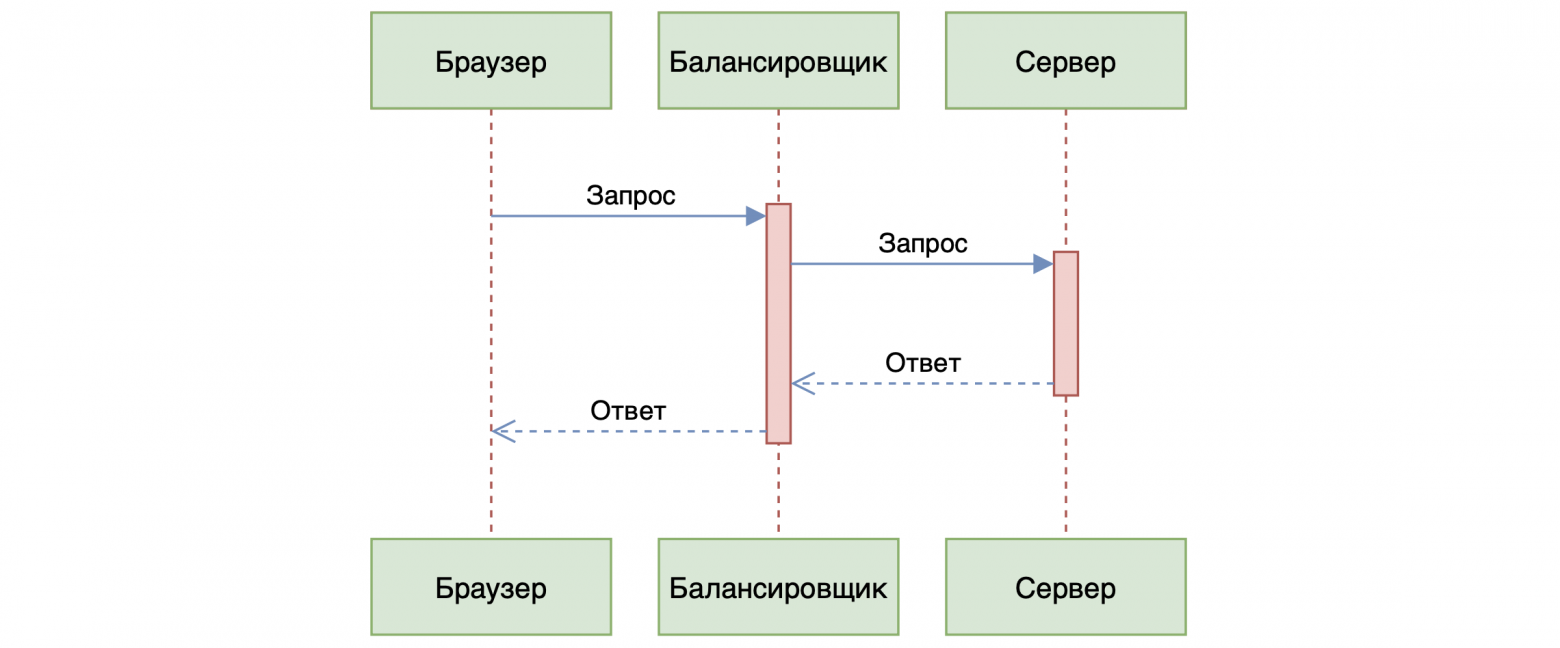
Что это?

Диаграмма последовательности используется для визуализации взаимодействия между объектами в системе. Она показывает порядок сообщений, которыми объекты обмениваются друг с другом во времени и их жизненный цикл. На мой взгляд, это самый простой и удобный инструмент для демонстрации всех интеграций и взаимодействий в рамках проектируемого бизнес-процесса.

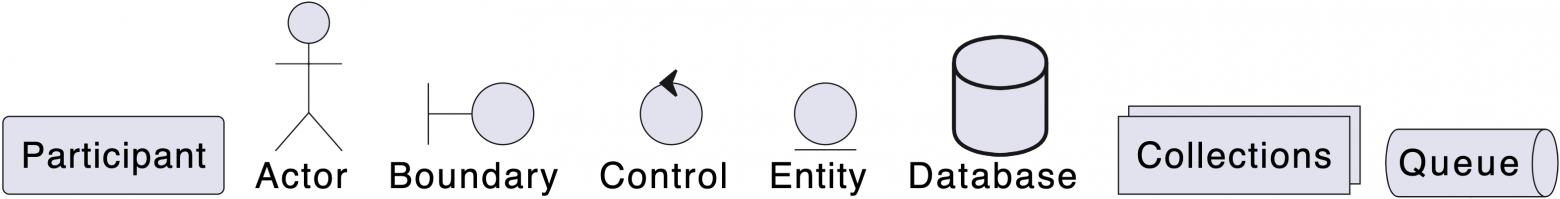
Из чего состоит?

Основные элементы, из которых состоит sequence‑диаграмма: объекты (зеленое), линии жизни (красное), сообщения (синее).



Пример sequence-диаграммы

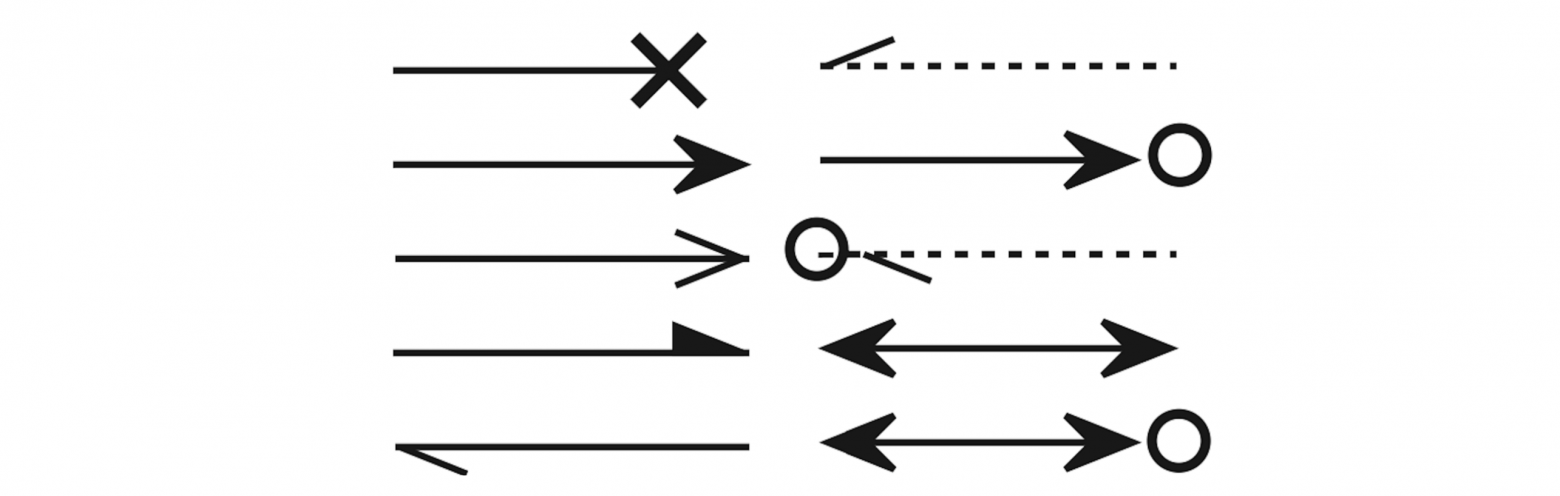
**Объекты** — это сущности, которые взаимодействуют друг с другом. Например, пользователь, топик или очередь, микросервис. Помимо названия, объекты могут еще отличаться своим типом:



Типы объектов

**Линии жизни**(lifelines) отображают течение времени, а также активность или исполнение определенных функций объекта. Представляют из себя вертикальную линию с блоком активности.

**Сообщения**(messages) показывают обмен информацией между объектами. Например, клик пользователя на кнопку, но чаще это запрос или ответ, вызов хранимой процедуры или отправка сообщения. Сообщения отображаются в виде стрелок, направленных от одной линии жизни к другой, и бывают нескольких типов:



Типы стрелок

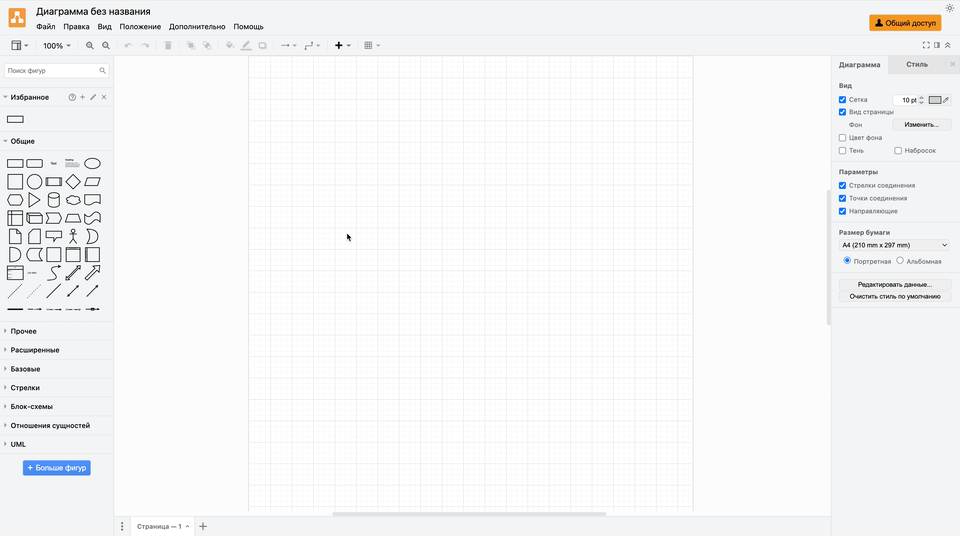
Где и как пользоваться?

Есть много вариантов, где можно рисовать UML-диаграммы. Самые популярные:

* Графический редактор [draw.io](https://app.diagrams.net/)
* Песочница от [plantUML](https://www.plantuml.com/plantuml/uml)
* С помощью плагина в инструментах JetBrains или VSCode

Графический редактор draw.io

Самое простое решение, которое не требует изучение синтаксиса UML — графический редактор draw.io, он же diagrams.net. Для создания диаграммы с помощью draw.io нужно только немного разобраться в интерфейсе программы, найти в поиске фигуры для создания sequence диаграмм и начать перетаскивать их на холст:



Создание диаграммы в draw.io

Все довольно просто, но проблемы начинаются, когда необходимо сделать диаграмму с большим количеством взаимодействий объектов. Будет много редактирований и нарисовать такую диаграмму займет гораздо больше времени. Поэтому лучше все-таки разобраться с синтаксисом UML и освоить 2 других варианта **«**рисования**»** диаграмм.

Разберемся с синтаксисом

К сожалению, просто взять и начать использовать UML диаграмму не получится. Нужно хотя бы поверхностно изучить синтаксис PlantUML. Он очень простой, поэтому базовых примеров будет достаточно, чтобы начать прорабатывать диаграммы.

Сначала нужно еще раз осознать, что есть вышеописанные сущности: объекты, линии жизни и сообщения. Чтобы появились линии жизни необходимо четко определить в начале документа объекты:

actor "Пользователь" as u  
participant "Фронт" as f  
participant "Бэк" as b

Или можно этого не делать (чего я не рекомендую, поскольку читаемость, на мой взгляд, сильно ухудшится), так как они автоматически появятся при описании второй сущности — сообщений. Для описания сообщений используются стрелки «->». Нужно только указывать из какого объекта идет сообщение и в какой — «client -> server»:

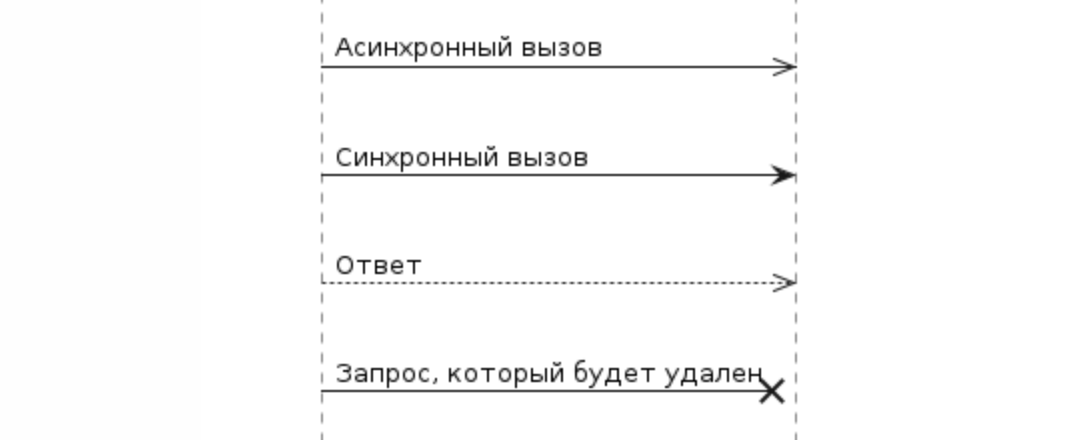
actor "Пользователь" as u  
participant "Фронт" as f  
participant "Бэк" as b  
  
u -> f  
f -> b  
b -->> f  
f -->> u

На основе своего опыта сразу подсвечу, обычно никто не пользуется сразу всеми типами объектов и стрелок. Среди типов объектов самые распространенные:

* **actor** — пользователь или «клиент»;
* **participant** — сервер, компонент системы;
* **database**— база данных;
* **queue** — топик или очередь.

Среди часто используемых стрелок:

a ->> b: Асинхронный вызов  
a -> b: Синхронный вызов  
a -->> b: Ответ  
a -x b: Запрос, который будет удален  
'или от которого решили отказаться/устаревший



Часто используемые стрелки

Синхронный/Асинхронный вызов это –

Группировка

Теперь, когда мы разобрались с самыми распространенными типами стрелок и объектов, предлагаю разобраться с еще одним важным понятием — группировка сообщений. Здесь я опять же разберу самые популярные и те, которые в моей практике реально встречаются. Группировка нужна для того, чтобы как‑то осмысленно скомпоновать сообщения, а также для того, чтобы наложить определенные правила для отдельно взятых сообщений. Например, можно объединить несколько сообщений по смыслу и как‑то их обозвать, а можно объединить их в группу **»**par» тем самым подсветив, что сообщения в этой группе выполняются параллельно.

**Alt** — группировка, которая накладывает определенные правила. Используется для того, чтобы показать альтернативные кейсы для взаимодействия, используется с оператором **else**. Например, объект может отправить запрос, а второй объект, в зависимости от полученных данных, выполнит разные действия:

Синтаксис и пример

**Opt** — используется для того, чтобы выделить сообщение в опциональный кейс. Например, объект может отправить запрос, а второй объект, в зависимости от полученных данных, может выполнить дополнительное действие:

Синтаксис и пример

**Par** — используется для того, чтобы выделить сообщения, которые выполняются параллельно. Используется с оператором **else**. Например, объект может отправить запрос, а второй объект параллельно выполнит внутренние вычисления и вернет ответ первому объекту:

Синтаксис и пример

**Loop** — используется для того, чтобы выделить сообщения, которые выполняются несколько раз. Например, объект может отправить запрос, а второй объект выполнит внутренние действия несколько раз в зависимости от отправленных данных:

Синтаксис и пример

**Group** — используется для того, чтобы скомпоновать сообщения по смыслу. Например, на диаграмме может быть много взаимодействий и для того, чтобы было меньше путаницы можно подписать группы сообщений:

Синтаксис и пример

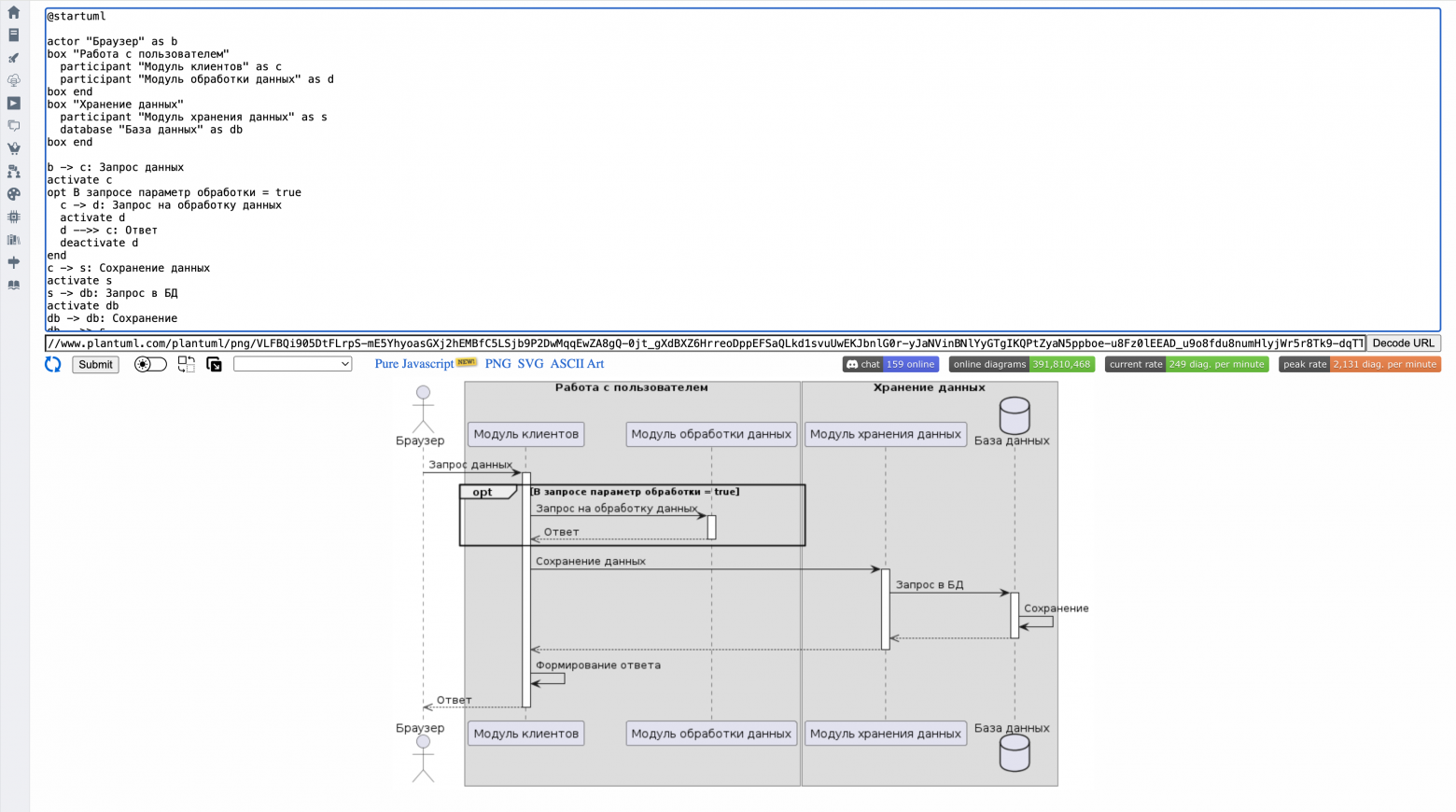
Еще сгруппировать можно сами объекты. Это может быть полезно для описания взаимодействия нескольких систем с микросервисами:

Синтаксис и пример

Подробное описание всех возможностей sequence-диаграмм можно прочитать [здесь](https://plantuml.com/sequence-diagram), включая все типы объектов, стрелок и группировок, которые я решил не описывать в этой статье.

Песочница от plantUML и плагины

После того, как мы разобрались с синтаксисом UML уже можно «рисовать» более сложные диаграммы, например, в [песочнице](https://www.plantuml.com/plantuml/uml) от plantUML. Выглядит это так:



Пример работы в plantUML

Исходный код

Но самый удобный инструмент для работы с sequence‑диаграммами — ваша любимая ide. Например, вы можете установить плагин прямо в WebStorm от JetBrains.

Подробнее:

Создавать диаграммы через любимые ide самый удобный способ, вдобавок файлы хранятся непосредственно на вашем устройстве. Формат UML для описания диаграмм последовательности очень хорошо работает с системами версионирования, поэтому можно смело вести свои проекты с использованиям git'а.

Интересные возможности

Цвета

Изменение цвета стрелки

Такая возможность пригодится в тех случаях, когда необходимо что-то подсветить.

*Я сталкивался с тем, что в отдельных компаниях был конкретный регламент на цвета стрелок. Например, необходимо было существующие потоки красить****в синий****цвет, существующие, которые дорабатываются —****в оранжевый****, а новые —****в красный****.*

Еще помимо стрелок можно красить и сам текст. Как это использовать — додумаете сами.

Изменение цвета текста

Также можно настраивать цвета для «линии жизни» и самого объекта. Кстати, в Интернете можно найти огромное количество настроек для кастомизации базовых стилей, включая шрифты, толщину линий, отступы и многое другое. Такие настройки, как правило, начинаются с оператора «skinparam».

Изменение стилей

Блок активности

Недавно открыл для себя, что необязательно каждый раз писать «activate» и «deactivate», досаточно «++» и «--».

Пример:

Autonumber

Многие знают, что это за оператор и как он работает, но далеко не многие знают, что его функционал куда шире, чем просто инкрементирование номера сообщения.

Номер можно еще стилизовать, переопределять, добавлять в него подпункты, использовать как переменную и многое другое (подробнее [здесь](https://plantuml.com/ru/sequence-diagram#%D0%9D%D1%83%D0%BC%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B9%20%D0%B2%20%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8F%D1%85)).

Изменение autonumber

Подсказка

Очень редко на своей практике встречал диаграммы, на которых кто-то пользовался этой крутой фишкой. С помощью нее можно подробнее описать какие-то нюансы взаимодействия или просто указать адрес (host:port).

Подсказка

Разделительная черта

Не смотря на всю крутость подсказок, их я использую редко, а вот разделительную черту я использую довольно часто. Бывают случаи когда интеграция имеет, например, 2 сценария. Чтобы не захламлять страницу с аналитикой и не дублировать кучу uml-кода, можно нарисовать одну диаграмму, но разбить ее на 2.

Разделительная черта

Заключение

В заключение хочу сказать, что диаграмма последовательности очень крутой и мощный инструмент для описания взаимодействий и интеграций, а научиться им пользоваться довольно легко, особенно когда есть под рукой материал с описанием синтаксиса.