private_key_to_address, \
    send_transaction, create_approval, use_approval, \
    cancel_approvals, pin_code_to_private_key, \
    normalize_value, check_registrar, check_certificates, \
    get_tx_list, address_to_phone
from config import get, set, clear
from output_util import print_error
from validators import PINPhoneValidate, TransactValidate, ApproveValidate, \
    PINTicketValidate
from video_detector import simple_identify, identify

parser = ArgumentParser(prog='Faceid service')

parser.add_argument('--find',
                    nargs=1,
                    metavar='VIDEO',
                    help='Identifies person, stores person id in person.json')
parser.add_argument('--actions',
                    action='store_true',
                    help='Generates needed face actions')
parser.add_argument('--add',
                    nargs=2,
                    metavar=('PIN_CODE', 'PHONE_NUMBER'),
```

```
                    action=PINPhoneValidate,
                    help='Sends register request')
parser.add_argument('--del',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    dest='delete',
                    help='Sends delete request')
parser.add_argument('--cancel',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    help='Cancels register or unregister request')
parser.add_argument('--balance',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    help='Prints user balance')
parser.add_argument('--send',
                    nargs=3,
                    metavar=('PIN_CODE', 'PHONE_NUMBER', 'VALUE'),
                    action=TransactValidate,
                    help='Sends money to another user')
parser.add_argument('--gift',
                    nargs=3,
                    metavar=('PIN_CODE', 'VALUE', 'EXPIRE_DATE'),
                    action=ApproveValidate,
                    help='Creates approval ticket')
parser.add_argument('--receive',
                    nargs=2,
                    metavar=('PIN_CODE', 'TICKET'),
                    action=PINTicketValidate,
                    help='Receives money via given ticket')
parser.add_argument('--withdraw',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    help='Withdraw money from unused ticket')
parser.add_argument('--ops',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    help='Prints all operations')
parser.add_argument('--opsall',
                    nargs=1,
                    metavar='PIN_CODE',
                    type=pin_code_to_private_key,
                    help='Prints all operations including certificates')

args = parser.parse_args()

if args.find:
    clear('person')
    video = args.find[0]

    actions = get('actions')
    if actions is None:
        person_id = simple_identify(video)
        set('person.id', person_id)
        print('%s identified' % person_id)
```

```
    else:
        person_id = identify(video, actions)
        set('person.id', person_id)
        print('%s identified' % person_id)

if args.actions:
    import json, random

    actions = random.choice([
        {'actions': ["OpenMouth", "YawLeft", "CloseLeftEye"]},
        {'actions': ["OpenMouth", "YawLeft", "CloseRightEye"]},
        {'actions': ["OpenMouth", "CloseRightEye", "YawLeft"]},
        {'actions': ["OpenMouth", "YawRight", "CloseRightEye"]}
    ])
    with open('actions.json', 'w') as out:
        json.dump(actions, out)

if args.add:
    check_registrar()
    private_key = args.add[0]
    phone = args.add[1]

    tx_hash = register_request(private_key, phone)

    print('Registration request sent by %s' % tx_hash)

if args.delete:
    check_registrar()
    private_key = args.delete[0]

    tx_hash = delete_request(private_key)

    print('Unregistration request sent by %s' % tx_hash)

if args.cancel:
    check_registrar()
    private_key = args.cancel[0]

    type, tx_hash = cancel_request(private_key)

    if type == 3:
        print('Registration canceled by %s' % tx_hash)
    else:
        print('Unregistration canceled by %s' % tx_hash)

if args.balance:
    private_key = args.balance[0]
    address = private_key_to_address(private_key)

    print('Your balance is %s' % normalize_value(get_balance(address)))

if args.send:
    check_registrar()
    private_key = args.send[0]
    to = args.send[1]
    value = args.send[2]

    tx_hash = send_transaction(private_key, to, value)

    print('Transaction Hash: %s' % tx_hash)
```

```
if args.gift:
    check_certificates()
    private_key = args.gift[0]
    value = args.gift[1]
    expire_time = args.gift[2]

    certificate = create_approval(private_key, value, expire_time)

    print(certificate)

if args.receive:
    check_certificates()
    private_key = args.receive[0]
    certificate = args.receive[1]

    ticket = use_approval(private_key, certificate)

    print(ticket)

if args.withdraw:
    check_certificates()
    private_key = args.withdraw[0]

    cancel_approvals(private_key)

    print('Cancelled approvals')

if args.ops:
    check_registrar()
    check_certificates()

    address = private_key_to_address(args.ops[0]).lower()

    tx_list = get_tx_list(address)

    if len(tx_list) == 0:
        print_error('No operations found')

    print('Operations:')
    for tx in tx_list:
        date = datetime.fromtimestamp(int(tx['timeStamp']))

        if int(tx['blockNumber']) >= get('registrar.registrar.startBlock'):
            if tx['to'] == address:
                from_phone = address_to_phone(tx['from'], int(tx['blockNumber']))
                if from_phone != 'UNKNOWN':
                    print('%s FROM: %s %s' % (
                    date.strftime('%H:%M:%S %d.%m.%Y'), from_phone,
                        normalize_value(int(tx['value']), 'ether')))
            else:
                to_phone = address_to_phone(tx['to'], int(tx['blockNumber']))
                if to_phone != 'UNKNOWN':
                    print('%s   TO: %s %s' % (
                    date.strftime('%H:%M:%S %d.%m.%Y'), to_phone,
                        normalize_value(int(tx['value']), 'ether')))
```

./face-management.py

```
from argparse import ArgumentParser
from threading import Thread
```

```
from time import sleep

from api import list_people_ids, delete_person, train, create_person, \
    add_face, create_group_if_not_exists
from output_util import print_error
from video_detector import detect_all, detect_middle_faces, \
    detect_roll_faces, detect_yaw_faces, \
    detect_open_mouth_faces, detect_open_eyes_faces

parser = ArgumentParser(prog='Faces management')

parser.add_argument('--simple-add',
                    nargs=1,
                    metavar='VIDEO',
                    help='Adds 5 faces from video')
parser.add_argument('--add',
                    nargs='+',
                    metavar='VIDEO',
                    help='Adds only needed faces from 1-5 videos')
parser.add_argument('--list',
                    action='store_true',
                    help='Prints all registered person ids')
parser.add_argument('--del',
                    nargs=1,
                    metavar='ID',
                    dest='delete',
                    help='Deletes person')
parser.add_argument('--train',
                    action='store_true',
                    help='Starts training')

args = parser.parse_args()

if args.simple_add:
    video = args.simple_add[0]

    faces = detect_all(video)
    if faces is None:
        print_error('Video does not contain any face')

    create_group_if_not_exists()

    print('%d frames extracted' % len(faces))

    person_id = create_person('1')

    print('PersonId: %s' % person_id)
    print('FaceIds')
    print('=======')

    for face in faces:
        print(add_face(face, person_id))

if args.add:
    person_id = ''
    faces = []

    done = 0
    results = [0, 0, 0, 0, 0]
    if len(args.add) > 0:
```

```
        def target(video):
            global results, done
            results[0] = detect_middle_faces(video)
            done += 1

        video = args.add[0]

        t = Thread(target=target, args=(video,))
        t.run()

    if len(args.add) > 1:
        def target(video):
            global results, done
            results[1] = detect_roll_faces(video)
            done += 1

        video = args.add[1]

        t = Thread(target=target, args=(video,))
        t.run()

    if len(args.add) > 2:
        def target(video):
            global results, done
            results[2] = detect_yaw_faces(video)
            done += 1

        video = args.add[2]

        t = Thread(target=target, args=(video,))
        t.run()

    if len(args.add) > 3:
        def target(video):
            global results, done
            results[3] = detect_open_mouth_faces(video)
            done += 1

        video = args.add[3]

        t = Thread(target=target, args=(video,))
        t.run()

    if len(args.add) > 4:
        def target(video):
            global results, done
            results[4] = detect_open_eyes_faces(video)
            done += 1

        video = args.add[4]

        t = Thread(target=target, args=(video,))
        t.run()

    while len(args.add) > done:
        sleep(0.5)

    if len(args.add) > 0:
        if results[0] is None:
            print_error('Base video does not follow requirements')
```

```
            faces += results[0]

        if len(args.add) > 1:
            if results[1] is None:
                print_error('Roll video does not follow requirements')

            faces += results[1]

        if len(args.add) > 2:
            if results[2] is None:
                print_error('Yaw video does not follow requirements')

            faces += results[2]

        if len(args.add) > 3:
            if results[3] is None:
                print_error('Video to detect open mouth does not follow requirements')

            faces += results[3]

        if len(args.add) > 4:
            if results[4] is None:
                print_error('Video to detect open eyes does not follow requirements')

            faces += results[4]

        create_group_if_not_exists()
        person_id = create_person('1')

        print('%d frames extracted' % len(faces))
        print('PersonId: %s' % person_id)
        print('FaceIds')
        print('=======')

        for face in faces:
            print(add_face(face, person_id))

    if args.list:
        list_ids = list_people_ids()

        if len(list_ids) == 0:
            print_error('No persons found')

        print('Persons IDs:')

        for id in list_ids:
            print(id)

    if args.delete:
        id = args.delete[0]

        delete_person(id)

        print('Person deleted')

    if args.train:
        train()

        print('Training successfully started')
```

./kyc.py

```
from argparse import ArgumentParser

from blockchain import get_reg_requests, get_del_requests, \
     confirm, phone_to_address, check_address, check_registrar
from checkers import check_phone_number
from output_util import print_error

parser = ArgumentParser(prog='KYC')

parser.add_argument('--list',
                    nargs=1,
                    metavar='REQUEST_TYPE',
                    help='Prints out all requests of given type')
parser.add_argument('--confirm',
                    nargs=1,
                    metavar='ADDRESS',
                    type=check_address,
                    help='Confirms request related to the given address')
parser.add_argument('--get',
                    nargs=1,
                    type=check_phone_number,
                    metavar='PHONE',
                    help='Prints address associated with given phone number')

args = parser.parse_args()

if args.list:
    check_registrar()
    request_type = args.list[0]

    lst = []
    if request_type == 'add':
        lst = sorted(get_reg_requests())
        if len(lst) == 0:
            print_error('No KYC registration requests found')
    elif request_type == 'del':
        lst = sorted(get_del_requests())
        if len(lst) == 0:
            print_error('No KYC unregistration requests found')

    for phone, address in lst:
        print('%s: +%s' % (address, str(phone)))

if args.confirm:
    check_registrar()
    address = args.confirm[0]

    status, tx_hash = confirm(address)

    if status == 1:
        print('Confirmed by %s' % tx_hash)
    else:
        print('Failed but included in %s' % tx_hash)

if args.get:
    check_registrar()
    phone = args.get[0]

    addr = phone_to_address(phone)
```

```
    if addr is None:
        print_error('Correspondence not found')
    print('Registered correspondence: %s' % phone_to_address(phone))
```

./output_util.py

```
from json import dumps

def print_error(message):
    print(message)
    quit(0)

def print_json(obj):
    print(dumps(obj, indent=4))

def handle_error(response):
    if type(response) == dict and response.get('error'):
        print_error(response['error'])
    return response
```

./setup.py

```
from time import sleep

from argparse import ArgumentParser

from blockchain import deploy_contract, get_owner, change_owner, get_owner_nonce
from config import set

parser = ArgumentParser(prog='Setup')

parser.add_argument('--deploy',
                action='store_true',
                help='Deploys registrar and certificates contracts to the blockchain')
parser.add_argument('--owner',
                    nargs=1,
                    metavar='CONTRACT',
                    help='Prints current owner of the contract')
parser.add_argument('--chown',
                    nargs=2,
                    metavar=('CONTRACT', 'NEW_OWNER'),
                    help='Changes owner of the specified contract')


args = parser.parse_args()

if args.deploy:
    nonce = get_owner_nonce()
    registrar = deploy_contract('registrar', nonce)
    print('KYC Registrar: %s' % registrar['address'])
    set('registrar.registrar', registrar)

    certificates = deploy_contract('certificates', nonce + 1)
    print('Payment Handler: %s' % certificates['address'])
    set('registrar.payments', certificates)

if args.owner:
    contract = args.owner[0]
```

```
    if contract == 'registrar':
        print('Admin account: %s' % get_owner())

if args.chown:
    contract = args.chown[0]
    new_owner = args.chown[1]

    if contract == 'registrar':
        change_owner(new_owner)

        print('New admin account: %s' % new_owner)
```

./validators.py

```
from argparse import Action

from checkers import check_pin_code, check_phone_number, check_date, check_ticket

from blockchain import pin_code_to_private_key, phone_to_address


# Substitutes pin with private_key
class PINPhoneValidate(Action):
    def __call__(self, parser, args, values, option_string=None):
        pin_code, phone_number = values
        res = [pin_code_to_private_key(pin_code), check_phone_number(phone_number)]
        setattr(args, self.dest, res)


# Substitutes pin with private_key
class PINTicketValidate(Action):
    def __call__(self, parser, args, values, option_string=None):
        pin_code, ticket = values
        res = [pin_code_to_private_key(pin_code), check_ticket(ticket)]
        setattr(args, self.dest, res)


# Substitutes pin with private_key
# Substitutes phone with address
class PersonValidate(Action):
    def __call__(self, parser, args, values, option_string=None):
        name, phone_number, pin_code = values
        res = [pin_code_to_private_key(name),
               phone_to_address(check_phone_number(phone_number)),
               check_pin_code(pin_code)]
        setattr(args, self.dest, res)


# Substitutes pin with private_key
class TransactValidate(Action):
    def __call__(self, parser, args, values, option_string=None):
        pin_code, phone_number, amount = values
        res = [pin_code_to_private_key(pin_code),
               check_phone_number(phone_number),
               int(amount)]
        setattr(args, self.dest, res)


# Substitutes pin with private_key
class ApproveValidate(Action):
```

```
    def __call__(self, parser, args, values, option_string=None):
        pin_code, value, date_to_expire = values
        res = [pin_code_to_private_key(pin_code), int(value),
            check_date(date_to_expire)]
        setattr(args, self.dest, res)
```

./video_ detector.py

```
import cv2
import imutils

from config import get, set, clear
from api import detect, verify_person, check_same_person, identify_person_id
from face_conditions import FaceConditions
from output_util import print_error

WIDTH = 1000
CONDITIONS_TO_VERIFY = 3

def detect_all(video):
    faces_images = []

    cap = cv2.VideoCapture(video)
    success, frame = cap.read()
    count = 0

    while success:
        if count % 5 == 0:
            frame = imutils.resize(frame, width=WIDTH)

            face_api_response = detect(frame)

            if face_api_response is not None:
                if check_same_person(face_api_response['faceId']):
                    print_error('The same person already exists')

                faces_images.append(frame)

                if len(faces_images) == 5:
                    return faces_images

        count += 1
        success, frame = cap.read()

    return None


def detect_by_conditions(video, conditions, face_api=True):
    cur = 0
    faces_images = []

    cap = cv2.VideoCapture(video)
    success, frame = cap.read()
    count = 0

    while success:
        if count % 3 == 0:
            frame = imutils.resize(frame, width=WIDTH)

            face_api_response = detect(frame) if face_api else 0
```

```
            if face_api_response is not None
                and conditions[cur].check_face(frame, face_api_response):
                if face_api_response != 0 and
                    check_same_person(face_api_response['faceId']):
                    print_error('The same person already exists')

                faces_images.append(frame)
                cur += 1

                if cur == len(conditions):
                    return faces_images

        count += 1
        success, frame = cap.read()

    return None


def detect_by_conditions_in_any_order(video, conditions, face_api=True):
    completed = 0
    faces_images = []

    cap = cv2.VideoCapture(video)
    success, frame = cap.read()

    while success:
        frame = imutils.resize(frame, width=WIDTH)
        face_api_response = detect(frame) if face_api else 0
        if face_api_response is not None:
            for condition in conditions:
                if condition.check_face(frame, face_api_response):
                    faces_images.append(frame)
                    completed += 1
                    break

            if completed == len(conditions):
                return faces_images

        success, frame = cap.read()

    return None


def detect_middle_faces(video):
    return detect_by_conditions(video, [
        FaceConditions('middle_face1', yaw=0, roll=0, delta=5),
        FaceConditions('middle_face2', yaw=0, roll=0, delta=5),
        FaceConditions('middle_face3', yaw=0, roll=0, delta=5),
        FaceConditions('middle_face4', yaw=0, roll=0, delta=5),
        FaceConditions('middle_face5', yaw=0, roll=0, delta=5)
    ])


def detect_roll_faces(video):
    return detect_by_conditions(video, [
        FaceConditions('roll_left_30', roll=30),
        FaceConditions('roll_left_15', roll=15),
        FaceConditions('roll_middle_0', roll=0),
        FaceConditions('roll_right_15', roll=-15),
        FaceConditions('roll_right_30', roll=-30)
```

```
    ])


def detect_yaw_faces(video):
    return detect_by_conditions(video, [
        FaceConditions('yaw_left_20', yaw=20),
        FaceConditions('yaw_left_10', yaw=10),
        FaceConditions('yaw_middle', yaw=0),
        FaceConditions('yaw_right_10', yaw=-10),
        FaceConditions('yaw_right_20', yaw=-20)
    ])


def detect_open_mouth_faces(video):
    return detect_by_conditions(video, [
        FaceConditions('open_mouth', mouth=True)
    ], face_api=False)


def detect_open_eyes_faces(video):
    return detect_by_conditions_in_any_order(video, [
        FaceConditions('closed_left_eye', left_eye=False, right_eye=True),
        FaceConditions('closed_right_eye', left_eye=True, right_eye=False),
    ], face_api=False)


def simple_identify(video):
    cap = cv2.VideoCapture(video)
    success, frame = cap.read()
    count = 0

    faces_ids = []

    while success:
        if count % 5 == 0:
            frame = imutils.resize(frame, width=WIDTH)

            face_api_response = detect(frame)

            if face_api_response is not None:
                faces_ids += [face_api_response['faceId']]

                if len(faces_ids) == 5:
                    break

        count += 1
        success, frame = cap.read()

    if len(faces_ids) < 5:
        print_error('The video does not follow requirements')

    person_id = identify_person_id(faces_ids)

    if person_id is None:
        print_error('The person was not found')

    return person_id


def action_to_face_condition(action):
```

```
    return action


def identify(video, actions):
    cap = cv2.VideoCapture(video)
    success, frame = cap.read()

    actions = list(map(action_to_face_condition, actions))

    count = 0
    cur = 0

    faces_ids = []

    while success:
        if count % 5 == 0:
            frame = imutils.resize(frame, width=WIDTH)

            face_api_response = detect(frame)

            if face_api_response is not None
                and actions[cur].check(frame, face_api_response):
                faces_ids += [face_api_response['faceId']]

                cur += 1
                if cur == len(actions):
                    break

        count += 1
        success, frame = cap.read()

    if cur < len(actions):
        print_error('The video does not follow requirements')

    person_id = identify_person_id(faces_ids)

    if person_id is None:
        print_error('The person was not found')

    return person_id
```