

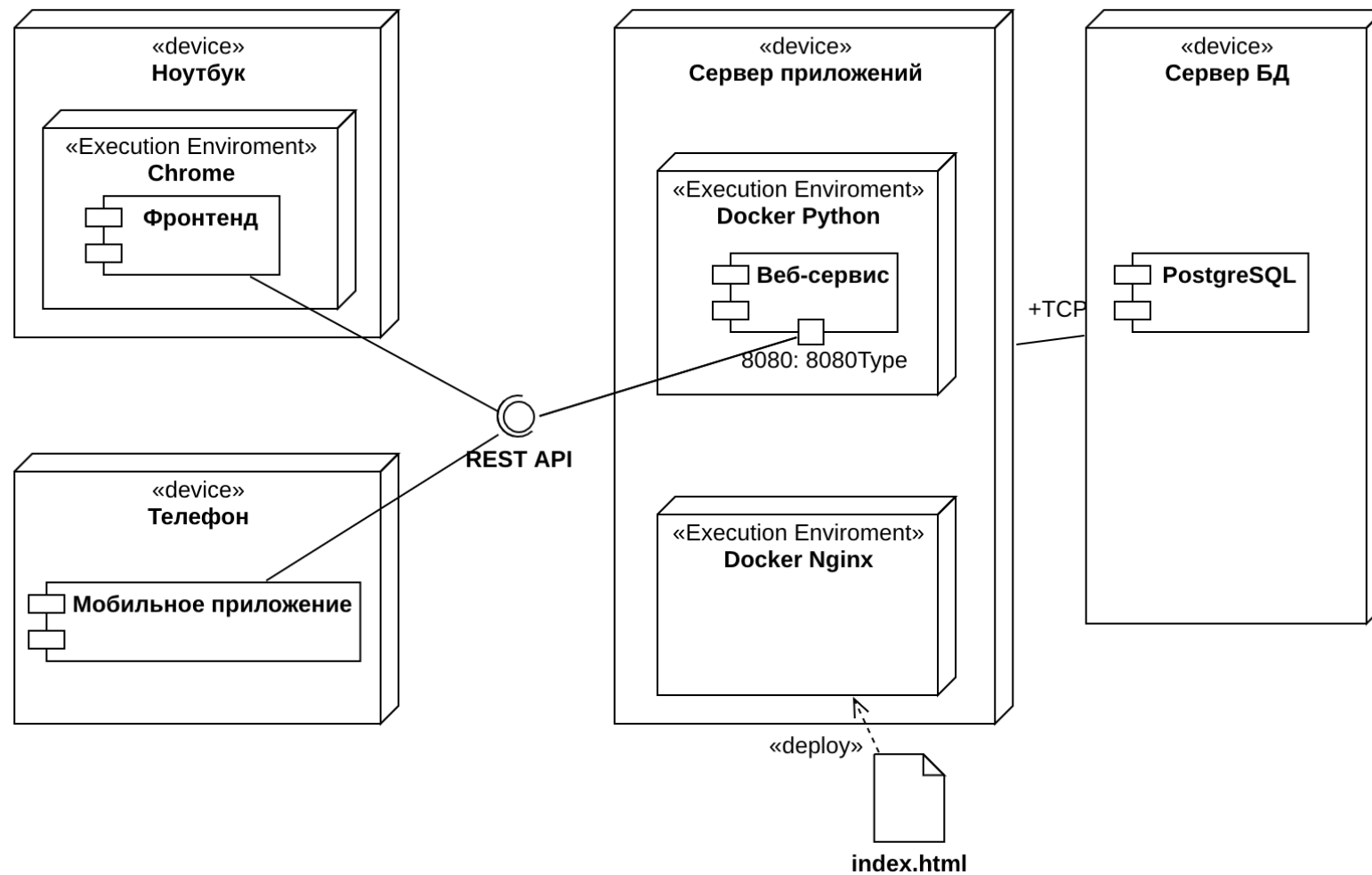
Лекция 5

Веб-сервисы

Разработка интернет приложений

Канев Антон Игоревич

Трехзвенная архитектура. API



Веб-служба

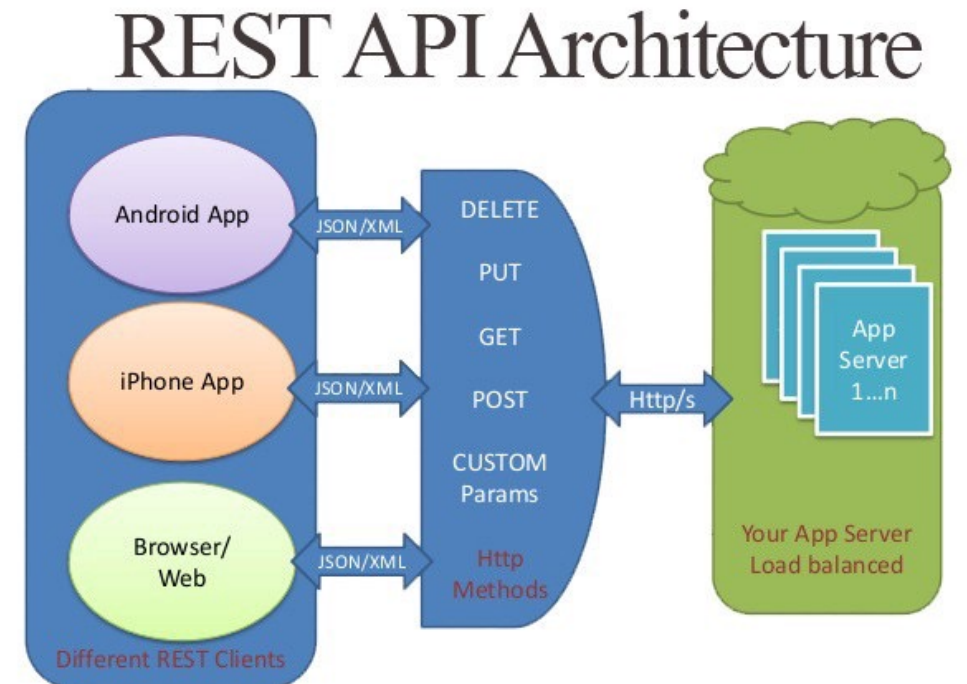
- Веб-служба, веб-сервис (web service) — идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.
- Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC и т. д.) и соглашениях (REST). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

API

- API (Application Programming Interface) — описание способов взаимодействия одной компьютерной программы с другими.
- Обычно входит в описание какого-либо интернет-протокола (например, SCIM), программного каркаса (фреймворка) или стандарта вызовов функций операционной системы.
- Часто реализуется отдельной программной библиотекой или сервисом операционной системы.
- Проще говоря, это набор компонентов, с помощью которых компьютерная программа (бот или же сайт) может использовать другую программу.

REST

- REST - Representational State Transfer. Набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать
- RESTful – веб-службы не нарушающие ограничений
- Ограничения: клиент-сервер, отсутствие состояния, кэширование, единообразие интерфейса, слои, код по требованию



RPC

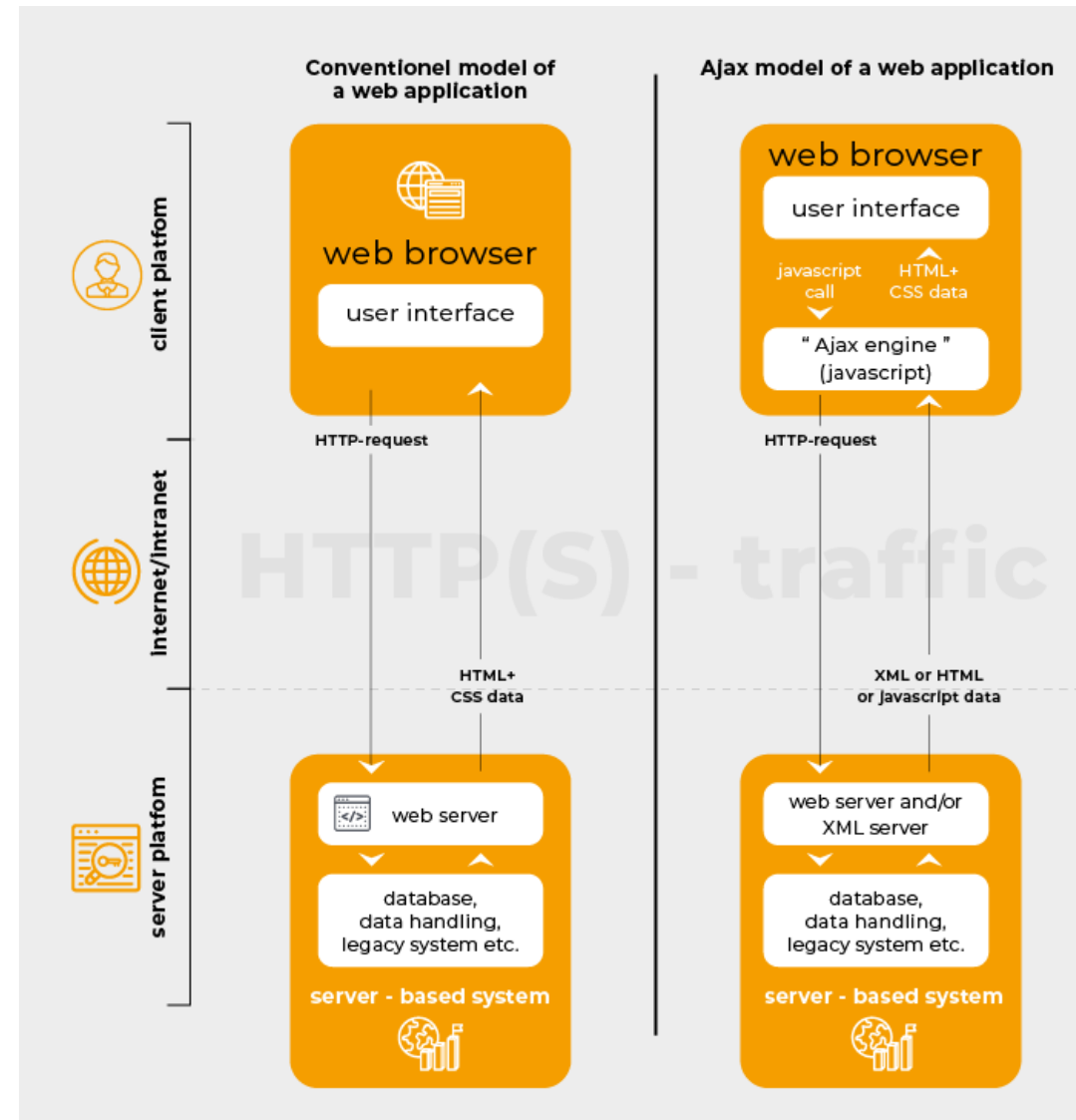
- RPC - remote procedure call. Удалённый вызов процедур
- Реализация RPC-технологии включает два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур для необъектных RPC)
- JSON-RPC
- XML RPC
- gRPC
- SOAP

gRPC

- **gRPC** (Remote Procedure Calls) — это система удалённого вызова процедур (RPC) с открытым исходным кодом, первоначально разработанная в Google
- В качестве транспорта используется HTTP/2, в качестве языка описания интерфейса — Protocol Buffers.
- gRPC предоставляет такие функции как аутентификация, двусторонняя потоковая передача и управление потоком, блокирующие или неблокирующие привязки, а также отмена и тайм-ауты.

AJAX

- **AJAX**, Ajax (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.
- В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.
- **JSON-RPC** (JavaScript Object Notation Remote Procedure Call — JSON-вызов удалённых процедур) — протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.



XMLHttpRequest

- **XMLHttpRequest** (XMLHTTP, XHR) — API, доступный в скриптовых языках браузеров, таких как JavaScript.
- Использует запросы HTTP или HTTPS напрямую к веб-серверу и загружает данные ответа сервера напрямую в вызывающий скрипт.

- Информация может передаваться в любом текстовом формате, например, в XML, HTML или JSON. Позволяет осуществлять HTTP-запросы к серверу без перезагрузки страницы.

```
var http_request = new XMLHttpRequest();
http_request.onreadystatechange = function () {
    if (http_request.readyState !== 4)
        return;

    if (http_request.status !== 200)
        throw new Error('request was defeated');

    do_something_with_object(JSON.parse(http_request.responseText));
    http_request = null;
};
http_request.open("GET", url, true);
http_request.send(null);
```

JSON

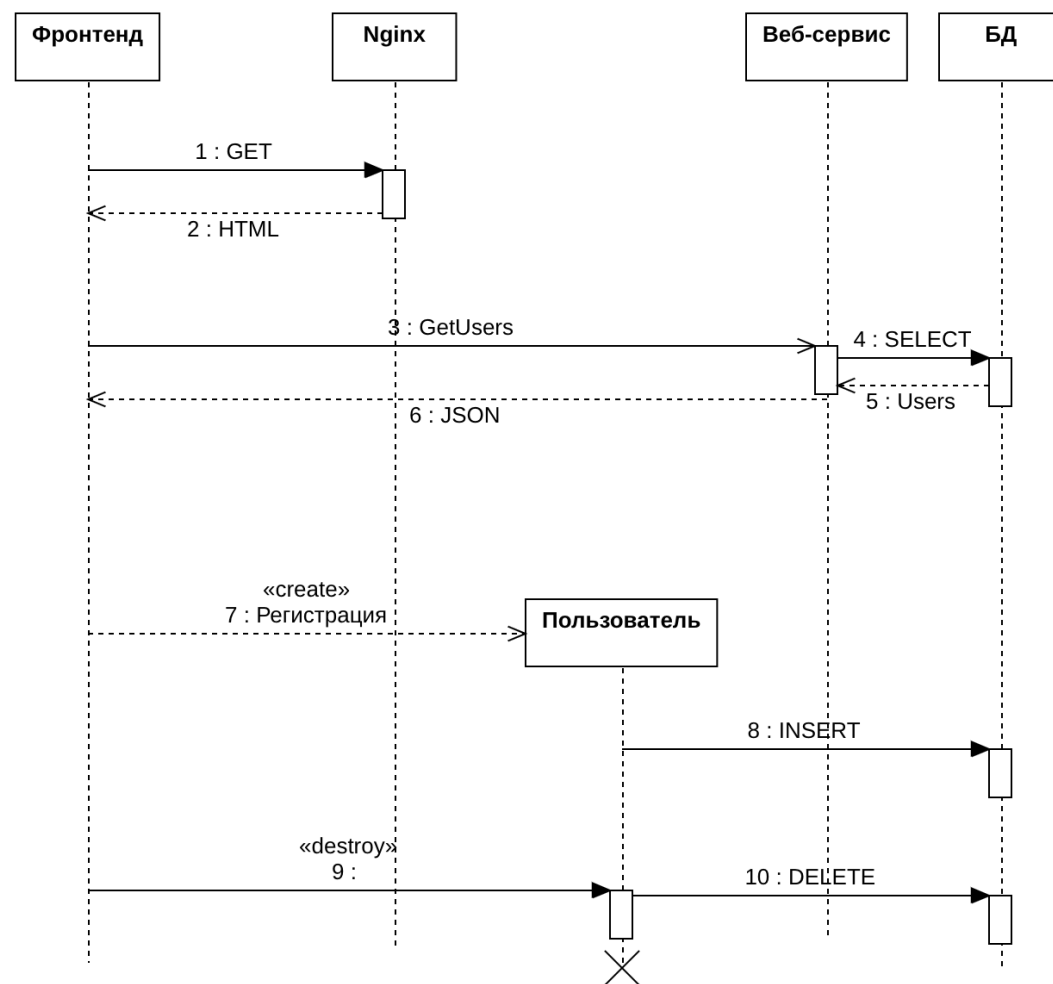
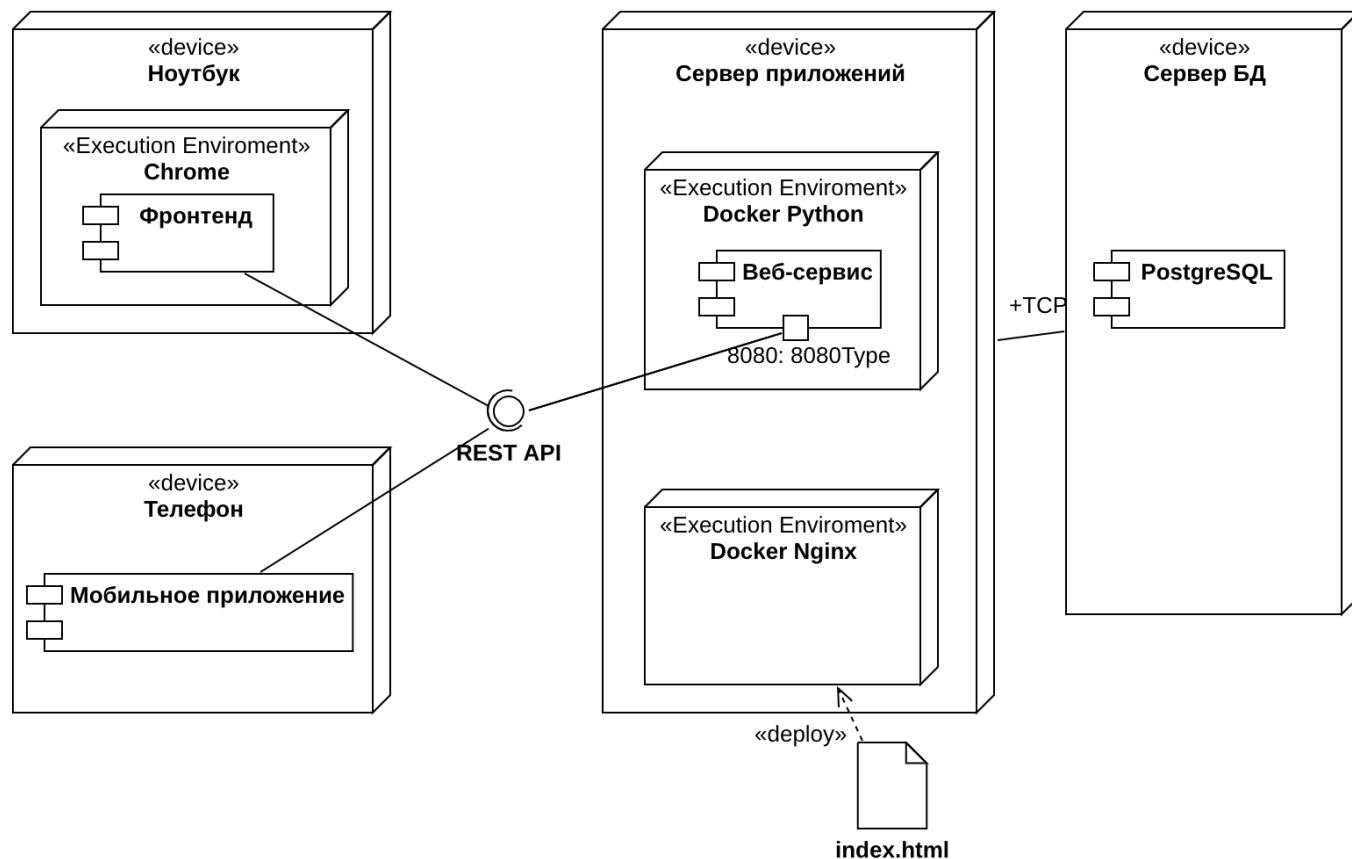
- JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

```
<person firstName="Иван" lastName="Иванов">
  <address streetAddress="Московское ш., 101, кв.101" city="Ленинград" postalCode="101101" />
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

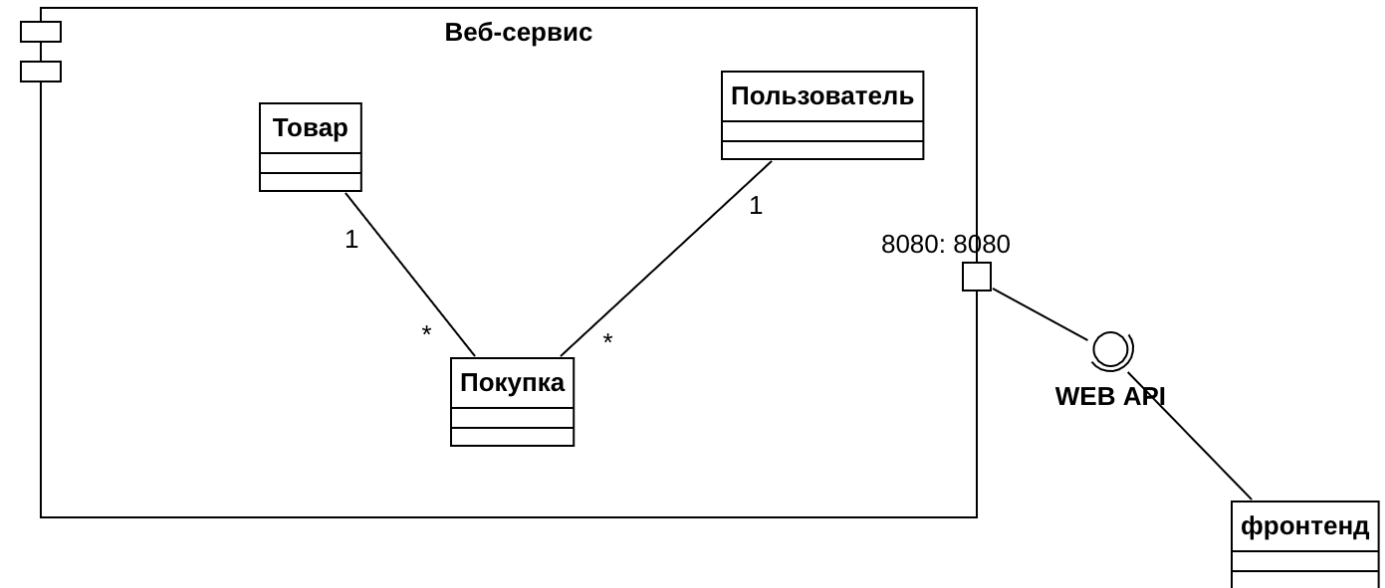
```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Московское ш., 101, кв.101",
    "city": "Ленинград",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

Трехзвенная архитектура. AJAX



Django Rest Framework

- Django REST <https://www.django-rest-framework.org>
- <https://django.fun/ru/docs/django-rest-framework/3.12/>
- Python (3.6, 3.7, 3.8, 3.9, 3.10)
- Django (2.2, 3.0, 3.1, 3.2, 4.0)
- `pip install djangorestframework`



Модель

```
from django.db import models

class Stock(models.Model):
    company_name = models.CharField(max_length=50, verbose_name="Название компании")
    price = models.DecimalField(max_digits=8, decimal_places=2, verbose_name="Цена акции")
    is_growing = models.BooleanField(verbose_name="Растет ли акция в цене?")
    date_modified = models.DateTimeField(auto_now=True, verbose_name="Когда последний раз обновлялось значение")
```

- `python manage.py makemigrations` (создаем файлы миграций)
- `python manage.py migrate` (применяем файлы миграций к базе)

Сериализация

- **Сериализация** — процесс перевода структуры данных в битовую последовательность.
- Обратной к операции сериализации является операция десериализации (структуризации) — создание структуры данных из битовой последовательности.

```
from stocks.models import Stock
from rest_framework import serializers

class StockSerializer(serializers.ModelSerializer):
    class Meta:
        # Модель, которую мы сериализуем
        model = Stock
        # Поля, которые мы сериализуем
        fields = ["pk", "company_name", "price", "is_growing", "date_modified"]
```

ViewSet

```
from rest_framework import viewsets
from stocks.serializers import StockSerializer
from stocks.models import Stock

class StockViewSet(viewsets.ModelViewSet):
    """
    API endpoint, который позволяет просматривать и редактировать акции компаний
    """
    # queryset всех пользователей для фильтрации по дате последнего изменения
    queryset = Stock.objects.all().order_by('date_modified')
    serializer_class = StockSerializer # Сериализатор для модели
```

ViewSet

```
class UserViewSet(viewsets.ViewSet):
    """
    Example empty viewset demonstrating the standard
    actions that will be handled by a router class.

    If you're using format suffixes, make sure to also include
    the `format=None` keyword argument for each action.
    """

    def list(self, request):
        pass

    def create(self, request):
        pass

    def retrieve(self, request, pk=None):
        pass

    def update(self, request, pk=None):
        pass

    def partial_update(self, request, pk=None):
        pass

    def destroy(self, request, pk=None):
        pass
```

```
from django.contrib.auth.models import User
from rest_framework import status, viewsets
from rest_framework.decorators import action
from rest_framework.response import Response
from myapp.serializers import UserSerializer, PasswordSerializer

class UserViewSet(viewsets.ModelViewSet):
    """
    A viewset that provides the standard actions
    """

    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=True, methods=['post'])
    def set_password(self, request, pk=None):
        user = self.get_object()
        serializer = PasswordSerializer(data=request.data)
        if serializer.is_valid():
            user.set_password(serializer.validated_data['password'])
            user.save()
            return Response({'status': 'password set'})
        else:
            return Response(serializer.errors,
                              status=status.HTTP_400_BAD_REQUEST)

    @action(detail=True, methods=['post', 'delete'])
    def unset_password(self, request, pk=None):
        ...
```


URL

```
from django.contrib import admin
from stocks import views as stock_views
from django.urls import include, path
from rest_framework import routers

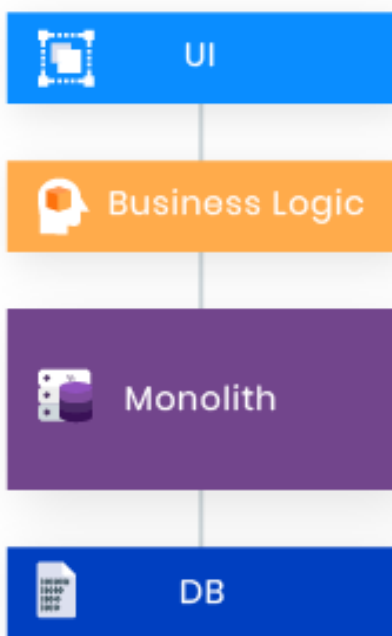
router = routers.DefaultRouter()
router.register(r'stocks', stock_views.StockViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),

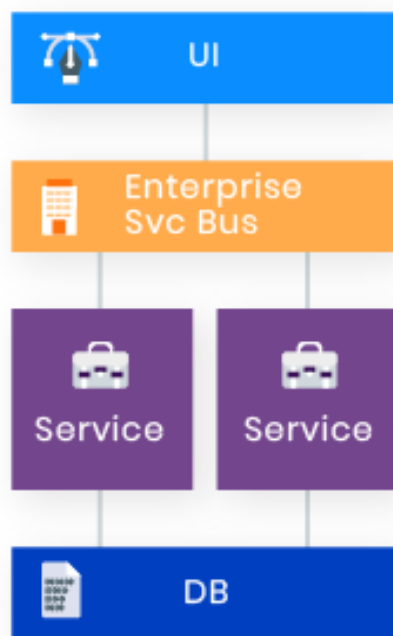
    path('admin/', admin.site.urls),
]
```

Сервис-ориентированная архитектура

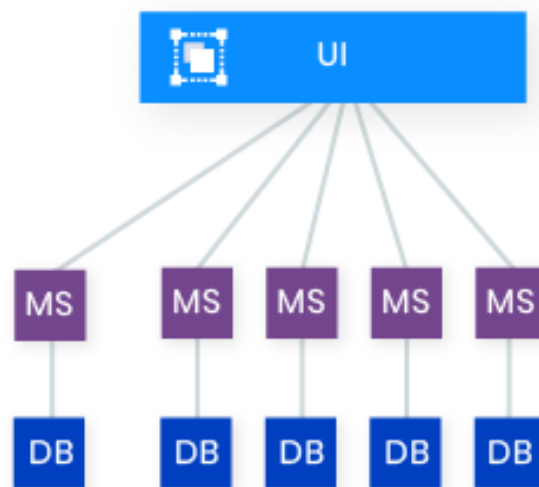
- **Сёрвис-ориенті́рованная архитекту́ра** (COA, англ. service-oriented architecture- SOA) — модульный подход к разработке программного обеспечения, базирующийся на обеспечении удаленного по стандартизированным протоколам использования распределённых, слабо связанных легко заменяемых компонентов (сервисов) со стандартизированными интерфейсами.



Monolithic



Service - Oriented



Microservices

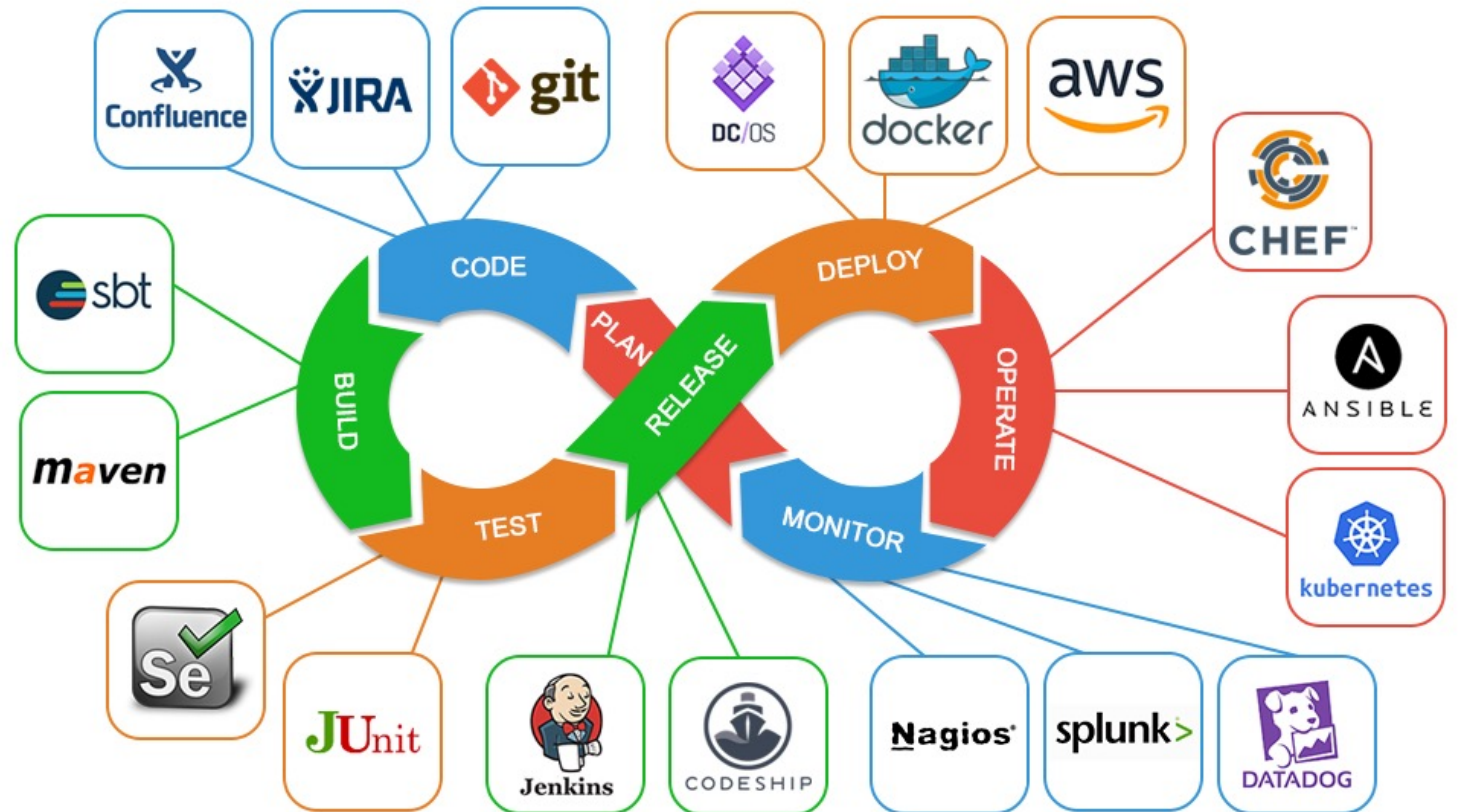
Agile

- Гибкая методология разработки – альтернатива последовательной водопадной
- Разделение процесса разработки на короткие итерации и повторение



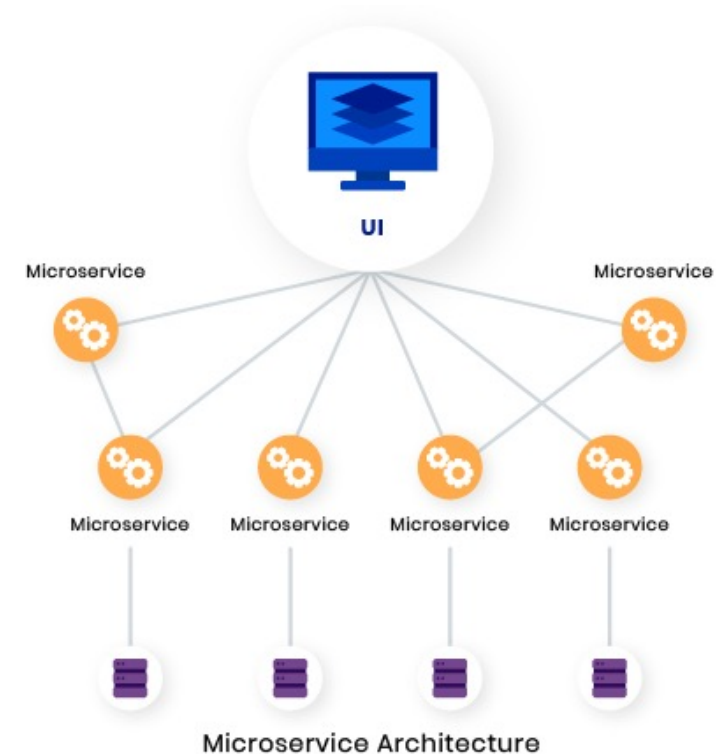
DevOps

- DevOps (development & operations) — методология автоматизации технологических процессов сборки, настройки и развёртывания программного обеспечения
- Быстрый перенос программного обеспечения между разными стадиями жизненного цикла ПО
- Снижение частоты отказов
- Сокращение времени доработок



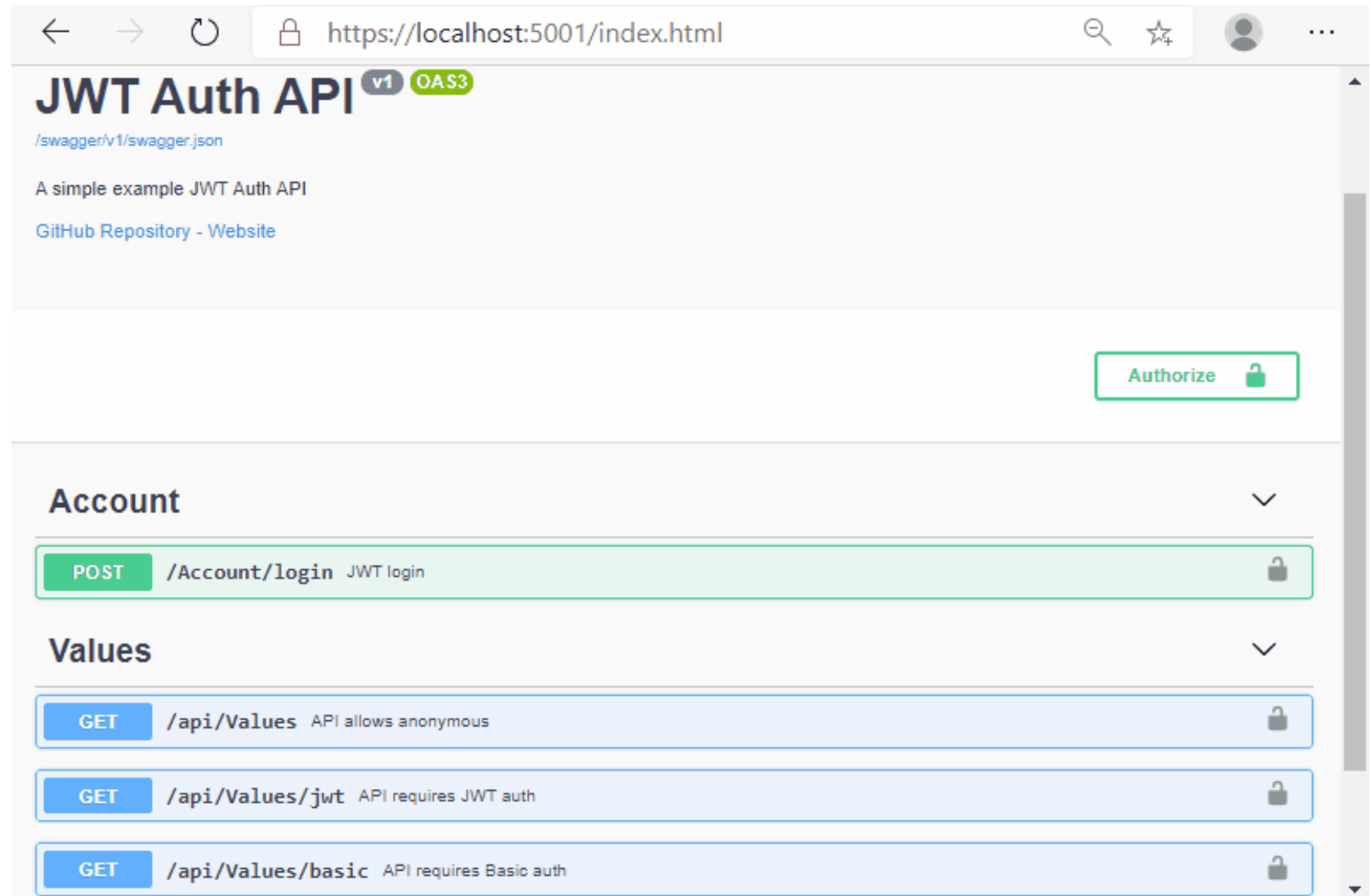
Микросервисная архитектура

- **Микросервисная архитектура** — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps



Инструменты тестирования

- Postman
- Insomnia
- Swagger



Swagger

Developing APIs

- When creating APIs, Swagger tooling may be used to automatically generate an Open API document based on the code itself. This embeds the API description in the source code of a project and is informally called code-first or bottom-up API development.
- Alternatively, using Swagger Codegen, developers can decouple the source code from the Open API document, and generate client and server code directly from the design. This makes it possible to defer the coding aspect.

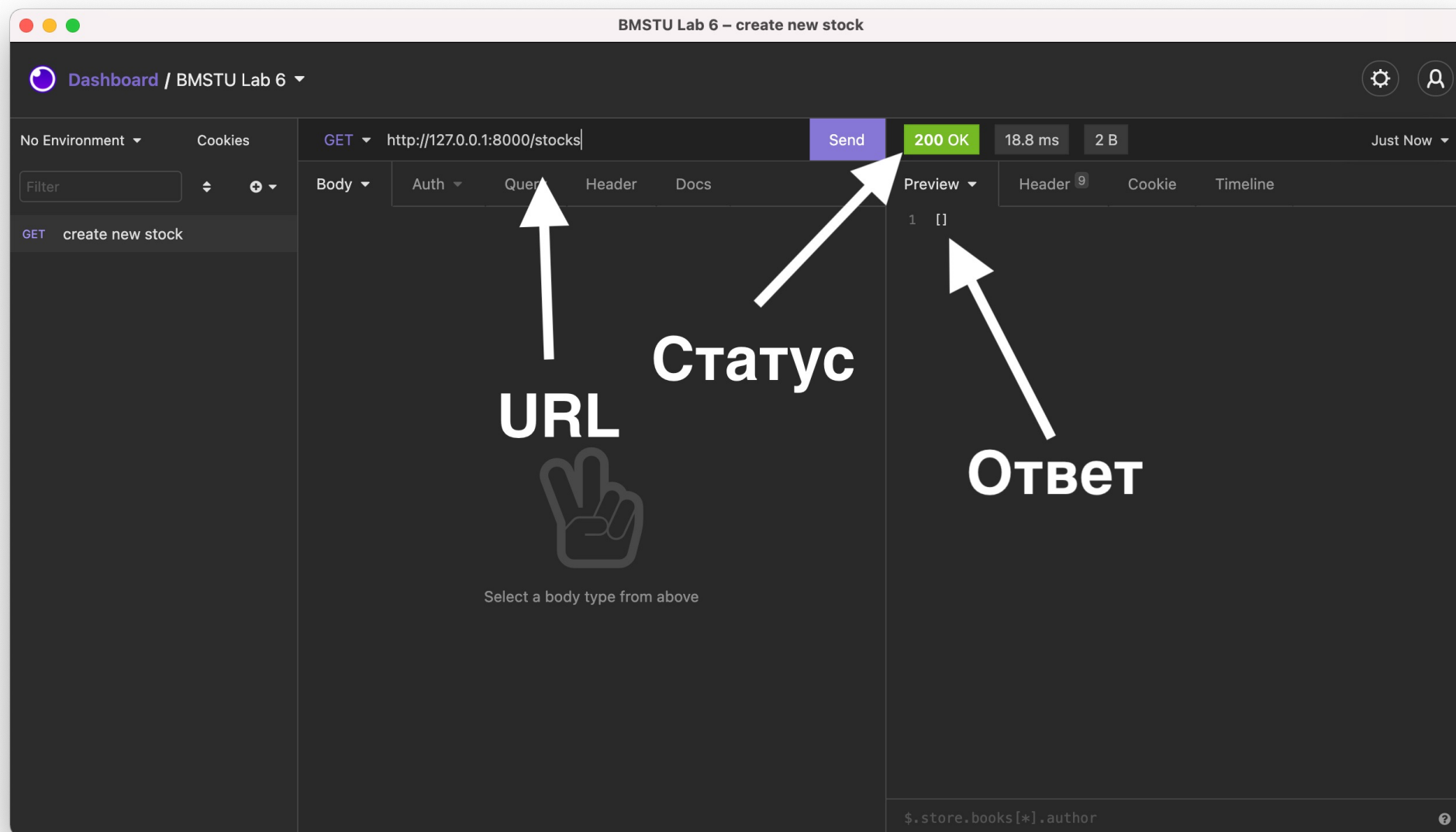
Interacting with APIs

- Using the Swagger Codegen project, end users generate client SDKs directly from the OpenAPI document, reducing the need for human-generated client code. As of August 2017, the Swagger Codegen project supported over 50 different languages and formats for client SDK generation.

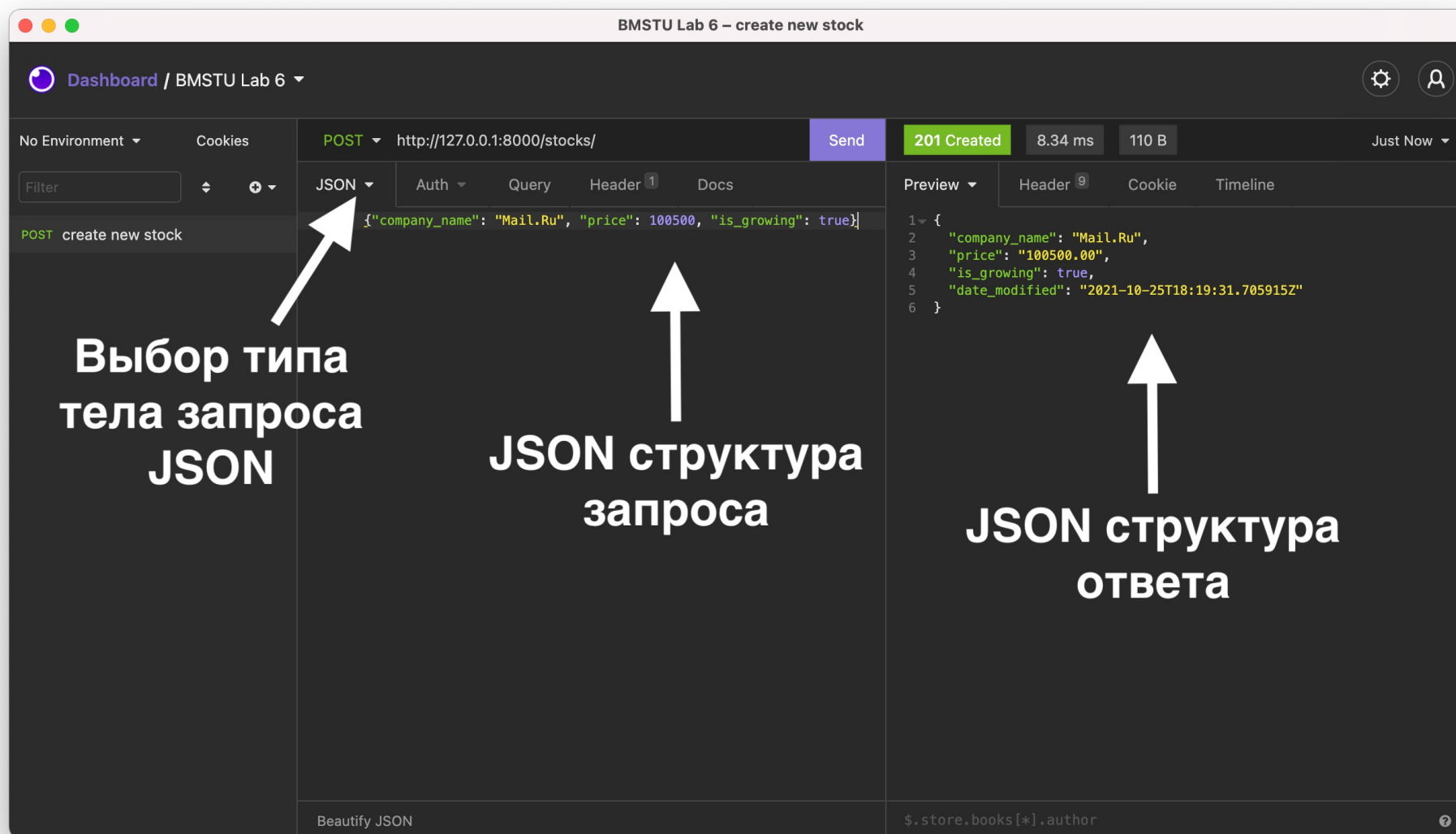
Documenting APIs

- When described by an OpenAPI document, Swagger open-source tooling may be used to interact directly with the API through the Swagger UI. This project allows connections directly to live APIs through an interactive, HTML-based user interface. Requests can be made directly from the UI and the options explored by the user of the interface.

Insomnia



Insomnia



Insomnia

