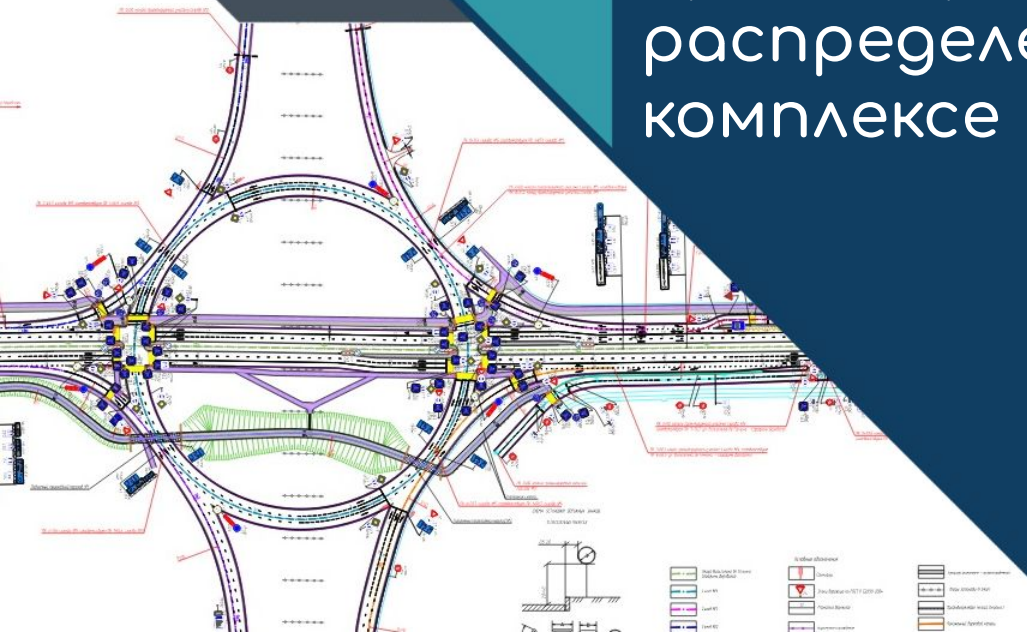
Транспортная подсистема в распределенном программном комплексе



Александр Мошков, ЦРТ

Определение транспортной подсистемы

Транспортная подсистема предназначена для удовлетворения транспортных потребностей **системы** и включает в себя **средства транспортировки**, **объекты транспортировки**, а также **инфраструктуру** для взаимодействия с ними.

Прикладная система для использования подсистемы

Каждая теория интересна не сама по себе, а в приложении к какой-то конкретной проблеме, где она может принести очевидную пользу

Мы в качестве примера возьмем WEB сервис биометрической верификации по лицу

WEB сервис биометрической верификации по лицу

(Регистрация эталона человека)

HTTP спецификация

```
Метод : POST
Ресурс: person/{person_id}
Вход  : {
          "type": "IMAGE",
          "data": "base64"
        }
Ответ : 204
Выход : нет
```

Функционал

- Принять фотографию + идентификатор
- Проверить привилегии пользователя
- Построить биометрическую модель
- Сохранить модель в БД системы
- Вернуть признак успеха операции

WEB сервис биометрической верификации по лицу

(Верификация человека по видео)

HTTP спецификация

```
Метод : PUT
Ресурс: person/{person_id}
Вход  : {
          "type": "VIDEO",
          "data": "base64"
        }
Ответ : 200
Выход : {
          "score": 56.3
        }
```

Функционал

- Принять идентификатор человека + видео
- Проверить привилегии пользователя
- Извлечь кадр из видео (при необходимости)
- Построить биометрическую модель по кадру
- Сравнить эталон с построенной моделью
- Вернуть степень схожести

WEB сервис биометрической верификации по лицу

(Удаление эталона из БД)

HTTP спецификация

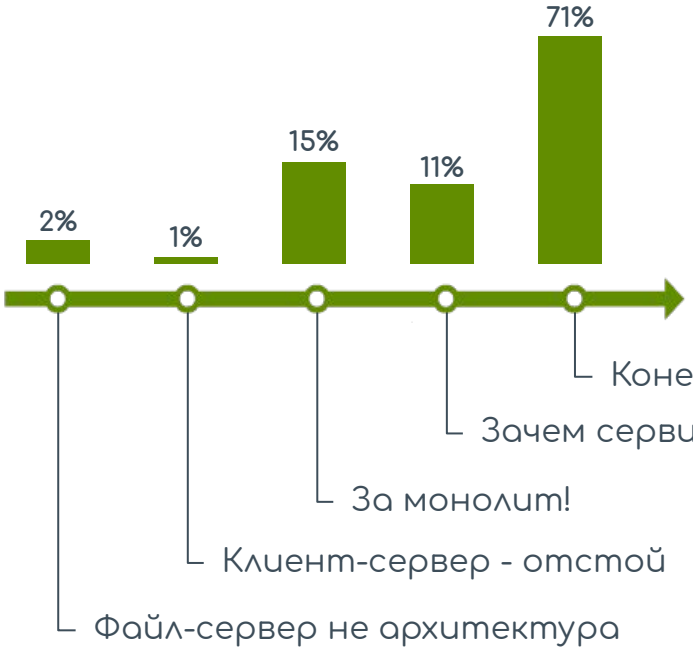
```
Метод : DELETE
Ресурс: person/{person_id}
Вход  : нет
Ответ : 204
Выход : нет
```

Функционал

- Принять идентификатор человека
- Проверить привилегии пользователя
- Удалить эталон из БД
- Вернуть OK

Выбор архитектуры для реализации системы

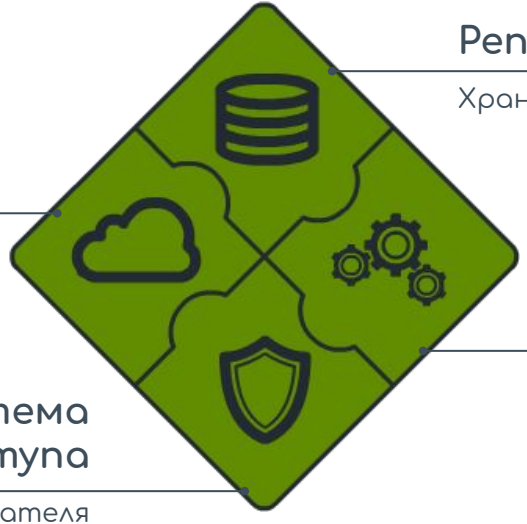
Выбор архитектуры для реализации системы



Выбор архитектуры для реализации системы

Микросервисы, так микросервисы

Состав сервисов целевой системы



Репозиторий

Хранение эталонных моделей

АПИ

Реалиация HTTP
Валидация параметров

Биометрический процессор

Построение моделей
Сравнение моделей

Подсистема разграничения доступа

Проверка привилегий пользователя

Схема взаимодействия сервисов системы

(Регистрация эталона человека)

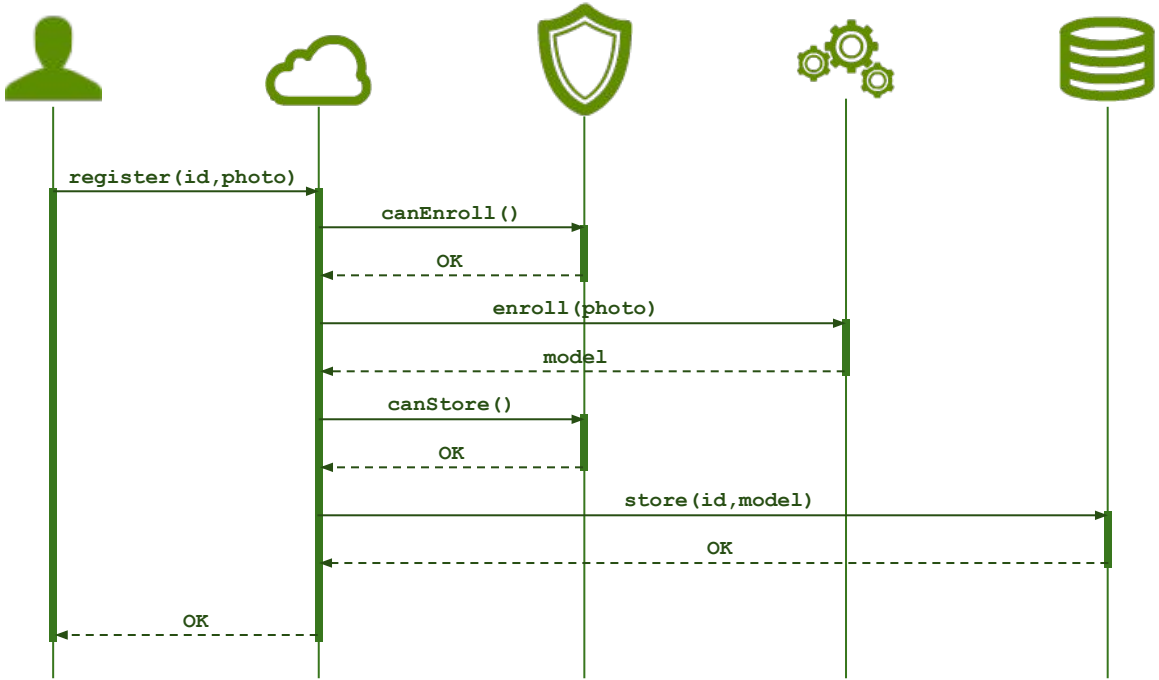


Схема взаимодействия сервисов системы

(Верификация человека по видео)

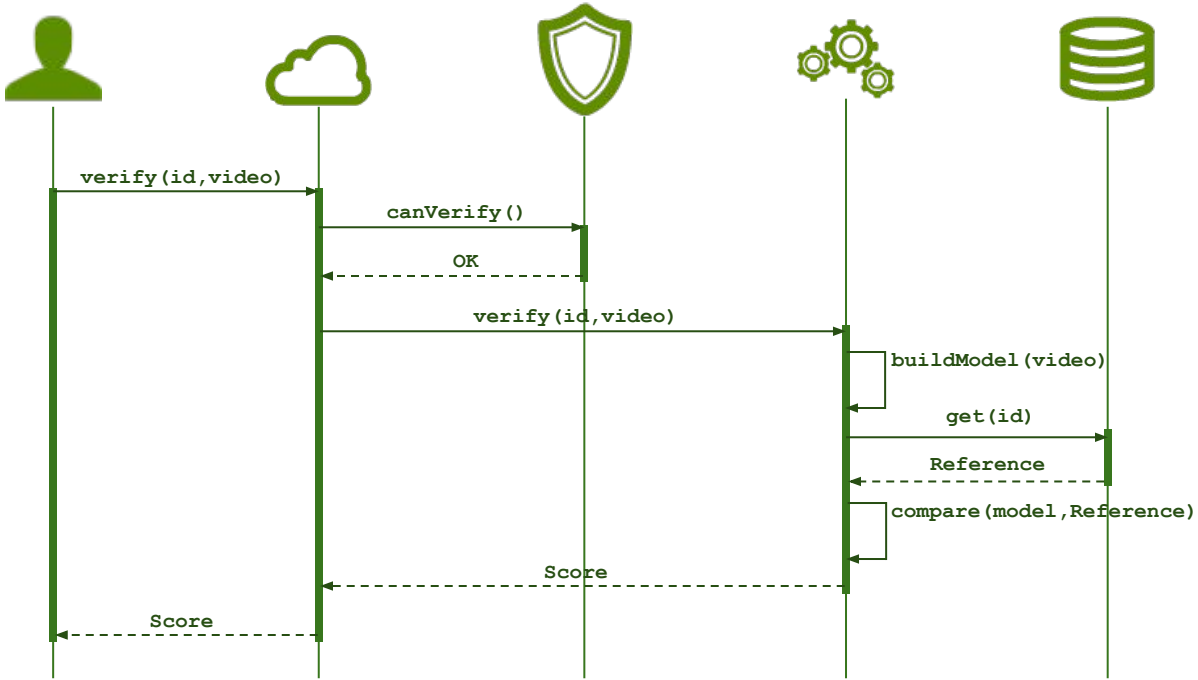
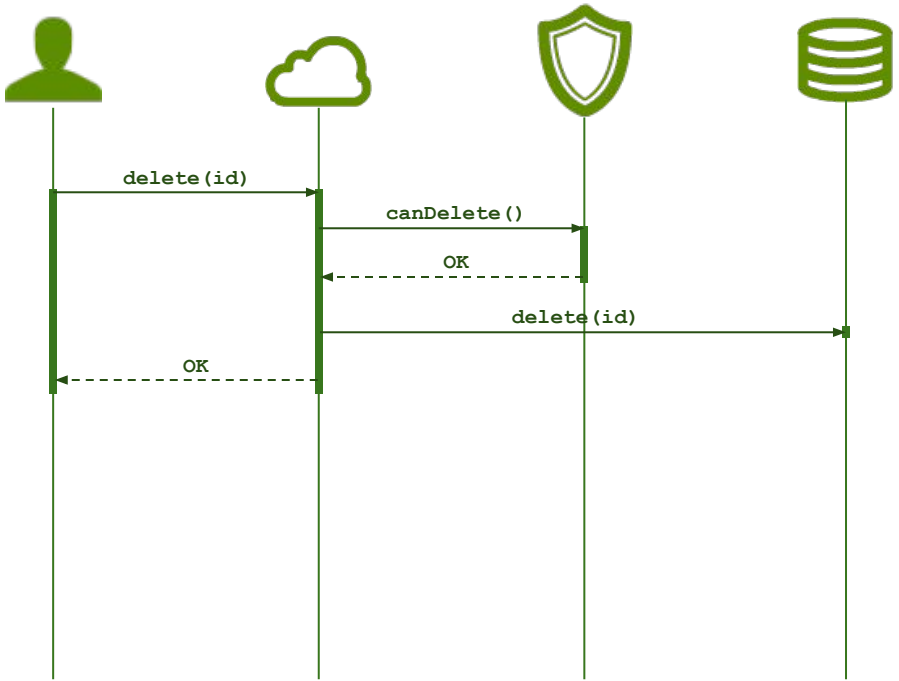


Схема взаимодействия сервисов системы

(Удаление эталона из БД)



Первичный функционал транспортной подсистемы

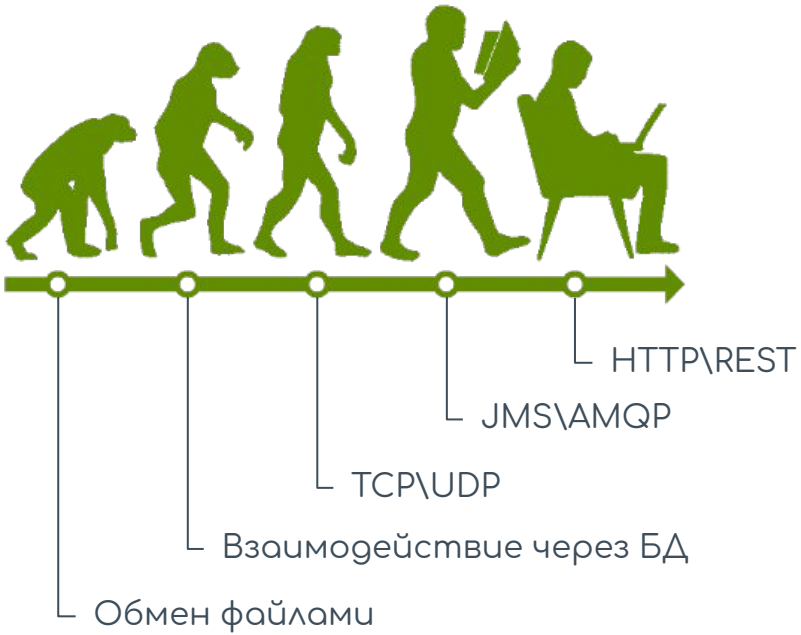
- Двусторонняя посылка сообщений
  самый распространенный случай
- Посылка бинарных данных (разного размера)
  при необходимости передать видео
- Односторонняя посылка сообщений
  когда не ждем подтверждения от репозитория

Выбор объекта транспортировки

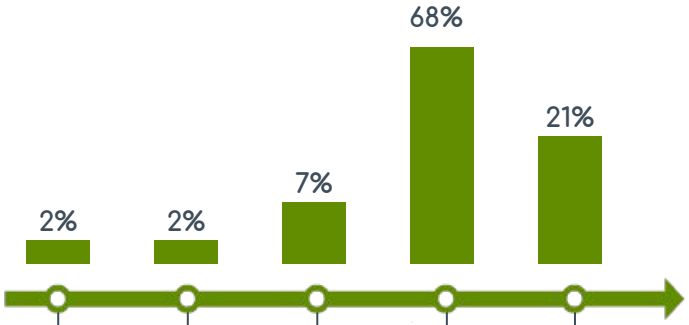
Передавать будем унифицированный объект-сообщение:

- Метаинформация
- Данные сообщения в сериализованном виде

Выбор средства транспортировки (транспортный протокол)

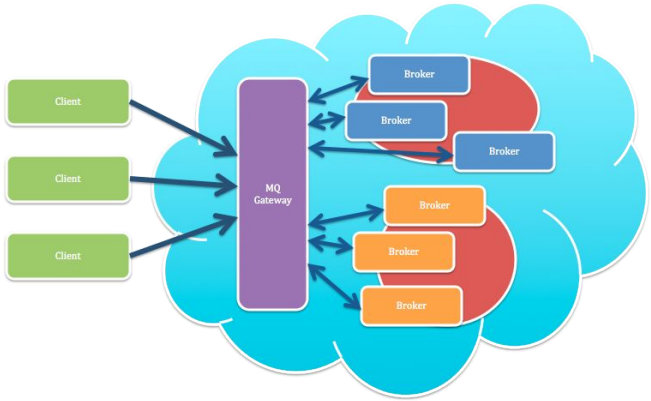


Выбор средства транспортировки (транспортный протокол)



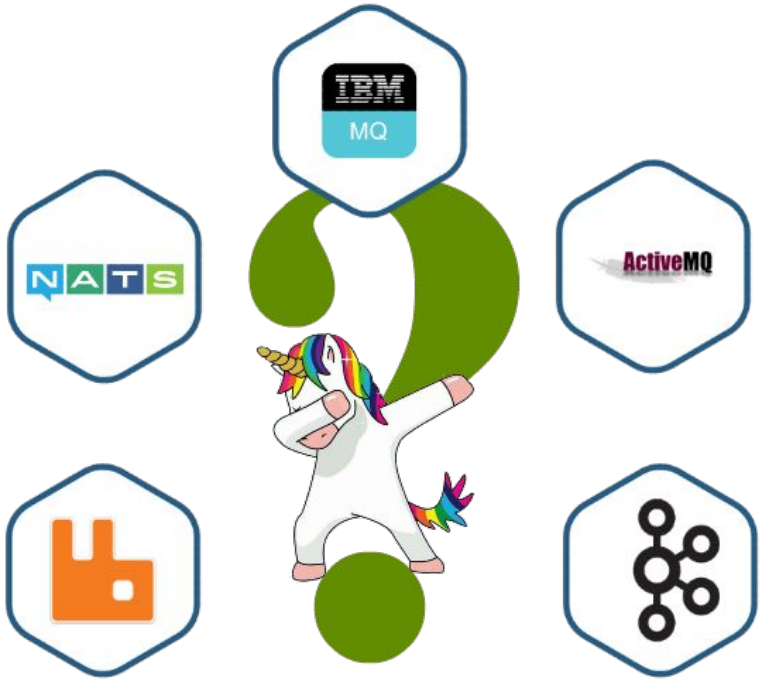
- Можно и по HTTP
- Все любят очереди! Конечно MQ!
- Больно хлопотно на уровне TCP\UDP писать
- Что значит ЧЕРЕЗ БД?!
- Как ФАЙЛАМИ?!

Выбор средства транспортировки (транспортный протокол)



Даешь MQ!

Какой брокер выбрать?



Какой брокер выбрать?

Правильно, НИКАКОЙ!

Вместо выбора конкретного провайдера выделим интерфейс:

```
public class Message {
   public MessageHeader header;
   public MessageData   data;
}

public interface IReceiver {
   void receive(Message message);
}

public interface ITransport {
   void post(Message message);
   void register(IReceiver receiver);
}
```

Реализация инфраструктуры транспортной подсистемы

невозможна без

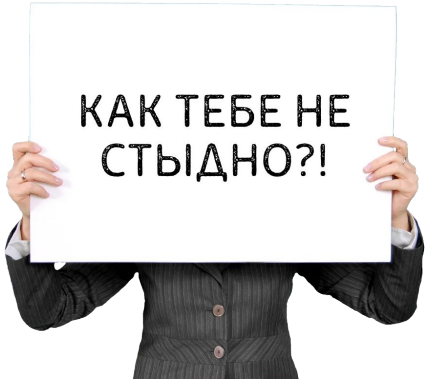
Реализация инфраструктуры транспортной подсистемы

(Поставщик требований #1: Заказчик)

Является поставщиком основных требований к разрабатываемой системе

Реализация инфраструктуры транспортной подсистемы

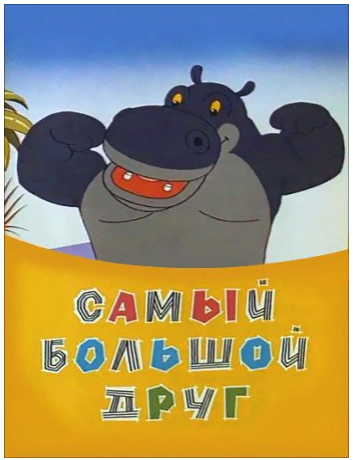
(Поставщик требований #2: Совесть)



Взывает к лучшим чертам разработчика, таким как профессионализм, ответственность и чувство прекрасного

Реализация инфраструктуры транспортной подсистемы

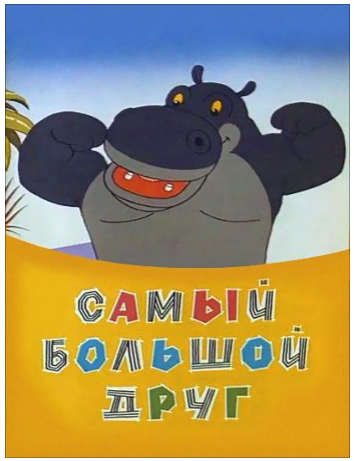
(Поставщик требований #3: Лень)



По совместительству лучший друг разработчика. По наблюдениям, является источников самых полезных фич

Реализация инфраструктуры транспортной подсистемы

(Поставщики требований для заказной разработки)



Реализация инфраструктуры транспортной подсистемы

(**Требование #1:** Односторонняя посылка сообщений)

Хочу чтобы:

- Можно было послать сообщение из одного сервиса в другой

Реализация инфраструктуры транспортной подсистемы

(Фича #1: Односторонняя посылка сообщений)

API_QUEUE

SECURITY_QUEUE

PROCESSOR_QUEUE

REPOSITORY_QUEUE

У каждого компонента своя очередь

Реализация инфраструктуры транспортной подсистемы

(**Фича #1:** Односторонняя посылка сообщений)

Message

Header:
- **Destination**
- **Type**

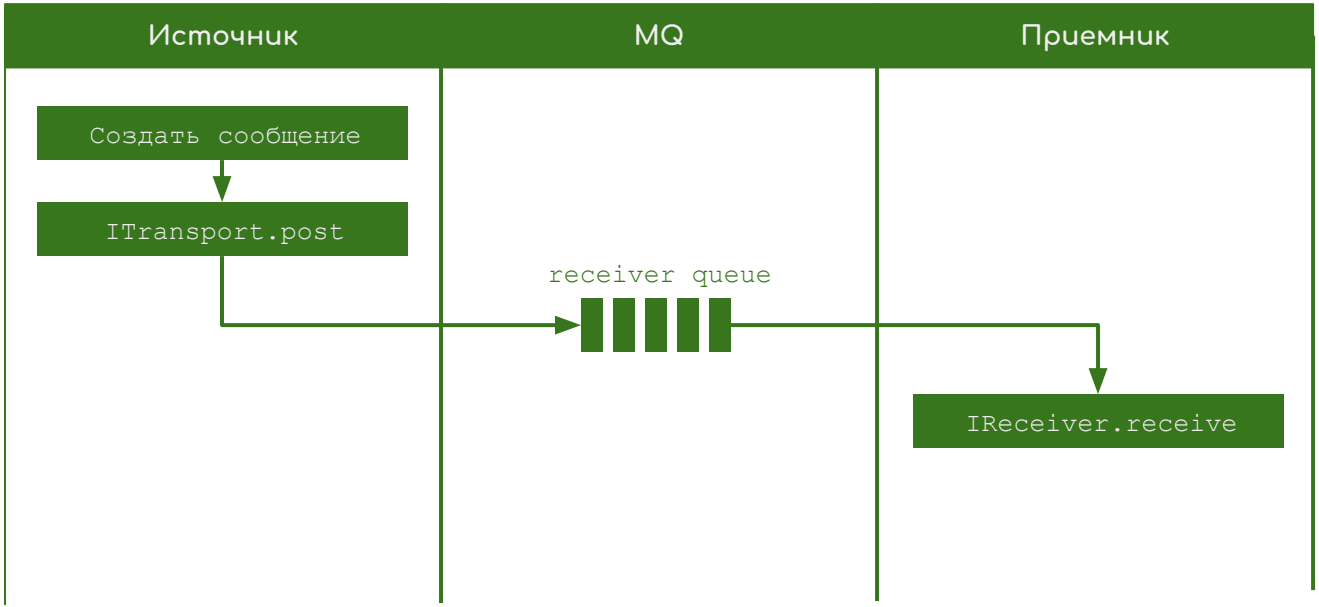
Data:
- **Data class**
- **Data JSON**

Структура сообщения:

1. Наименование очереди-приемника
2. Тип сообщения
3. Данные
   a. Наименование класса данных
   b. Объект, сериализованный в JSON

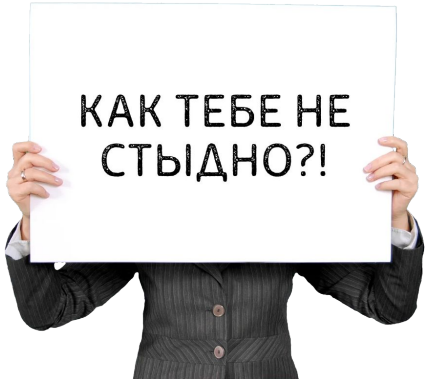
Реализация инфраструктуры транспортной подсистемы

(Фича #1: Односторонняя посылка сообщений)



Реализация инфраструктуры транспортной подсистемы

(**Требование #2:** Диспетчер сообщений)



Надо бы:

- Изолировать логику обработки одного сообщения от другого

Реализация инфраструктуры транспортной подсистемы

(Фича #2: Диспетчер сообщений)

Каждому сообщению свой обработчик:

```
public class Handler implements IReceiver {

  @Override
  public void receive(Message message) {
    //-- Process message
  }
}
```

Реализация инфраструктуры транспортной подсистемы

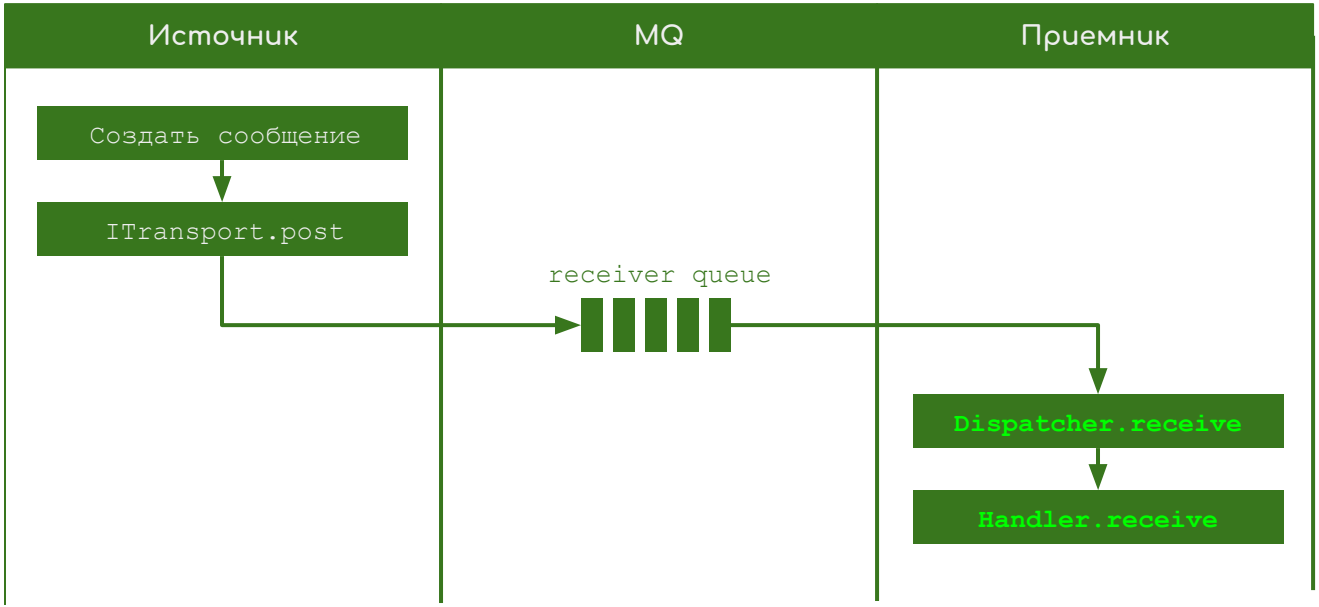
(Фича #2: Диспетчер сообщений)

Диспетчер сообщений:

```
public class Dispatcher implements IReceiver {

    private final Map<String, Handler> handlers = Maps.newHashMap();
    private final ExecutorService      executor = Executors.newFixedThreadPool(10);

    @Override
    public void receive(Message message) {
      String    messageType = message.header.type;
      Handler handler       = handlers.get(messageType);

      executor.submit(() -> handler.receive(message));
    }

    public void register(String messageType, Handler handler) {
      handlers.put(messageType, handler);
    }
}
```

Реализация инфраструктуры транспортной подсистемы

(**Фича #2:** Диспетчер сообщений)



Реализация инфраструктуры транспортной подсистемы

(**Требование #3:** Двусторонняя передача данных)

Хочу чтобы:

- Можно было и получить ответ от сервиса, которому послал сообщение

Реализация инфраструктуры транспортной подсистемы

(Фича #3: Двусторонняя передача данных)

Message

Header:
- Id
- Source
- IsOneWay
- Destination
- Type

Data:
- Data class
- Data JSON

Структура сообщения:

1. Добавляется уникальный идентификатор
2. Добавлятся очередь-источник
3. Добавляется признак однонаправленного сообщения

Реализация инфраструктуры транспортной подсистемы

(Фича #3: Двусторонняя передача данных)

Изменяется обработчик:

```
public abstract class Handler {

  public abstract Message handle(Message message);
}
```

Реализация инфраструктуры транспортной подсистемы

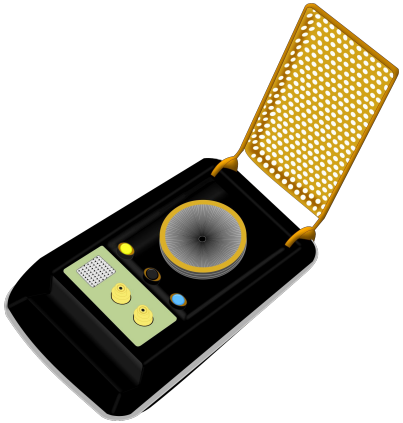
(Фича #3: Двусторонняя передача данных)

Диспетчер сообщений:

```
public class Dispatcher implements IReceiver {

  private final ITransport transport;

  @Override
  public void receive(Message message) {
    String    messageType = message.header.type;
    Handler handler          = handlers.get(messageType);

    executor.submit(() -> handle(handler, message));
  }

  private void handle(Handler handler, Message message) {
    Message response = handler.handle(message);

    if (!message.header.isOneWay) {
      transport.post(response);
    }
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #3: Двусторонняя передача данных)

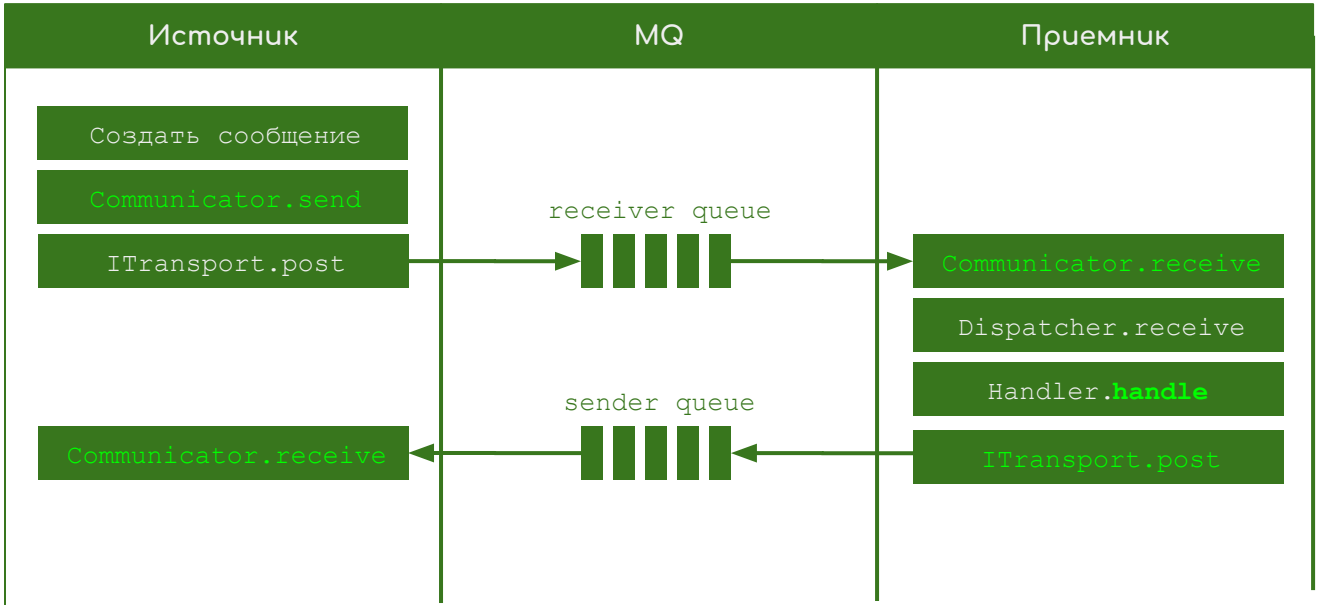


Создаем объект-коммуникатор:

```
public class Communicator implements IReceiver {

  private final ITransport                                transport;
  private final Dispatcher                                dispatcher;
  private final Map<UUID, SettableFuture<Message>> awaitingResponse = …;

  …

  public Message send(Message message) {
    SettableFuture<Message> future = SettableFuture.create();

    awaitingResponse.put(message.header.id, future);
    transport.post(message);

    return getMessage(future);
  }

  public void receive(Message message) {
    if (awaitingResponse.containsKey(message.header.id)) {
      awaitingResponse.get(message.header.id).set(message);
    } else {
      dispatcher.receive(message);
    }
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #3: Двусторонняя передача данных)



Реализация инфраструктуры транспортной подсистемы

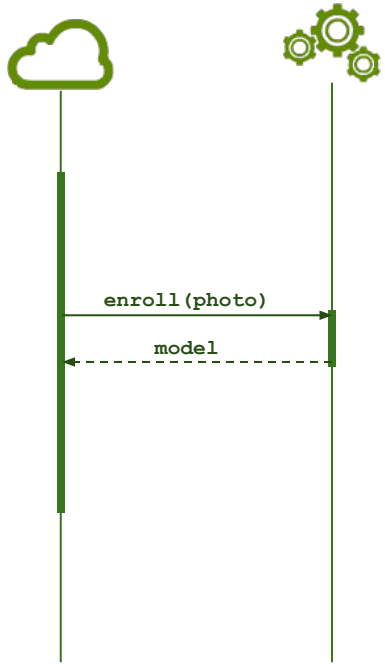
(**Требование #4:** Передача бинарных данных)

Хочу чтобы:

- Можно было передавать не только JSON сообщения, но и картинки, звуки, видео и другие бинарные данные

Реализация инфраструктуры транспортной подсистемы

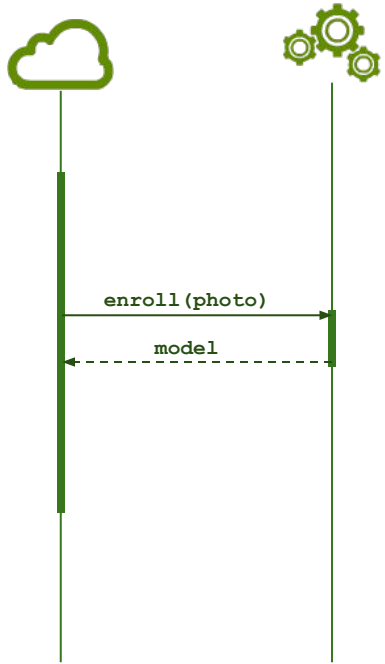
(Пример использования реализованного функционала: АПИ)




```
public Model enroll(Communicator communicator, byte[] image) {

  Message request = new Message();

  request.header.id              = UUID.randomUUID();
  request.header.destination = "PROCESSOR_QUEUE";
  request.header.isOneWay   = false;
  request.header.source        = "API_QUEUE";
  request.header.type           = "ENROLL_MESSAGE";

  request.data.type              = image.getClass().getName();
  request.data.json              = toJson(image);

  Message response   = communicator.send(request);
  Class       dataClass = classForName(response.data.type);

  return (Model)fromJson(response.data.json, dataClass);
}
```

Реализация инфраструктуры транспортной подсистемы

(Пример использования реализованного функционала: Обработчик)




```
public class EnrollHandler extends Handler {

  @Override
  public Message handle(Message message) {
     Byte[]  image = jsonToBytes(message.data.json);
     Model  model = imageToModel(image);

     message.data.type = model.getClass().getName();
     message.data.json = toJson(model);

     return message;
  }

  private Model imageToModel(byte[] image) {
     //-- Image to model transformation logic here
  }
}
```

Реализация инфраструктуры транспортной подсистемы

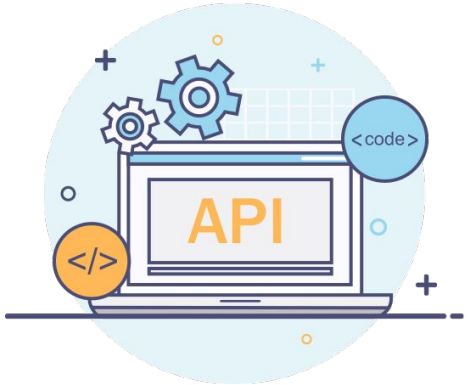
(Требование #4: Реализация АПИ компонента)

А давай как-нибудь так чтобы:

1. Не надо было каждое сообщение руками составлять...
2. Потом отправлять...
3. Потом ответ парсить...
4. И вообще, чтобы было похоже на вызов метода... А?

Реализация инфраструктуры транспортной подсистемы

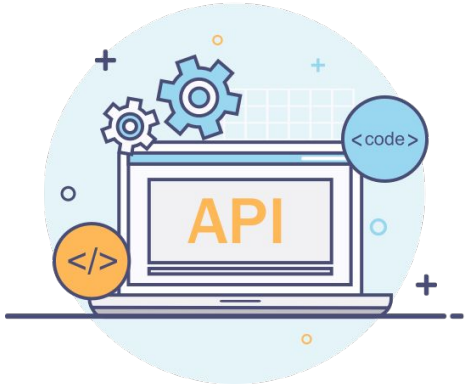
(Фича #4: Реализация АПИ компонента)




```
public class Api {
   private final Communicator communicator;
   private final String       source;
   private final String       destination;

   public Api(Communicator communicator, String source, String destination) {
      this.communicator = communicator;
      this.source = source;
      this.destination = destination;
   }

   protected <D, R> R send(String messageType, D data) {
      Message request = createMessage(); //-- id, source, destination

      request.header.isOneWay = false;
      request.header.type = messageType;

      request.data.type = data.getClass().getName();
      request.data.json = toJson(data);

      Message response  = communicator.send(request);
      Class   dataClass = classForName(response.data.type);

      return (R) fromJson(response.data.json, dataClass);
   }
}
```

Реализация инфраструктуры транспортной подсистемы

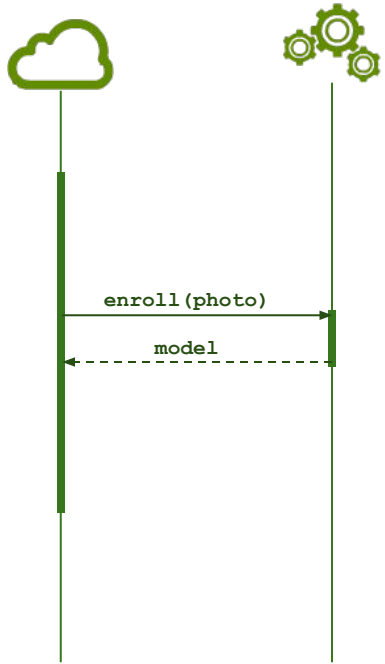
(Фича #4: Реализация АПИ компонента)




```
public class ProcessorApi extends Api {

  public ProcessorApi(Communicator communicator, String source) {
    super(communicator, source, "PROCESSOR_QUEUE");
  }

  public Model enroll(byte[] image) {
    return send("ENROLL_MESSAGE", image);
  }

  public Score verify(String personId, byte[] video) {
    return send("VERIFY_MESSAGE", toDto(personId, video));
  }

  private Object toDto(String personId, byte[] video) {
    //-- Verification DTO creation logic here
   }
}
```

Реализация инфраструктуры транспортной подсистемы

(Пример использования реализованного функционала: АПИ)




```
public Model enroll(Communicator communicator, byte[] image) {
   ProcessorApi api     = new ProcessorApi(communicator,"API_QUEUE");
   Model           model = api.enroll(image);

   return model;
}
```

Реализация инфраструктуры транспортной подсистемы

(Требование #5: Нужен базовый обработчик)

А давай как-нибудь так чтобы:

1. На стороне обработчика не надо было данные десериализовать...
2. Потом сериализовать... А?

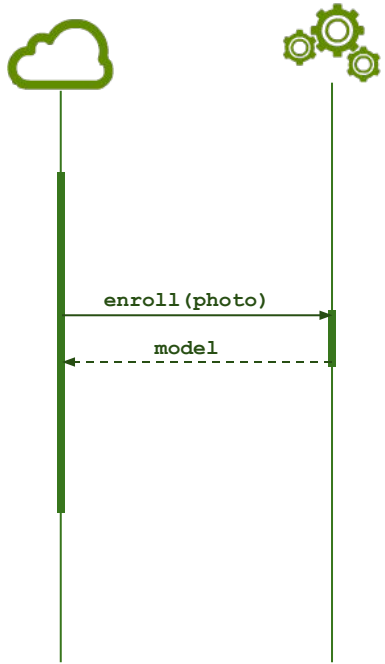
Реализация инфраструктуры транспортной подсистемы

(Фича #5: Реализация базового обработчика)

```
public abstract class BaseHandler<D, R> extends Handler {

  @Override
  public Message handle(Message message) {
    Class<D> type   = classForName(message.data.type);
    D        data   = fromJson(message.data.json, type);
    R        result = processData(data);

    message.data.type = result.getClass().getName();
    message.data.json = toJson(result);

    return message;
  }

  protected abstract R processData(D data);
}
```

Реализация инфраструктуры транспортной подсистемы

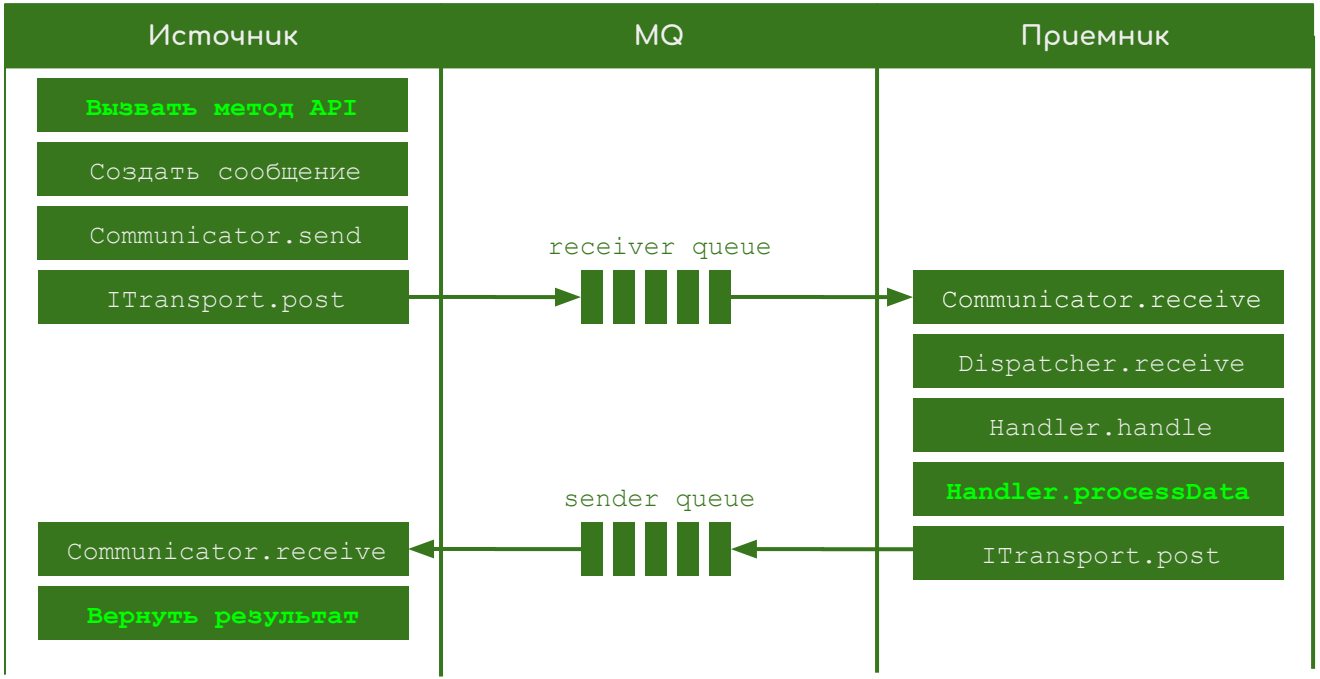
(Пример использования реализованного функционала: Обработчик)




```
public class EnrollHandler extends BaseHandler<byte[], Model> {

  @Override
  protected Model processData(byte[] image) {
     return imageToModel(image);
  }

  private Model imageToModel(byte[] image) {
     //-- Image to model transformation logic here
  }
}
```

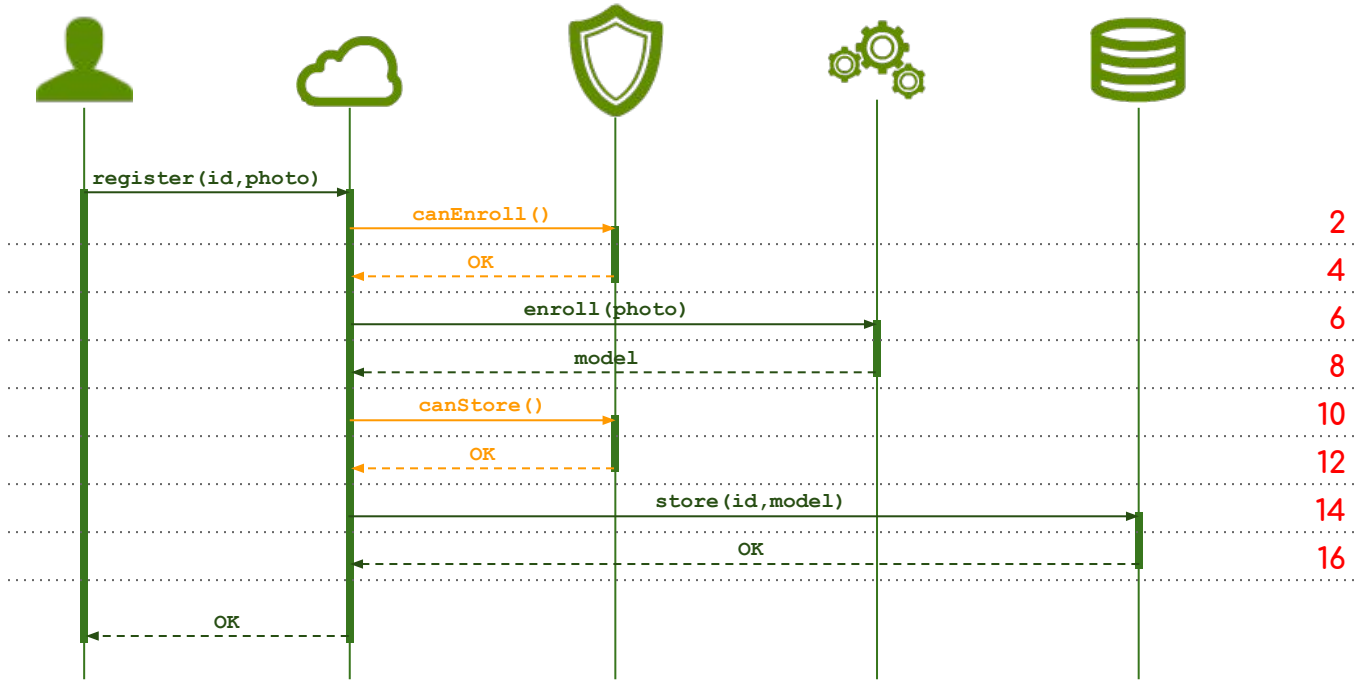
Реализация инфраструктуры транспортной подсистемы

(Основные этапы работы транспортной подсистемы)



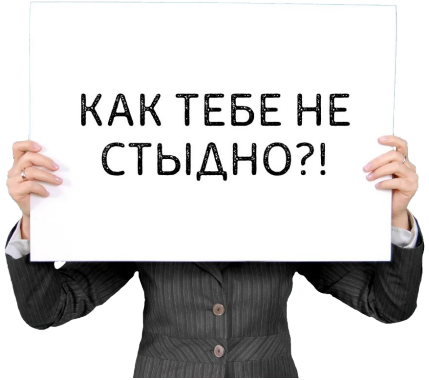
Реализация инфраструктуры транспортной подсистемы

(Проблема “лишних вызовов”)



Реализация инфраструктуры транспортной подсистемы

(Требование #6: Более сложные маршруты сообщений)



Надо бы:

- Оптимизировать количество шагов для каждого из сообщений и для процесса
- Уметь создавать более сложные маршруты

Реализация инфраструктуры транспортной подсистемы

(Фича #6: Реализация маршрутов)

Маршрут:

- Состоит из списка точек
- У каждой точки своя очередь
- У точки есть статус (посещена/нет)

```
public class PathPoint {
  public final String     queue;
  public        boolean visited;

  public PathPoint(String queue) {
    this.queue = queue;
    this.visited = false;
  }
}

public class Path extends ArrayList<PathPoint> {
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #6: Реализация маршрутов)

Message

Header:
- **Id**
- **Source**
- **IsOneWay**
- **Path**
- **Type**

Data:
- **Data class**
- **Data JSON**

Структура сообщения:

1. Исчезает наименование очереди-приемника
2. Добавлятся маршрут сообщения

Реализация инфраструктуры транспортной подсистемы

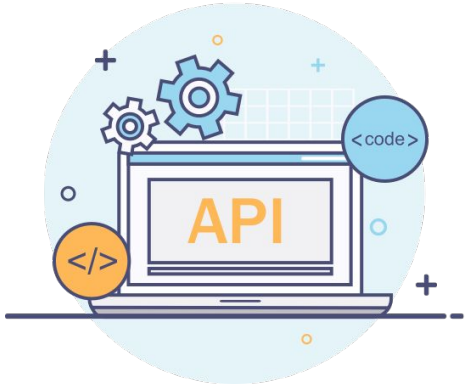
(Фича #6: Реализация маршрутов)

Диспетчер сообщений:

```
public class Dispatcher implements IReceiver {

  private final ITransport                transport;
  private final Map<String, Handler> handlers = ...;
  private final ExecutorService          executor = ...;

  @Override
  public void receive(Message message) {
     String    messageType = message.header.type;
     Handler handler          = handlers.get(messageType);

     executor.submit(() -> handle(handler, message));
  }

  private void handle(Handler handler, Message message) {
     Message response = handler.handle(message);

     if (!isFinished(response.header.path) || !message.header.isOneWay) {
        transport.post(response);
     }
  }
}
```

Реализация инфраструктуры транспортной подсистемы

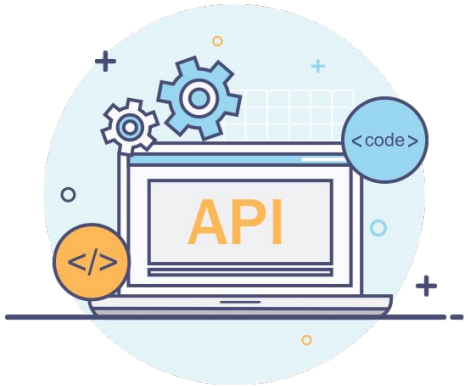
(Фича #6: Реализация маршрутов)




```
public class Api {
  private final Communicator communicator;
  private final String       source;

  public Api(Communicator communicator, String source) {
    this.communicator = communicator;
    this.source = source;
  }

  protected <D, R> R send(String messageType, D data, String... points) {
    Message request = createMessage(points); //-- id, source, path
    …
    return (R) fromJson(response.data.json, dataClass);
  }

  private Message createMessage(String... points) {
    Message result = new Message();

    result.header.id = UUID.randomUUID();
    result.header.source = source;
    result.header.path = createPath(points);

    return result;
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #6: Реализация маршрутов)




```
public class ProcessorApi extends Api {

  private final String pQueue = "PROCESSOR_QUEUE";
  private final String sQueue = "SECURITY_QUEUE";

  public ProcessorApi(Communicator communicator, String source) {
    super(communicator, source);
  }

  public Model enroll(byte[] image) {
    return send("ENROLL_MESSAGE", image, pQueue, sQueue);
  }

  public Model verify(String personId, byte[] video) {
    return send("VERIFY_MESSAGE", toDto(personId, video), pQueue, sQueue);
  }

  private Object toDto(String personId, byte[] video) {
    //-- Verification DTO creation logic here
    return null;
  }
}
```

Реализация инфраструктуры транспортной подсистемы

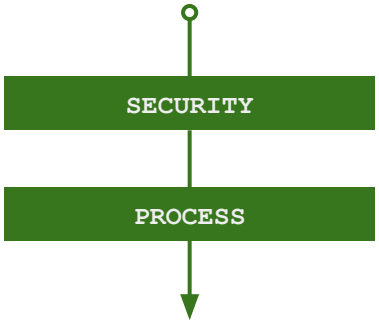
(Требование #7: Автоматическая маршрутизация сообщений)

А давай как-нибудь так чтобы:

1. Не знать очереди на которые надо посылать сообщения...
2. Пусть они сами как-нибудь определяются... А?

Реализация инфраструктуры транспортной подсистемы

(Фича #7: Автоматическая маршрутизация сообщений)

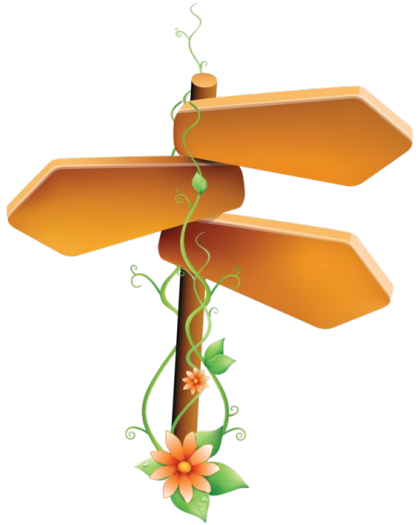


Шаблон маршрута

1. Описывает желаемый маршрут сообщения в виде списка шагов
2. Каждый шаг имеет свою семантику (обработка, проверка прав доступа, кэширование и т.д.)
3. Отвязывает нас от конкретных имен очередей

Реализация инфраструктуры транспортной подсистемы

(Фича #7: Автоматическая маршрутизация сообщений)



Маршрутизатор

1. Хранит в себе информацию о том, какими очередями какие сообщения и в какой момент обслуживаются
2. Выдает по запросу маршрут для сообщения;
3. У него выделенная очередь про которую все знают

Реализация инфраструктуры транспортной подсистемы

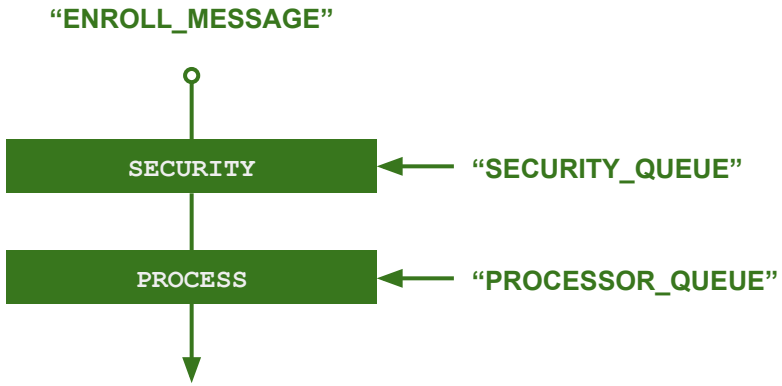
(Фича #7: Автоматическая маршрутизация сообщений)

```
{
  "queue": "PROCESSOR_QUEUE",
  "steps": [
    {
      "step": "PROCESS",
      "messages": [
        "ENROLL_MESSAGE",
        "VERIFY_MESSAGE"
      ]
    }
  ]
}
```

```
{
  "queue": "SECURITY_QUEUE",
  "steps": [
    {
      "step": "PROCESS",
      "messages": [
        "CAN_ENROLL_MESSAGE",
        "CAN_VERIFY_MESSAGE"
      ]
    },
    {
      "step": "SECURITY",
      "messages": []
    }
  ]
}
```

Реализация инфраструктуры транспортной подсистемы

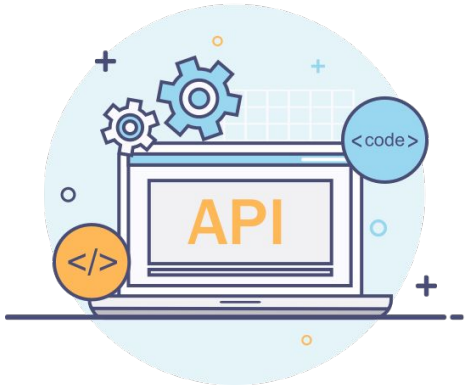
(Фича #7: Автоматическая маршрутизация сообщений)



Заполнение маршрута определенными очередями из конфигурации Маршрутизатора

Реализация инфраструктуры транспортной подсистемы

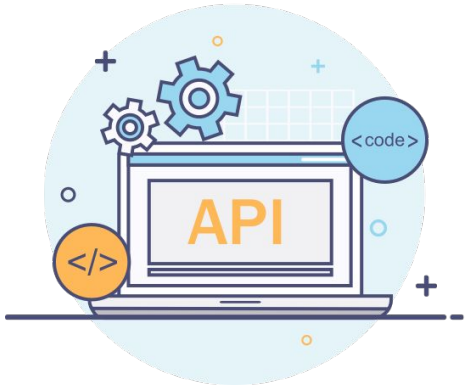
(Фича #7: Автоматическая маршрутизация сообщений)




```
public class Api {
  private final Communicator communicator;
  private final String       source;

  private final Map<String, Path> paths = …;
  ...

  private Message createMessage(String messageType, String... points) {
     Message result = new Message();

     result.header.id = UUID.randomUUID();
     result.header.source = source;
     result.header.path = paths.computeIfAbsent(messageType, t -> createPath(t, points));

     return result;
  }

  private Path createPath(String messageType, String... steps) {
     Message request  = createRouterMessage(messageType, steps);
     Message response = communicator.send(request);

     return extractPath(response);
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #7: Автоматическая маршрутизация сообщений)




```
public class ProcessorApi extends Api {

  private final String[] steps = {"SECURITY", "PROCESS"};

  public ProcessorApi(Communicator communicator, String source) {
    super(communicator, source);
  }

  public Model enroll(byte[] image) {
    return send("ENROLL_MESSAGE", image, steps);
  }

  public Model verify(String personId, byte[] image) {
    return send("VERIFY_MESSAGE", toDto(personId, image), steps);
  }

  private Object toDto(String personId, byte[] image) {
    //-- Dto logic creation here
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #7: Автоматическая маршрутизация сообщений)

Обработчик запроса на регистрацию эталона:

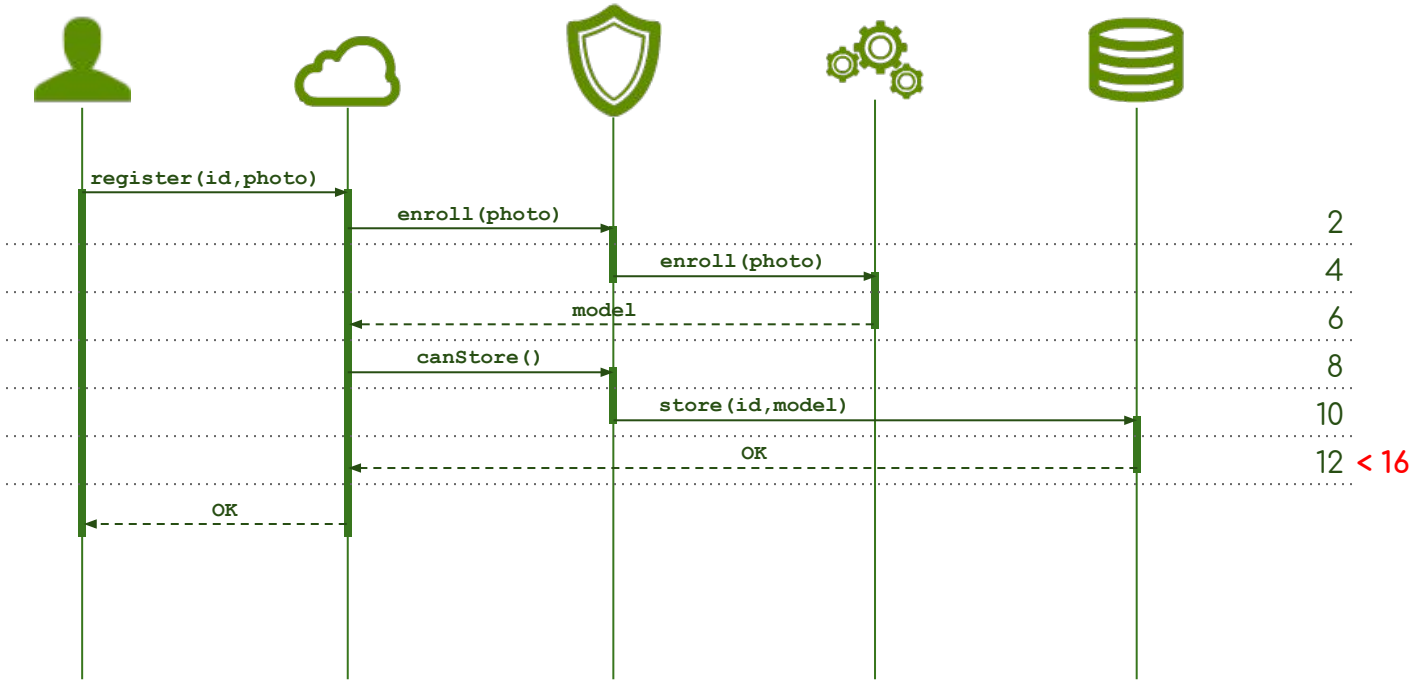
```
public class EnrollHandler extends BaseHandler<byte[], Model> {

  @Override
  protected String getStep() {
    return "PROCESS";
  }
  ...
}
```

Обработчик подсистемы разграничения доступа:

```
public class SecurityHandler extends Handler {

  protected String getStep() {
    return "SECURITY";
  }

  public Message handle(Message message) {
    if (checkPrivilege(message.header)) {
      return message;
    } else throw new SecurityException();
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Итог решения проблемы “лишних вызовов”)



Файл конфигурации маршрутов для небольшой системы

Реализация инфраструктуры транспортной подсистемы

(Требование #8: Автоматическая регистрация маршрутов)

А давай как-нибудь так чтобы:

1. Маршрутизатор создавался с пустой конфигурацией маршрутов...
2. Компоненты при старте сами регистрировались в маршрутизаторе...
3. При повторной перерегистрации компонента обновлялись сведения о нем... А?

Реализация инфраструктуры транспортной подсистемы

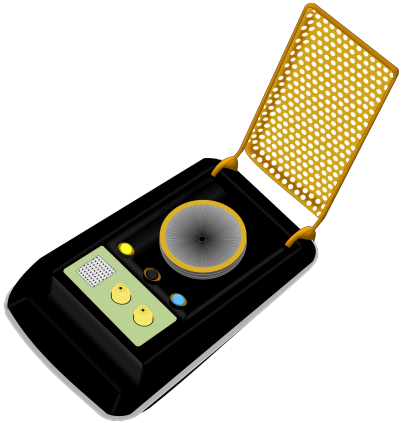
(Фича #8: Автоматическая регистрация маршрутов)

Диспетчер сообщений:

```
public class Step {
   public String   step;
   public String[] messages;
}

public class StepsDescriptor {
   public String queue;
   public Step[] steps;
}

public class Dispatcher implements IReceiver {
   ...
   public StepsDescriptor generateStepsDescriptor() {
      //-- Iterate through registered handlers and create StepsDescriptor
   }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #8: Автоматическая регистрация маршрутов)

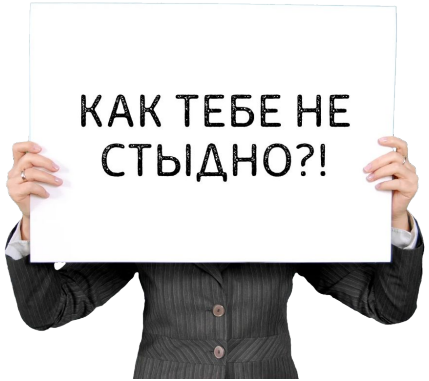


Коммуникатор:

```
public class Communicator implements IReceiver {
  …
  public void start() {
    StepsDescriptor descriptor  = dispatcher.generateStepsDescriptor();
    Message         request     = generateStepsRequest(descriptor);
    boolean         registered = false;

    while (!registered) {
      registered = extractBoolResult(send(request));
    }
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Требование #9: Обработка исключений)



Надо бы:

- Чтобы при возникновении исключения посередине маршрута сообщение не шло дальше, а возвращалось отправителю
- На стороне отправителя при этом возникало исключение с информацией о том, что случилось

Реализация инфраструктуры транспортной подсистемы

(Фича #9: Обработка исключений)

Описываем базовое исключение

```
public enum ErrorCode {
  BAD_REQUEST,
  NOT_FOUND,
  INTERNAL_SERVER_ERROR,
  NO_RESPONSE,
  FORBIDDEN
}

public class BaseException extends RuntimeException {

  public final ErrorCode errorCode;
  public final String       reason;

  public BaseException(ErrorCode errorCode, String reason, String message) {
    super(message);

    this.errorCode = errorCode;
    this.reason = reason;
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #9: Обработка исключений)

Message

Header:
- **Id**
- **Source**
- **IsOneWay**
- **Path**
- **Type**

Data:
- **Data class**
- **Data JSON**

Структура сообщения:

1. Не меняется
2. Добавляется особый тип сообщения “**EXCEPTION_MESSAGE**”
3. В поле **Data** кладется сериализованное базовое исключение

Реализация инфраструктуры транспортной подсистемы

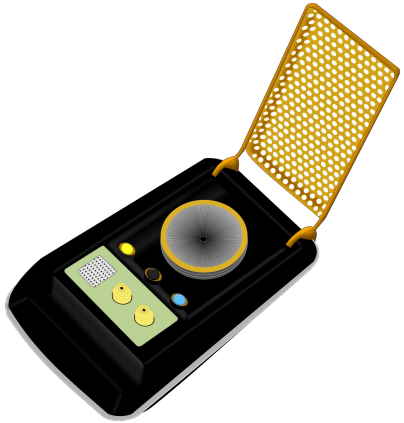
(Фича #9: Обработка исключений)

Диспетчер сообщений:

```
public class Dispatcher implements IReceiver {

  private void handle(Handler handler, Message message) {
    try {
      Message response = handler.handle(message);
      ...
    } catch (Throwable ex) {
      transport.post(generateExceptionMessage(message, ex));
    }
  }
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #9: Обработка исключений)

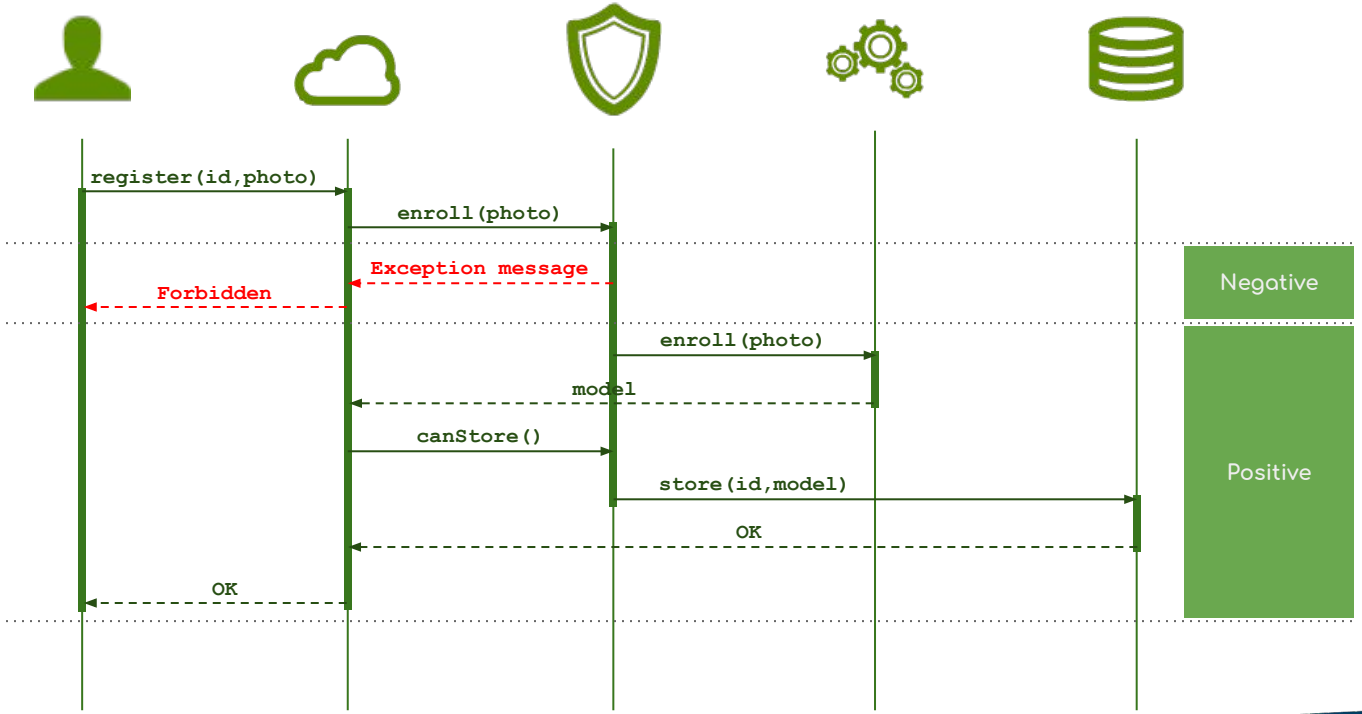


Коммуникатор

```
public class Communicator implements IReceiver {
  ...
  public Message send(Message message) {
    SettableFuture<Message> future = SettableFuture.create();

    awaitingResponse.put(message.header.id, future);
    transport.post(message);

    Message response = getMessage(future);

    if (response.header.type.equals("EXCEPTION_MESSAGE")) {
      throw extractException(message.data);
    } else {
      return response;
    }
  }
  ...
}
```

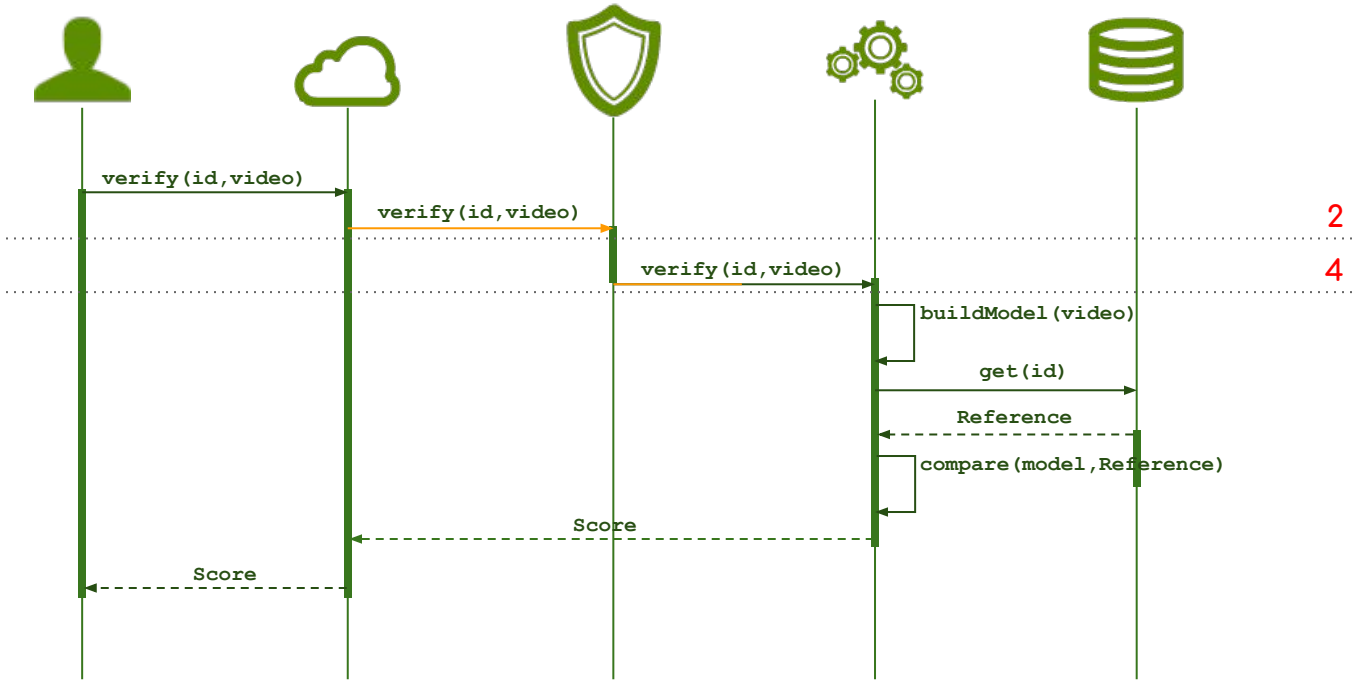
Реализация инфраструктуры транспортной подсистемы

(Итог решения задачи обработки исключений)



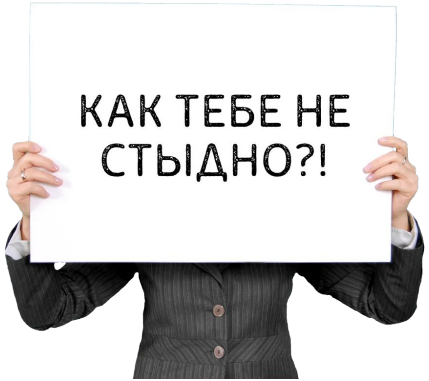
Реализация инфраструктуры транспортной подсистемы

(Излишняя передача бинарных данных: Nx1,3x4)



Реализация инфраструктуры транспортной подсистемы

(Требование #10: **Оптимизация** передачи бинарных данных)



Надо бы:

- Чтобы бинарные данные путешествовали по сети минимально необходимое количество раз
- Не надо было их переводить в base64 и наоборот

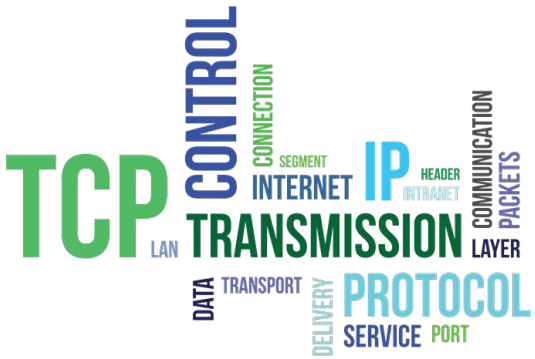
Реализация инфраструктуры транспортной подсистемы

(**Требование #4:** Передача бинарных данных)

А я говорил...

Реализация инфраструктуры транспортной подсистемы

(Фича #10: Передача больших бинарных данных)

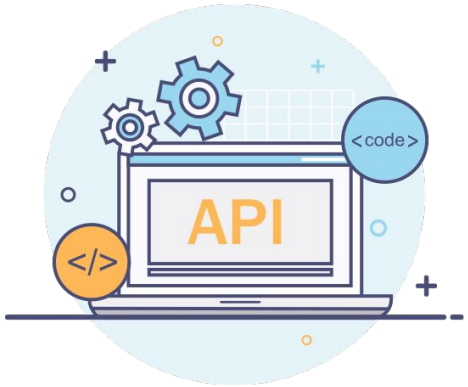


TCP инфраструктура:

```
public class TransmissionEndpoint {
   public String host;
   public int     port;
}

public class TransmissionDescriptor {
   public UUID                       id;
   public TransmissionEndpoint endpoint;
}

public interface ITcpDataHub {

   TransmissionDescriptor transmit(byte[] data);

   byte[] receive(TransmissionDescriptor descriptor);
}
```

Реализация инфраструктуры транспортной подсистемы

(Фича #10: Передача больших бинарных данных)




```
public class Api {
  private final ITcpDataHub  dataHub;
  ...
  public Api(Communicator communicator, ITcpDataHub dataHub, String source) {
     this.source = source;
     ...
  }

  protected <D, R> R send(String messageType, D data, String... points) {
     Message request = createMessage(messageType, points); //-- id, source, path

     request.header.isOneWay = false;

     if (data instanceof byte[]) {
        request.data = toMessageData(dataHub.transmit((byte[]) data));
     } else {
        request.data = toMessageData(data);
     }

     Message response  = communicator.send(request);
     Class   dataClass = classForName(response.data.type);

     return (R) fromJson(response.data.json, dataClass);
  }
}
```

Реализация инфраструктуры транспортной подсистемы

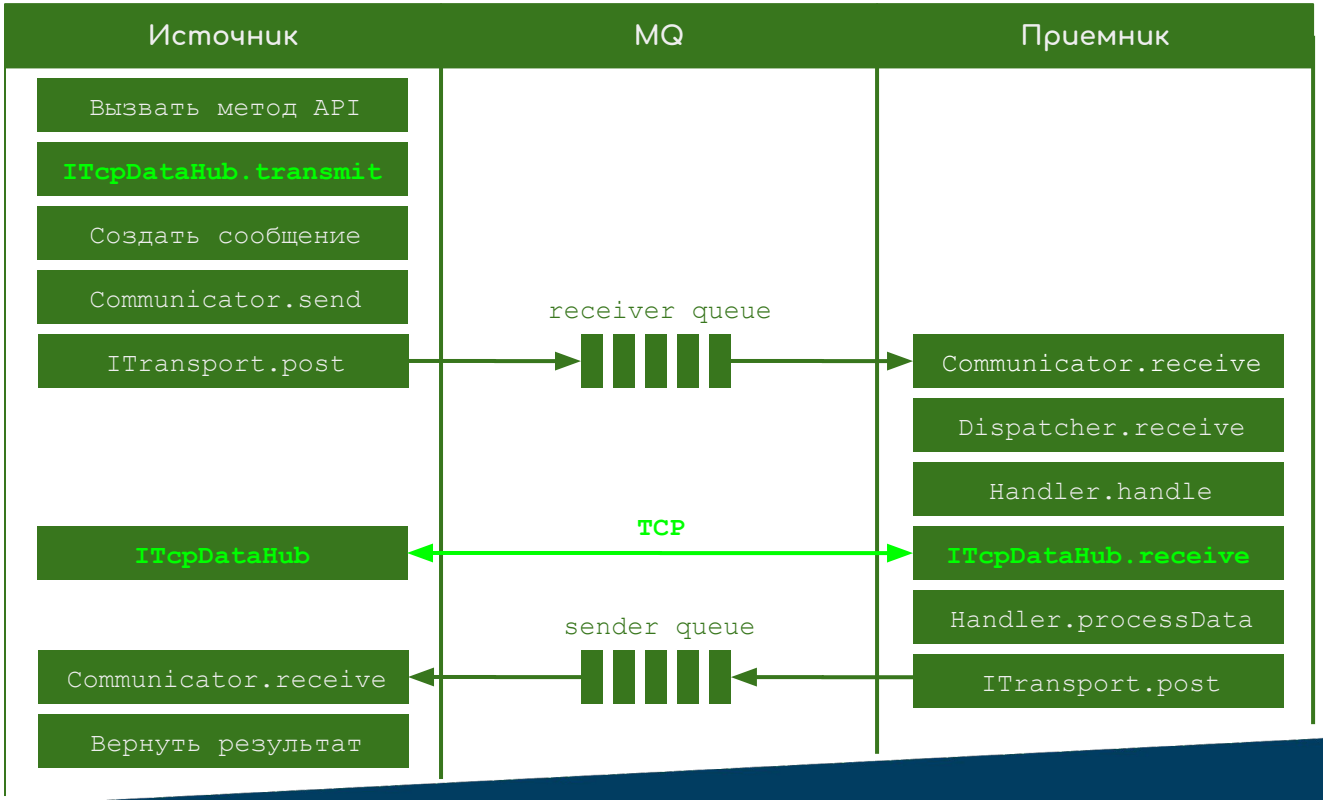
(Фича #10: Передача больших бинарных данных)

Базовый обработчик сообщений данных:

```
public abstract class BaseHandler<D, R> extends Handler {

  protected final ITcpDataHub dataHub;

  public BaseHandler(ITcpDataHub dataHub) {
    this.dataHub = dataHub;
  }

  @Override
  public Message handle(Message message) {
    D data  = extractData(message.data);
    ...
    return message;
  }

  private D extractData(MessageData messageData) {
    if (messageData.type.equals(TransmissionDescriptor.class.getName())) {
      return (D) dataHub.receive(extractDescriptor(messageData));
    } else {
      return (D) fromJson(messageData.json, classForName(messageData.type));
    }
  }
}
```

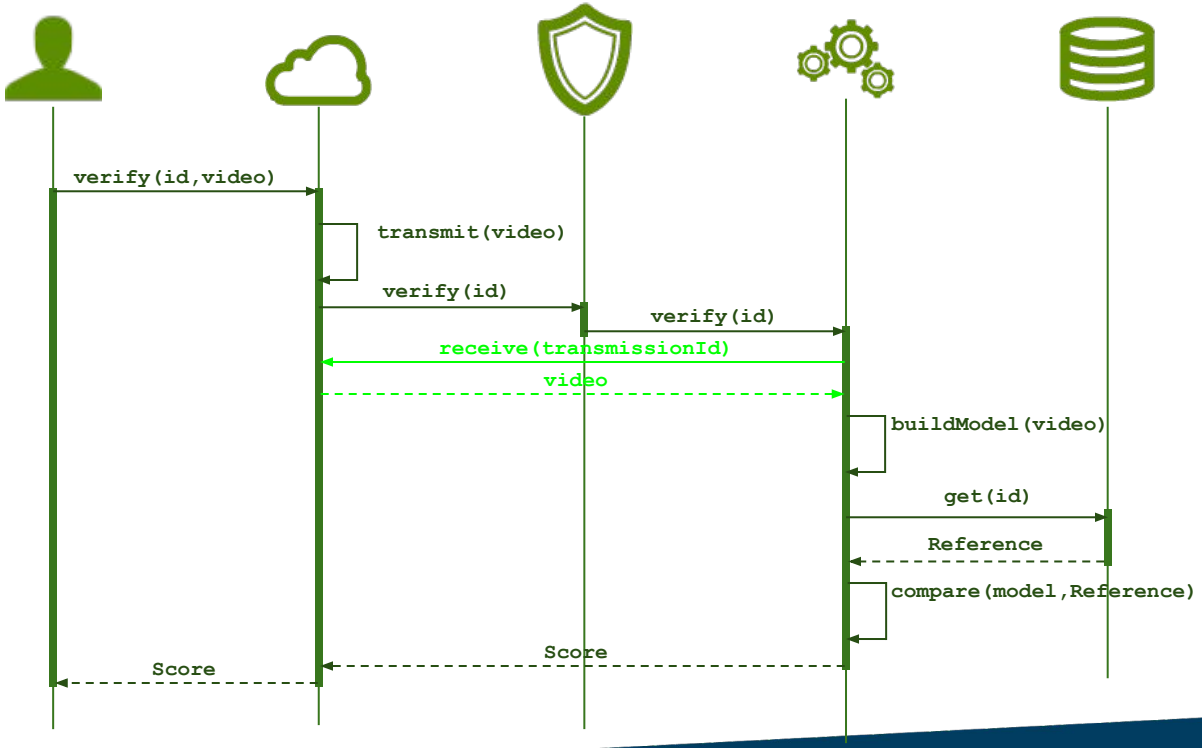
Реализация инфраструктуры транспортной подсистемы

(Фича #10: Передача больших бинарных данных)



Реализация инфраструктуры транспортной подсистемы

(Излишняя передача бинарных данных)



Спасибо за внимание!

1. Управление таймаутами сообщений
2. Асинхронная посылка сообщений
3. Произвольные маршруты сообщений
4. Балансировка
5. Отказоустойчивость
6. Широковещательная посылка сообщений
7. Синхронизация состояния