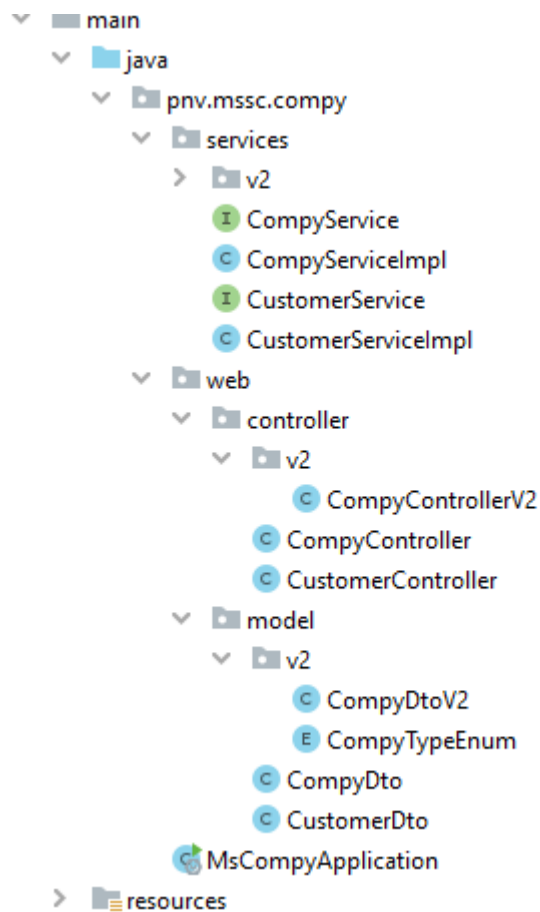


## №5 REST Template

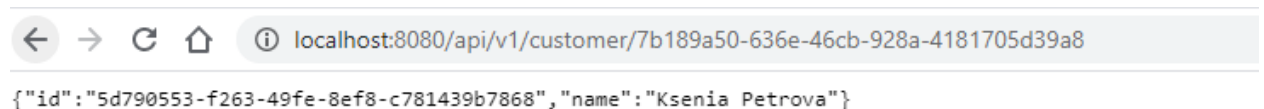
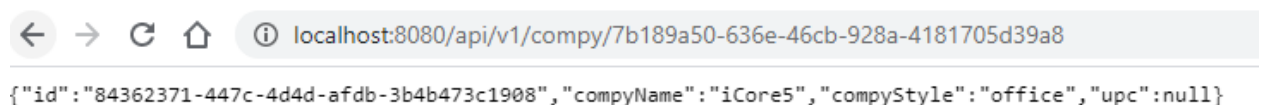
На прошлой лекции мы создали REST ресурс, который возвращает данные и это достаточно просто сделать, но не особо полезно, когда данные получаются через браузер или curl.

Здесь можно посмотреть код:

<https://github.com/PatseiBSTU/mssc-computer>



Проверку мы делали из браузера и через Postman.



## Введение в Spring RestTemplate

Более полезный способ взаимодействия с REST web-сервисом является программный. Для решения этой задачи, Spring предоставляет удобный шаблонный класс **RestTemplate** (<https://docs.spring.io/spring-framework/docs/4.0.x/javadoc-api/org.springframework.web.client.RestTemplate.html>)

Spring RestTemplate - это часть модуля Spring-web, представленного в Spring 3.

- 1) Это блокирующий и синхронный **HttpClient**, который предоставляет простой API-интерфейс шаблонного метода поверх базовых клиентских библиотек HTTP, таких как JDK HttpURLConnection, Apache HttpComponents, и др.
- 2) Spring RestTemplate - это верхний уровень клиента Http, он решает проблему преобразования объектов JSON или XML в объекты Java.
- 3) Клиент Http заботится обо всех низкоуровневых деталях связи через HTTP.
- 4) Он поддерживает веб-службы Restful только с протоколом Http, но не с протоколом Https.
- 5) В будущем он будет устаревшим, поскольку есть WebClient, поддерживающий реактивное программирование, неблокирующий HttpClient.

**RestTemplate** - это класс, который предлагает метод для вызова REST API (веб-API) и является HTTP-клиентом, предлагаемым Spring Framework. Для этого пользователь должен предоставить **URL**, параметры (если есть) и извлечь полученные результаты. По умолчанию он использует JDK HttpURLConnection для связи, но можно переключаться на разные источники HTTP через RestTemplate.setRequestFactory (): Apache HttpComponents, Netty, OkHttp и так далее.

Рассмотрим последовательность его работы:



**delete ()**: удаляет ресурсы по заданному URL-адресу. Использует метод HTTP DELETE.

**put ()**: создает новый ресурс или выполняет обновление для данного URL-адреса с помощью метода HTTP PUT.

**postForObject ()**: создает новый ресурс с использованием метода HTTP POST и возвращает объект.

**postForLocation ()**: создает новый ресурс с помощью метода HTTP POST и возвращает местоположение созданного нового ресурса.

**postForEntity ()**: создает новостной ресурс, используя метод HTTP POST для заданного шаблона URI. Возвращает ResponseEntity.

## Создание RESTful Client

Нужно создать отдельный проект для клиента. Он должен использовать библиотеки **Spring Restful Client**. Можно использовать следующие стартеры:  
spring-boot-starter-web  
spring-boot-starter-data-rest

**spring-boot-starter-web** включает библиотеки для построения веб приложения используя **Spring MVC**, и использовать **tomcat** как **Web Container** по умолчанию. Включает библиотеки для приложения **RESTful**.

В проекте мы так и подключим зависимости

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

Создадим RESTful Client. Настроим внешнее имя хоста через свойства и будем запрашивать compy по id.

В отдельном проекте (можно создавать через Boot инициализатор) создаем класс CompyClient



```
@Component  
@ConfigurationProperties (prefix = "spf.compy", ignoreUnknownFields = false)  
public class CompyClient {
```

```

public final String COMPY_PATH_V1 = "/api/v1/compy/";
private String apihost;

private final RestTemplate restTemplate;

public CompyClient(RestTemplateBuilder restTemplateBuilder) {
    this.restTemplate = restTemplateBuilder.build();
}

public CompyDto getCompyById (UUID uuid){
    return restTemplate.getForObject(apihost+ COMPY_PATH_V1 + uuid.toString(),
CompyDto.class);
}

public void setApihost(String apihost){
    this.apihost = apihost;
}
}

```

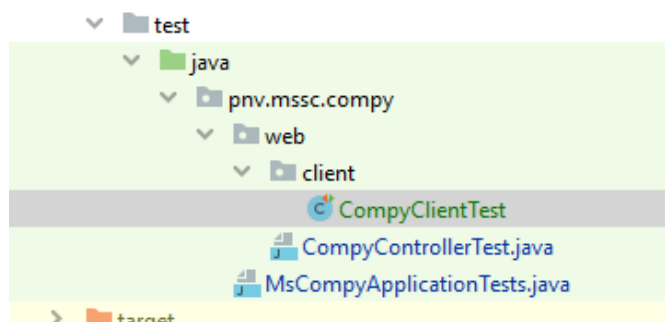
В application.properties определяем

```

spf.compy.apihost = http://localhost:8080

```

Создадим тест для проверки работы REST API на основе созданного клиента



```

@SpringBootTest
class CompyClientTest {

    @Autowired
    CompyClient client;

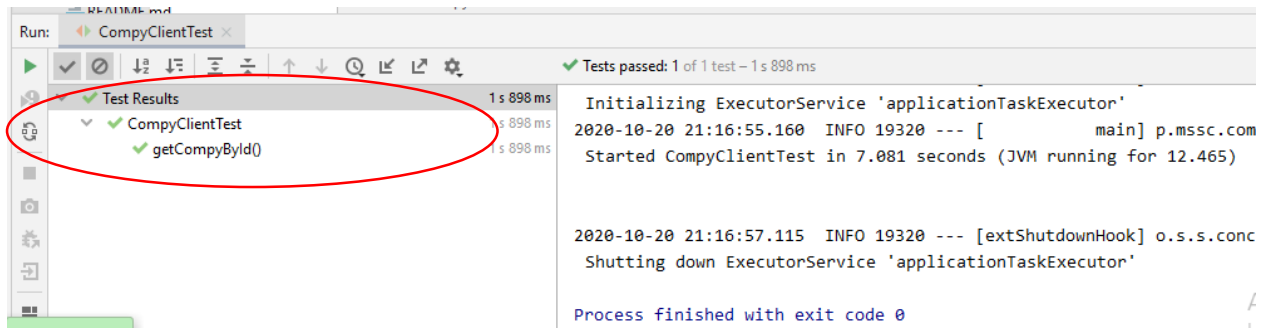
    @Test
    void getCompyById() {
        CompyDto dto = client.getCompyById(UUID.randomUUID());

        assertNotNull(dto);
    }
}

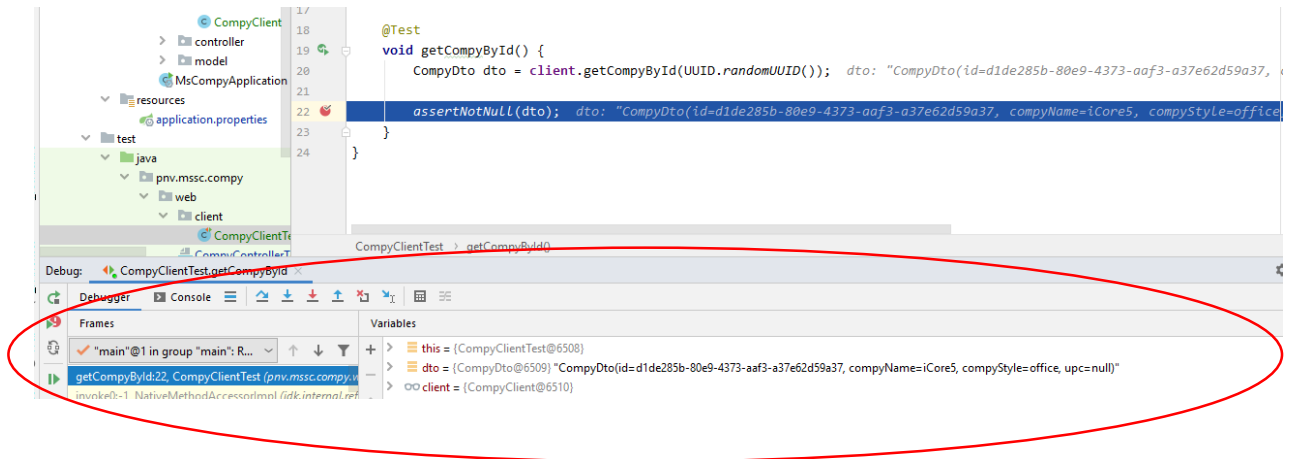
```

Запускаем

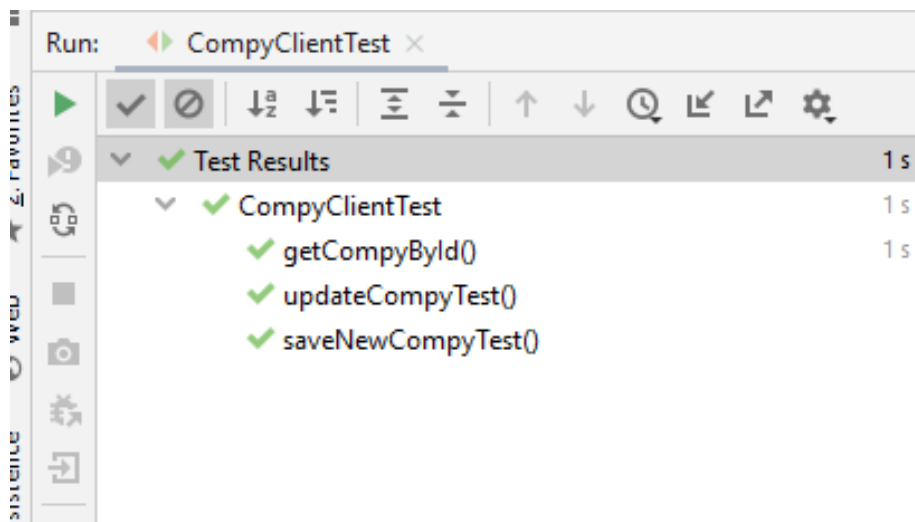
Сначала надо запустить сервер а потом тест



И можно посмотреть в отладчике что возвращается нужный объект



Добавим еще методы в клиента, который будет проверять все запросы (сохранение, удаления, получение и т.д.)



Код клиента можно посмотреть по ссылке

<https://github.com/PatseiBSTU/mssc-computer-client/tree/master>

## HTTP Client

Spring поддерживает несколько клиентских библиотек http через свою абстракцию ClientHttpRequestFactory:

### Блокирующие Clients

- JDK - Java's реализация
- Apache HTTP Client
- Jersey
- OkHttp

### NIO Clients

- Apache Async Client
- Jersey Async HTTP Client
- Netty – используется в Reactive Spring

## Apache HTTP Async Client конфигурация

Мы можем создать дополнительную настройку для всего приложения.

Это немного более сложный подход. Во-первых нужно добавить зависимости

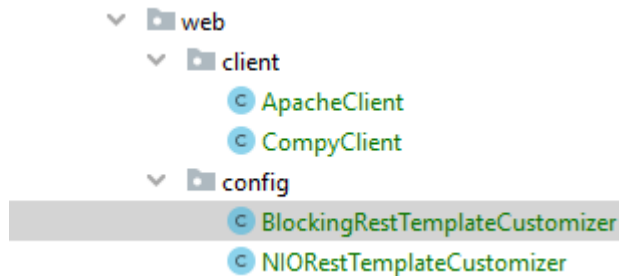
```
<dependency>  
  <groupId>org.apache.httpcomponents</groupId>  
  <artifactId>httpasyncclient</artifactId>  
  <version>4.1.4</version>  
</dependency>
```

Существует три основных подхода к настройке RestTemplate.

Чтобы максимально сузить область любых настроек, введите автоматически настраиваемый RestTemplateBuilder, а затем вызовите его методы по мере необходимости. Каждый вызов метода возвращает новый экземпляр RestTemplateBuilder, поэтому настройки повлияют только на использование строителя.

Чтобы сделать приложение доступным, можно использовать аддитивную настройку bean-компонента RestTemplateCustomizer. Все такие bean-компоненты регистрируются в автоматически настраиваемом RestTemplateBuilder и будут применяться ко всем шаблонам, созданным с его помощью.

Создадим два класса конфигурации RestTemplate и нового клиента ApacheClient



```

@Component
public class BlockingRestTemplateCustomizer implements RestTemplateCustomizer {

    public ClientHttpRequestFactory clientHttpRequestFactory(){
        PoolingHttpClientConnectionManager connectionManager = new
        PoolingHttpClientConnectionManager();
        connectionManager.setMaxTotal(100);
        connectionManager.setDefaultMaxPerRoute(20);

        RequestConfig requestConfig = RequestConfig
            .custom()
            .setConnectionRequestTimeout(3000)
            .setSocketTimeout(3000)
            .build();

        CloseableHttpClient httpClient = HttpClients
            .custom()
            .setConnectionManager(connectionManager)
            .setKeepAliveStrategy(new DefaultConnectionKeepAliveStrategy())
            .setDefaultRequestConfig(requestConfig)
            .build();

        return new HttpComponentsClientHttpRequestFactory(httpClient);
    }

    @Override
    public void customize(RestTemplate restTemplate) {
        restTemplate.setRequestFactory(this.clientHttpRequestFactory());
    }
}

```

```

//@Component
public class NIORestTemplateCustomizer implements RestTemplateCustomizer {

    public ClientHttpRequestFactory clientHttpRequestFactory() throws
    IOException {
        final DefaultConnectingIOReactor ioreactor = new
        DefaultConnectingIOReactor(IOReactorConfig.custom().
            setConnectTimeout(3000).
            setIoThreadCount(4).
            setSoTimeout(3000).
            build());

        final PoolingNHttpClientConnectionManager connectionManager = new
        PoolingNHttpClientConnectionManager(ioreactor);
        connectionManager.setDefaultMaxPerRoute(100);
        connectionManager.setMaxTotal(1000);

        CloseableHttpAsyncClient httpAsyncClient = HttpAsyncClients.custom()
            .setConnectionManager(connectionManager)
            .build();
    }
}

```



```

        return new HttpComponentsAsyncClientHttpRequestFactory(httpAsyncClient);
    }

    @Override
    public void customize(RestTemplate restTemplate) {
        try {
            restTemplate.setRequestFactory(clientHttpRequestFactory());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

@ConfigurationProperties(prefix = "spf.my.compy", ignoreUnknownFields = false)
@Component
public class ApacheClient {

    public final String COMPY_PATH_V1 = "/api/v1/compy/";
    public final String CUSTOMER_PATH_V1 = "/api/v1/customer/";
    private String apihost;

    private final RestTemplate restTemplate;

    public ApacheClient(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public CompyDto getCompyById(UUID uuid){
        return restTemplate.getForObject(apihost + COMPY_PATH_V1 +
uuid.toString(), CompyDto.class);
    }

    public URI saveNewCompy(CompyDto compyDto){
        return restTemplate.postForLocation(apihost + COMPY_PATH_V1, compyDto);
    }

    public void updateCompy(UUID uuid, CompyDto compyDto){
        restTemplate.put(apihost + COMPY_PATH_V1 + uuid, compyDto);
    }

    public void deleteCompy(UUID uuid){
        restTemplate.delete(apihost + COMPY_PATH_V1 + uuid );
    }

    public void setApihost(String apihost) {
        this.apihost = apihost;
    }

    public CustomerDto getCustomerById(UUID customerId) {
        return restTemplate.getForObject(apihost+ CUSTOMER_PATH_V1 +
customerId.toString(), CustomerDto.class);
    }

    public URI saveNewCustomer(CustomerDto customerDto) {
        return restTemplate.postForLocation(apihost + CUSTOMER_PATH_V1,
customerDto);
    }

    public void updateCustomer(UUID customerId, CustomerDto customerDto) {

```

```

        restTemplate.put(apihost + CUSTOMER_PATH_V1 + customerId, customerDto);
    }

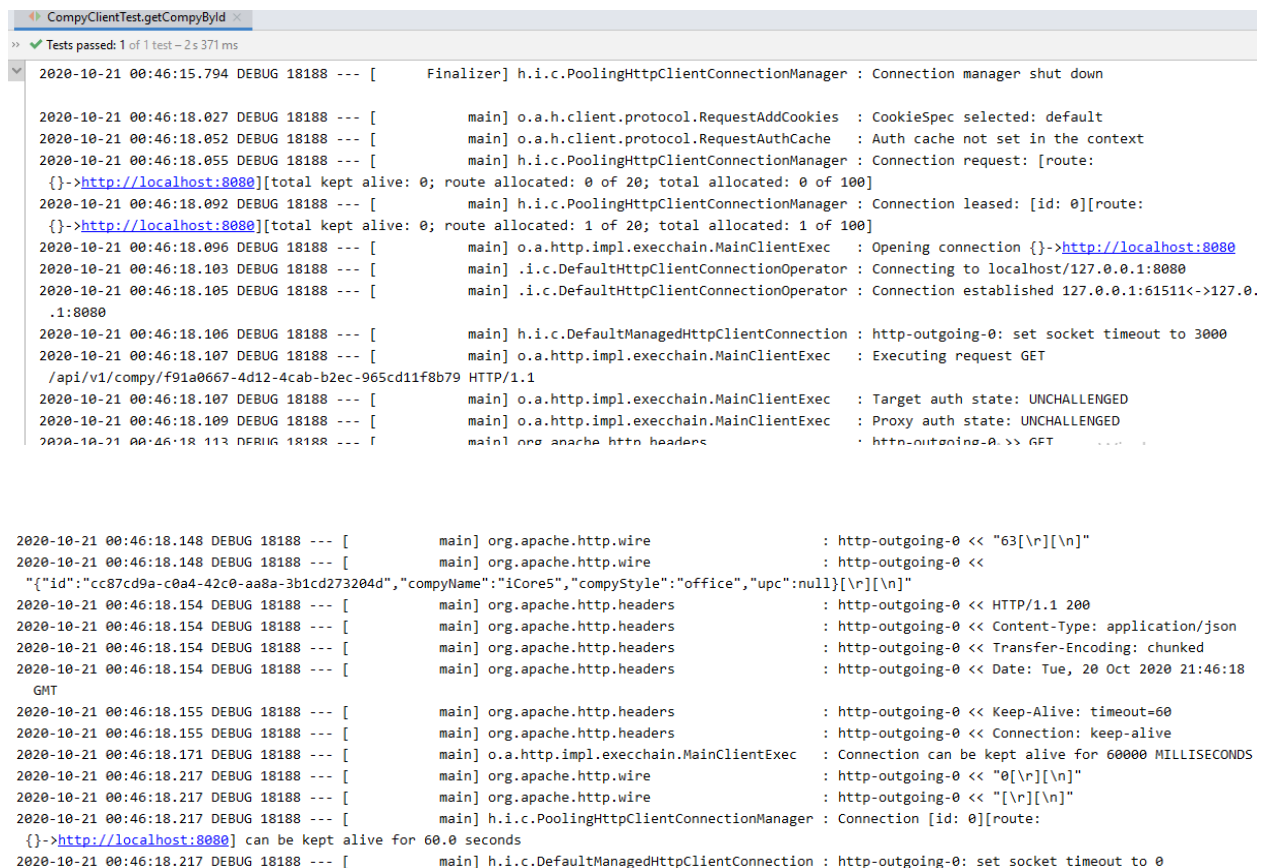
    public void deleteCustomer(UUID customerId) {
        restTemplate.delete(apihost + CUSTOMER_PATH_V1 + customerId);
    }
}

```

## Логгирование Apache Commons HttpClient

В файле свойств включите  
`spf.compy.apihost = http://localhost:8080`  
`logging.level.org.apache.http=debug`

Теперь запустите любой тестовый метод и вы увидите много тестовой полезной информации.



```

2020-10-21 00:46:15.794 DEBUG 18188 --- [        Finalizer] h.i.c.PoolingHttpClientConnectionManager : Connection manager shut down

2020-10-21 00:46:18.027 DEBUG 18188 --- [        main] o.a.h.client.protocol.RequestAddCookies : CookieSpec selected: default
2020-10-21 00:46:18.052 DEBUG 18188 --- [        main] o.a.h.client.protocol.RequestAuthCache : Auth cache not set in the context
2020-10-21 00:46:18.055 DEBUG 18188 --- [        main] h.i.c.PoolingHttpClientConnectionManager : Connection request: [route:
{}->http://localhost:8080][total kept alive: 0; route allocated: 0 of 20; total allocated: 0 of 100]
2020-10-21 00:46:18.092 DEBUG 18188 --- [        main] h.i.c.PoolingHttpClientConnectionManager : Connection leased: [id: 0][route:
{}->http://localhost:8080][total kept alive: 0; route allocated: 1 of 20; total allocated: 1 of 100]
2020-10-21 00:46:18.096 DEBUG 18188 --- [        main] o.a.http.impl.execchain.MainClientExec : Opening connection {}->http://localhost:8080
2020-10-21 00:46:18.103 DEBUG 18188 --- [        main] .i.c.DefaultHttpClientConnectionOperator : Connecting to localhost/127.0.0.1:8080
2020-10-21 00:46:18.105 DEBUG 18188 --- [        main] .i.c.DefaultHttpClientConnectionOperator : Connection established 127.0.0.1:61511<->127.0.0.1:8080
2020-10-21 00:46:18.106 DEBUG 18188 --- [        main] h.i.c.DefaultManagedHttpClientConnection : http-outgoing-0: set socket timeout to 3000
2020-10-21 00:46:18.107 DEBUG 18188 --- [        main] o.a.http.impl.execchain.MainClientExec : Executing request GET
/api/v1/compy/f91a0667-4d12-4cab-b2ec-965cd11f8b79 HTTP/1.1
2020-10-21 00:46:18.107 DEBUG 18188 --- [        main] o.a.http.impl.execchain.MainClientExec : Target auth state: UNCHALLENGED
2020-10-21 00:46:18.109 DEBUG 18188 --- [        main] o.a.http.impl.execchain.MainClientExec : Proxy auth state: UNCHALLENGED
2020-10-21 00:46:18.113 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 >> GET ...

2020-10-21 00:46:18.148 DEBUG 18188 --- [        main] org.apache.http.wire : http-outgoing-0 << "63[\r][\n]"
2020-10-21 00:46:18.148 DEBUG 18188 --- [        main] org.apache.http.wire : http-outgoing-0 <<
{"id":"cc87cd9a-c0a4-42c0-aa8a-3b1cd273204d","compyName":"iCore5","compyStyle":"office","upc":null}[\r][\n]"
2020-10-21 00:46:18.154 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << HTTP/1.1 200
2020-10-21 00:46:18.154 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << Content-Type: application/json
2020-10-21 00:46:18.154 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << Transfer-Encoding: chunked
2020-10-21 00:46:18.154 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << Date: Tue, 20 Oct 2020 21:46:18 GMT
2020-10-21 00:46:18.155 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << Keep-Alive: timeout=60
2020-10-21 00:46:18.155 DEBUG 18188 --- [        main] org.apache.http.headers : http-outgoing-0 << Connection: keep-alive
2020-10-21 00:46:18.171 DEBUG 18188 --- [        main] o.a.http.impl.execchain.MainClientExec : Connection can be kept alive for 60000 MILLISECONDS
2020-10-21 00:46:18.217 DEBUG 18188 --- [        main] org.apache.http.wire : http-outgoing-0 << "0[\r][\n]"
2020-10-21 00:46:18.217 DEBUG 18188 --- [        main] org.apache.http.wire : http-outgoing-0 << "[\r][\n]"
2020-10-21 00:46:18.217 DEBUG 18188 --- [        main] h.i.c.PoolingHttpClientConnectionManager : Connection [id: 0][route:
{}->http://localhost:8080] can be kept alive for 60.0 seconds
2020-10-21 00:46:18.217 DEBUG 18188 --- [        main] h.i.c.DefaultManagedHttpClientConnection : http-outgoing-0: set socket timeout to 0

```