# Документация системы “Аэропорт”

Александр - самолеты

Антон - пассажиры

Дарья - визуализация

Дмитрий - багаж, расписание

Наталья - автобусы

Никита - follow-me, заправка

Сергей - УНД

### RabbitMQ

URL для работы из программы: amqp://aazhpoyi:wl3G3Fu\_s88DNK0Fr0N9XxsUBxmlzUcK@[duckbill.rmq.cloudamqp.com/aazhpoyi](http://duckbill.rmq.cloudamqp.com/aazhpoyi)

Web-интерфейс для тестирования:

<https://duckbill.rmq.cloudamqp.com/>

Логин: aazhpoyi

Пароль: wl3G3Fu\_s88DNK0Fr0N9XxsUBxmlzUcK

### GitHub

Ссылка на репозиторий: https://github.com/SpinDOS/Airport

### Общие соглашения

Локации обозначаются словами и номерами (подробнее - см. Визуализатор).

Номера пассажиров, самолетов, багажа (то есть всего, что генерируется) обозначаются гуидами.

Вещественные числа пишутся через точку.

Строки передаем в UTF8.

## Пассажиры

Основной цикл: генерация пассажиров по одному на рейсы. Как только их накопилось достаточно/истекло время - отправляет сообщение “хочу улететь” (и тут понеслась обработочка).

API:

1. “Посадка в автобус”, flightId, busId, n мест в автобусе. Возвращает passengerIds посаженных в автобус пассажиров.
2. “Высадка из автобуса”, busId. Действие возвращает ок.
3. “Посадка в самолет”, passengerId. Действие возвращает ок (наверно лучше массивом)
4. “Вылет”, flightId. Действие возвращает ок.
5. “Получить пассажиров”, flightId. Возвращается passengerIds для заданного рейса.
6. Необходимо добавить генерацию служебного багажа с привязкой по flight\_id, а не к пассажиру

Через API выдает массив пассажиров по flightId

Вылет:

Сгенерировать пассажира в состоянии "Жду конца регистрации на рейс flightId"

Послать сообщение "Сохрани багаж" MQ Bag с рандомным багажом (багаж хранит id владельца) и flightId

Если текущий пассажир - n-й (или если прошло n минут от первого пассажира), то:

Изменить статус всех пассажиров с flightId на "Регистрация окончена"

Послать сообщение "Хочу улететь" в MQ UND с flightId. Изменить статус всех пассажиров этого рейса на "Прохожу таможню, жду автобус, ..."

При получении запроса "Посадка в автобус" (с flightId, busId, на n пассажиров):

Найти n рандомных пассажиров с рейса, которые еще не в автобусе

Изменить их статус на "Сажусь в автобус"

Подождать m секунд

Изменить статус пассажиров на "Еду в автобусе"

Вернуть массив этих пассажиров

При получении запроса "Высадка из автобуса" (с busId):

Изменить статус на "Ожидаю посадки в самолет"

Послать сообщение "Хочу сесть в самолет" в MQ Aircraft с flightId и ids пассажиров из этого автобуса

Вернуть ОК

При получении запроса "Посадка в самолет" (passengerId):

Подождать m секунд

Изменить статус пассажира на "Сел в самолет"

Вернуть ОК

При получении запроса "Вылет" (flightId):

Изменить статус всех пассажиров рейса на "Улетел"

Вернуть ОК

### API

Методы API:

* /passengers
  + GET - возвращает список пассажиров, параметры:
    - ids - IDs пассажиров через запятую, информацию о которых нужно вернуть
    - flight - ID рейса, пассажиров которого нужно вернуть
    - status - фильтрует пассажиров, возвращая только тех из запрошенных, которые находятся в данном состоянии
    - extended - Подробная информация о пассажирах/пассажире
  + POST - добавляет пассажиров, параметры
    - count - количество добавляемых пассажиров, от 1 до 25, 1 если не указано
    - place - место/состояние, в котором будут сгенерированы пассажиры ('Unknown', 'WaitForBus', 'InBus', 'WaitForAirplane', 'InAirplane')
    - flight - идентификатор рейса, с которого они летят, либо на который пришли
    - status - фильтрация по статусу, выдаёт список только тех пассажиров, кто имеет указанный статус/указанное состояние (например InBus, WaitForBus, InAirplane)
  + DELETE - удаляет пассажиров
    - id - ID пассажира, которого нужно удалить (если не указано - удаляет всех)
    - confirm - подтверждение удаления (если удаляются все)
* /landing/<transport> (transport = bus или airplane)
  + POST/GET - высадка пассажиров, параметры:
    - id - ID пассажира, которых нужно высадить из <transport>
    - ids - IDs пассажиров через запятую, которых нужно высадить из <transport>
    - transportID - ID транспорта, из которого осуществляется высадка пассажиров
* /boarding/<transport> (transport = bus или airplane)
  + POST/GET - посадка пассажиров, параметры:
    - id - ID пассажира, которых нужно посадить в <transport>
    - ids - IDs пассажиров через запятую, которых нужно посадить в <transport>
    - flight - рейс, пассажиров которого нужно посадить
    - transportID - ID транспорта, в который сажаем пассажиров
    - seats - количество мест в транспорте
* /generate\_flight
  + POST/GET - генерация пассажиров и багажа на рейс, параметры:
    - flightID - ID рейса, пассажиров и багаж на который генерируем
    - pas - количество пассажиров (от 0 до 100, default = 20)
    - lug - количество багажа у пассажиров (от 0 до pas, default = random)
    - serlug - количество служебного багажа (от 0 до 50, default = random)
    - arriving - флаг, говорящий о том, что пассажиры прилетают на самолёте
    - transportID - обязательное значение, если указан флаг arriving. требуется указать ID самолёта, на котором прилетают пассажиры
    - extended - возвращает расширенную информацию о сгенерированных пассажирах на рейс
* /luggage\_notify
  + POST - доставка багажа
    - luggage - список гуидов багажа которые доставлены для выдачи пассажирам
* /change\_status
  + POST - изменение статуса пассажиров
    - newStatus - новый статус, который необходимо установить указанным пассажирам
    - transportID - ID самолёта, если статус = FlewAway/InAirplane, ID автобуса, если статус = InBus **(GUID, а не “Type|GUID”!)**
    - passengers - список id пассажиров, которым необходимо изменить статус

Возможные состояния пассажиров:

InAirplane - в самолёте

LandingFromAirplaneToBus - пересадка из самолёта в автобус

InBus - сидят в автобусе, едут к аэропорту

LandingFromBusToGate - высадка из автобуса в терминал

WaitForLuggage - ожидают свой багаж в терминале

Left - ушли

WaitForBus - ожидают автобус на посадку

BoardingToBusFromGate - посадка в автобус из терминала

InBus - сидят в автобусе, едут к самолёту

BoardingFromBusToAirplane - пересадка из автобуса в самолёт

InAirplane - в самолёте

FlewAway - улетели

Все пассажиры со статусом Left/FlewAway удаляются при перезагрузке сервера

## УНД

Предполагаю, что УНД знает все дороги, а машины не знают

**API**

exchange: **amqp.gtc**

type: **direct**

queue: **gtc.gate**

отправлять сообщения с помощью **basic\_publish** функции

в свойствах сообщения (у меня это msg.properties, на других ЯП аналогично должно быть) необходимо указать

1. correlation\_id = uuid вашей машины - эта информация понадобится вам, когда я пришлю разрешение, чтобы вы поняли, для какой именно машины пришел ответ
2. reply\_to = “queuename” - ваша очередь, куда вы будете ждать ответ, советую сделать всего одну и не париться

Формат сообщения, которые ждет УНД:

Сообщение типа **movement (обязательно заполняйте reply\_to и correlation\_id - это guid вашего объекта)**

Запрос на передвижение

{

“service”: “bus|follow\_me|fuel|baggage”,

“request”: “movement”,

“from”: “current pos”,

“to”: “destination”,

“status”: “need|done” //need - когда вы запрашиваете, done - когда доехали, куда я вам сказал

}

Ответ на передвижние

{

“service”: “bus|follow\_me|fuel|baggage”,

“request”: “movement”,

“from”: “current pos”,

“to”: “destination”, //я изменяю это поле на то, куда вам нужно в данный момент

}

*То есть, схема сообщений на передвижение выглядит так:*

***movement need*** *(от вас)*

***movement*** *(от меня с указанием, куда нужно проехать в данный момент)*

***movement done*** *(от вас, когда доехали до места, указанного мною в предыдущем сообщении)*

Сообщение типа **maintain**

запрос (уведомление унд), что обслуживание закончено

{

“service”: “bus|follow\_me|fuel|baggage”,

“request”: “maintain”,

“airplane\_id”: “id”

} // на него я не отвечаю (заполнять reply\_to не надо)

Сообщение типа **accept** (только для follow\_me)

{

“service”: “follow\_me”,

“request”: “accept”,

“airplane\_id”: “id”,

“parking\_id”: “id”,

} //без ответа, просто уведомление, что появился новый самолет (reply\_to не надо)

Сообщение типа **service**

**{**

“service”: “bus|follow\_me|fuel|baggage”,

“request”: “service”,

“action”: “unload|load” //это поля только для автобуса и багажа, у остальных его не будет

“gate\_id”: “id” //куда ехать, чтобы разгрузить/загрузить (оно всегда одинаковое)

“airplane\_id”: “id”,

“parking\_id”: “id”,

**}** // не жду ответа (reply\_to не надо)

## Автобус

MQ:

1. “Перевези людей”, flightId, gateId, parkingId.

Сообщения *service* присылать на очередь “Bus”

Для *movement* каждая машина будет в reply\_to указывать свою очередь

Сообщения “service” получает BusCenter(MQ Bus), все остальные сообщения игнорируются (но логируются).

После получения сообщения BusCenter обращается к API пассажиров (возможно у меня устаревшая информация, но это можно будет поправить) и получает по flightId количество пассажиров со статусом “WaitForBus”.

Далее происходит поиск свободной машины:

При запускании нового процесса отдельный автобус кидает в Bus сообщение приветствия, мол, “запиши меня в список”.

Количество мест в автобусах одинаковое (по-хорошему, надо 100, но пока 50)

Если Bus|xxx-xxx-xxx - “Free”, то сообщение УНД переадресовывается его MQ

Иначе сообщение пересылается в конец очереди Bus

Таким образом, если в одну машину не вместилось, то сообщение продублируется и будет ждать следующую свободную машину

Bus|xxx-xxx-xxx :

Сообщение в BusCenter о смене состояния: оно просто есть ¯\\_(ツ)\_/¯

Поездка из одного конца в другой:

При получении сообщения “service” отправляет “movement need” УНД, ждет ответ “movement”. При получении “movement” двигается в заданную точку, пересылая сообщения о нахождении визуализатору. По окончании движения посылает “movement done”.

Самолет (еще не точно):

Посадка:

Обращаюсь к API пассажиров, чтобы указать, какие пассажиры высаживаются (из автобуса). Отправляю в MQ самолета запрос на забор пассажиров. Жду ответ. Очищаю массив пассажиров.

Высадка:

Отправляю в MQ самолета запрос на выдачу пассажиров. Жду ответ. Получаю массив пассажиров. Обращаюсь к API пассажиров, чтобы указать, какие пассажиры сели.

API пассажиров:

Посадка: (не знаю)

Высадка: (не знаю)

Процесс работы автобуса:

“action” : “unload”

Сообщение в BusCenter (“Busy”) -> Поездка из гаража к самолету -> Самолет Посадка -> Поездка от самолета к PassengerGate -> API Высадка -> Сообщение в BusCenter(“Free”) -> (ожидание “service”) -> Поездка от PassengerGate в гараж

“action” : “load”

Сообщение в BusCenter (“Busy”) -> Поездка из гаража к PassengerGate -> API Посадка -> Поездка от PassengerGate к самолету -> Самолет Высадка -> Сообщение в BusCenter(“Free”) -> (ожидание “service”) -> Поездка от самолета в гараж

## Багаж

Сообщения *service* присылать на очередь “Baggage”

MQ:

1. “Сохрани багаж (пока подразумевается пассажир в аэропорте)”, passengerId (владелец), flightId (рейс владельца)
2. "Перевези багаж", flightId, gateId, parkingId.

API:

1. Получить количество багажа на рейс, fligtId.
2. Добавить (Зарегистрировать) багаж по flight\_id
3. Извлечь N багажа по flight\_id

Методы **API**:

Антон: А URL то какой?

Дима: Пока никакой, не заливал ещё никуда

Антон: ((((((((((((((((((((((((((((((((

Дима: DIMA\_API = ‘http://localhost.ru’

Дима: content = request.get(f”{DIMA\_API}/api/baggage/{flight\_id}”)

Дима: Можешь сделать так

Антон: А что такое f ?

Дима: ну это типо как request.get(”{}/api/baggage/{}”.format(DIMA\_API, flight\_id))

Антон: Ебать, а так можно было штоле

Дима: агась, сам офигел когда узнал)

Дима: но до этого юзал и format и %s ))

Антон: Мой мир перевернулся (ред.)

Никита присоединился к чату.

Никита: Дима, не шали

Дима был кикнут, так как шалил

Антон: (

Антон грустит.

Обозначения:

1. Обязательный - (\*)

- /api/baggage/”flight\_id”

* GET - возвращает список, состоящий из id багажа
  + параметры:
    - flight\_id - id рейса (\*, тип uuid, в URL)
    - length (вернуть количество багажа на рейс flight\_id)

- /api/baggage

- POST - добавить багаж по flight\_id

- параметры:

- flight\_id - id рейса (\*)

- baggage\_ids - список id багажа (\*)

- registration - параметр, который указывает багажной компоненте, что это багаж регистрируется на рейс

- пример ответа : {“status”:”ok”}

- DELETE - извлечь/удалить багаж, возвращает список ids

- параметры:

- flight\_id (\*)

- count - кол-во багажа (\*)

- /api/baggage/delete

- DELETE - удалить багаж по baggage\_id

- параметры:

- baggage\_ids - список id багажа

MQ:

queue: **baggage**

Вид сообщения:

{

“action”: “load” or “unload”, (загрузи багаж на самолёт / разгрузи),

“flight\_id”: “”,

“parking\_id”: “”,

“gate\_id”: “”

}

отправлять сообщения с помощью **basic\_publish** функции

Через API выдает количество багажа на рейс по flightId.

При получении сообщения "Сохрани багаж" в MQ Bus с объектом багажа и flightId:

Сохранить багаж в словарь по ключу flightId

При получении сообщения "Перевези багаж" в MQ Bus с flightId, fromId, toId:

Найти свободные машины для перевода всего багажа

Для каждой машины параллельно:

Послать запрос "Перевези" в УНД с bagCarIconId, bagParkingId, fromId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Подождать m1 секунд

Послать запрос "Перевези" в УНД с bagCarIconId, fromId, toId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Послать запрос "Получи багаж" в Самолет с flightId, n(сколько багажа в машине). Дождаться ответа ОК

Послать сообщение "Багаж ОК" в MQ UND с flightId.

Послать запрос "Перевези" в УНДс busIconId, toId, busParkingId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

## Заправка

MQ:

Очередь основная: **refuelerMQ -** здесь жду service

Очередь для ответов: **refuelerAnswerMQ**

Ответ от Самолета:

{

“request”:”answer”

“aircraftid”: “id”

“fuelerid”: ”id”

“status”: “ok”

}

1. “Заправь”,aircraftId..

При получении сообщения "Заправь" в MQ Bus с aircraftId:

Получить parkingId через API Самолет по aircraftId

Получить вместимость по топливу n через API Самолет по aircraftId

Найти свободную машину

Подождить m секунд

Послать запрос "Перевези" в УНД с fuelIconId, fuelParkingId, parkingId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Послать запрос "Заправься" в Самолет с aircraftId. Дождаться ответа ОК

Послать сообщение "Заправка ОК" в MQ UND с aircraftId

Послать запрос "Перевези" в УНД с fuelIconId, parkingId, fuelParkingId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

## Follow-me

MQ:

Очередь основная: **FMMQ**

Запрос на посадку

Вид сообщения

{

“service”: “follow\_me”

“request” : “landing”

“aircraftId”: id

}

Сообщение об окончании посадки

{

“service”: “follow\_me”

“request” : “landingcomp”

“fmid”: id / будет передаваться в correlation\_id

} correlation\_id = aircraftId

В очередь FMMQ

1. "Посади", aircraftId.

*При получении сообщения "Посади" в MQ Follow-me с* aircraftId*:*

Ожидаем пока освободятся ВПП, Машина и Парковка

Посылаю в очередь Самолет “Посадка”, aircraftId*,* vppId parkingId

ОК Посылает запрос УНД “Перевезти”, fmparkingId, vppId . От гаража до ВПП

ОК. Визуализируем

"Прицепись к Follow-me", aircraftId, fmId

ОК Посылает запрос УНД “Перевезти”, vppId parkingId. От ВПП до Парковки

ОК. Визуализируем

Послать запрос "Отцепись от Follow-me" в Самолет с aircraftId.

Посылаю в очередь УНД “Посадка”, aircraftId*,* vppId parkingId

ОК Посылает запрос УНД “Перевезти”, parkingId, fmparkingId. От Парковки до гаража

Послать сообщение "Follow-me на посадку ОК" в MQ UND

1. "Отвези на ВПП", aircraftId.

*При получении сообщения "Отвези на ВПП" в MQ Follow-me с* aircraftId

Найти свободную машину fmId

Послать запрос "Перевези" в УНД с fmParkingId, parkingId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Послать запрос "Прицепись к Follow-me" в Самолет с aircraftId, fmId. Дождаться ответа ОК

Послать запрос "Перевези" в УНД с parkingId, vppId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Послать запрос "Отцепись от Follow-me" в Самолет с aircraftId. Дождаться ответа ОК

Послать запрос "Перевези" в УНД с vppId, fmParkingId. Дождаться ответа ОК. Отрисовать перевозку. Отослать в УНД “Перевозка ОК”.

Послать сообщение "Follow-me на вылет ОК" в MQ UND

## Самолет

Примечания:

Тип Guid означает строку, представляющую UUID v4

Обозначение Type[] означает массив объектов типа Type. Выглядит так:

[

{...}, // объект типа Type

{...} // объект типа Type

]

### Http server - http://airplane.eu-4.evennode.com/

* GET /info - возвращает информацию о всех имеющихся самолетах в виде массива объектов следующего вида

{

id: Guid,

landingFlightId: Guid,

departureFlightId: Guid,

passengersCount: int,

baggageCount: int,

fuel: double,

maxFuel: double,

maxPassengersCount: int,

maxBaggageCount: int,

status: string,

additionalInfo: {

// здесь ситуативная информация о состоянии самолета (айди заправщика, место на парковке, айди автобуса или машины багажа и т.д.)

},

}

Также имеющиеся самолеты можно отфильтровать (по факту получается, что после фильтрации будет либо 1 самолет в массиве. либо пустой массив). Условия фильтрации задаются с помощью query string. Знак вопрос после имени поля означает, что это поле опционально

{

id?: Guid,

landingFlightId?: Guid,

departureFlightId?: Guid,

parkingId?: string

}

Например, запрос может выглядеть так: /info?landingFlightId=3c072970-61e5-47d7-8c4a-534510b12507

В последующих 5 url’ах плейсхолдер *:airplane* означает айди самолета. Например, url может выглядеть так: */3c072970-61e5-47d7-8c4a-534510b12507/followMe/startToParking.* В заголовках запроса должен быть указан *content-type: application/json.*

* POST /:airplane/followMe/startToParking - начать следовать за FollowMe машиной на парковку. Body выглядит так: *{ carId: string }*
* POST /:airplane/followMe/EndToParking - закончить следовать за FollowMe машиной на парковку. Body выглядит так: *{ parkingId: string }*
* POST /:airplane/followMe/startToStrip - начать следовать за FollowMe машиной на ВПП. Body выглядит так: *{ carId: string }*
* POST /:airplane/followMe/endToStrip - закончить следовать за FollowMe машиной на ВПП. Body выглядит так: *{ stripId: string }*
* DELETE /:airplane - удалить самолет

Во всех вышеперечисленных случаях в случае успеха возвращается пустой ответ с кодом 200. Если самолет не найден - 404. Если нарушена логика самолета (например, незаправленный самолет пытаются повезти на ВПП) - 400.

### MQ

Очередь называется Airplane. При отправке ответных сообщений учитывается reply\_to и correlation\_id из props’ов. Сейчас очереди заправщика и FollowMe забиты в код, указывать их в props’ах не обязательно, но можно. Формат ожидаемых сообщений описан ниже

* Создать самолет в небе.

{

type: "CreateLandingAirplane";

value: {

landingFlight: {

id: Guid,

code: string,

passengersCount: int,

serviceBaggageCount: int

},

departureFlight: такая же структура, как и у landingFlight;

}

}

* Разгрузить багаж

{

type: "UnloadBaggage";

value: {

airplaneId: Guid,

carId: string,

count: int

}

}

* Загрузить багаж

{

type: "LoadBaggage";

value: {

airplaneId: Guid,

carId: string,

baggages: Guid[]

}

}

* Посадка

{

type: "Landing";

value:{

stripId: string,

aircraftId: Guid,

}

}

* Взлет

{

type: "Fly";

value:{

aircraftId: Guid

}

}

* Заправка

{

type: "Refuel";

value:{

volume: double,

aircraftId: Guid

}

}

* Загрузка пассажиров

{

type: "LoadPassengers";

value:{

airplaneId: Guid,

busId: string,

passengers: Guid[],

}

}

* Выгрузка пассажиров

{

type: "UnloadPassengers";

value: {

airplaneId: Guid,

busId: string,

count: int

}

}

## Расписание

Генерирует рейсы (с пассажирами/без) и посылает запросы к модулям пассажиров и самолета на генерацию.

MQ

1. Потребляет очередь ‘schedule’ и принимает на вход строку в виде json:

{

"input\_passenger\_count": N,

"input\_service\_baggage\_count": S,

"output\_passenger\_count": M,

"output\_service\_baggage\_count": P

},

где N - кол-во пассажиров, которые находятся в новом самолёте,

S - кол-во служебного багажа на борту,

M - кол-во пассажиров, которые будут на следующем рейсе при нового самолёта,

P - кол-во служебного багажа в следующем рейсе

## Визуализатор

Сообщения посылаются прямиком в очередь visualizer. Если кому будет нужен exchange - пишите, выслушаю ваши предложения :)

### Отрисовка перемещений

Отрисовывает по сообщениям с заданным прогрессом.

Формат сообщения перевозки:

* Type: movement
* From - откуда перевезти
* To - куда перевезти
* Transport - что перевезти
* Duration - поле для плавного перемещения картинки; в нем вы указываете, сколько машинка должна ехать от точки до точки, и тогда машинка плавно переместится в новую точку за указанный период времени. В миллисекундах.
* Aircraft - guid самолета, если машинка followMe везет его до стоянки/полосы.

Транспорт описывается типом и id в виде строки формата “TransportType|Id”

Id - номер транспорта в строковом виде, это может быть число (1, 2,3), guid или что-то еще на ваше усмотрение (доступные айди согласуются при подключении компоненты к визуализатору - они отправляют сообщение). В коде визуализатора всё интерпретируется как строка.

TransportType - одно из следующих значений:

* Aircraft - самолет
* Bus - автобус
* Fuel - машина с топливом
* FollowMe - машина “follow-me”
* Baggage - машина с багажом

Пример: если нужно передвинуть автобус с пассажирами от аэропорта до самолета на парковке 2 за 10 секунд, то посылается сообщение:

Type: “movement”

From: “PassengerGate|1”

To: “AirParking|2”

Transport: “Bus|3”

Duration: 10000

### Инициализация транспорта

Прежде, чем начать работать с транспортом, нужно прислать список ids всех используемых машин.

*Для тех, кто хочет знать больше: каждой машинке потенциально могут соответствовать разные картинки; для того, чтобы у одной и той же машины все время была одна и та же картинка, необходимы айдишники.*

Формат сообщения:

Type: “init”

TransportType: “type”

Ids: [“id1”, “id2”, … ]

Add: true/false - необязательно поле (Дима, special for you. Ну, спасибо))

где type - это один из перечисленных выше TransportType (Aircraft, Bus и т.п.); id1, id2 и т.д. - строковые айдишники транспорта; Add - добавить машинки из init сообщения к тем, что уже есть (true) или заменить на новые (false), по умолчанию происходит замена (false).

### Отрисовка анимации

Предполагаемые виды анимации:

* пассажиры садятся в автобус
* пассажиры выходят из автобуса
* пассажиры садятся в самолет
* пассажиры выходят из самолета
* заправка самолета
* и т.п.

Сообщения для анимации:

1. Взлет самолета

Type: “animation”

AnimationType: “wheelsup”

Duration: - длительность анимации взлета, в миллисекундах.

Strip: - полоса (от 1 до 3).

Transport: “Aircraft|guid”

2. Посадка

Type: “animation”

AnimationType: “touchdown”

Duration: - длительность анимации посадки, в миллисекундах.

Strip: - полоса (от 1 до 3).

Transport: “Aircraft|guid”

Для всех остальных анимаций формат такой:

Type: “animation”

AnimationType: “(строка из списка ниже)”

Duration: - длительность анимации, в миллисекундах.

Transport: “TransportType|Id”

Виды AnimationType (они обобщенные, потому что я вряд ли успею сделать их особенными для каждого вида):

* baggage - загрузка/выгрузка багажа - (транспорт - багажная машинка)
* passengers - посадка/высадка пассажиров (транспорт - автобус)
* filling - заправка (транспорт - заправочная машина)

# Вопросы к Н.С. Чапкину

1. Как организовать общение с ответом? web, tcp, udp
2. Что очередями, а что с ответом? +
3. Что с ошибками (не все пассажиры сели, например) +
4. Что с надежностью (компонента упала) +
5. Что с рассинхроном визуализатора (можно сделать запросы к нему блокирующими) +
6. Пассажиры должны создаваться/садиться в самолет по одному или массивом? +

По кусочкам передвигать машинку, когда доехала - сказать, я приехала.

Подумать схему очистки, очистка очередей.

Самолет должен появляться с рейсом, на который он потом полетит. Переделать логику пассажиров, чтобы их забирали, а не они сами просились

# Обсудить

1. Общение УНД и самолета: как начинается и как заканчивается.
2. Откуда взялся модуль расписание (подозреваю, что это дело рук Александра).