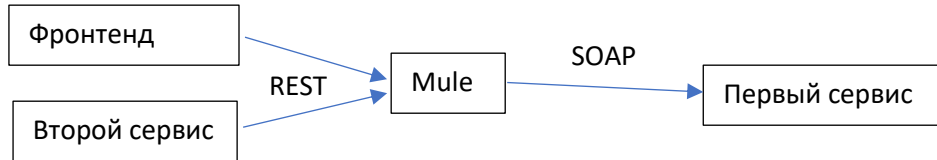


Тьюториал по 4 лабе SOA

Если у вас на SOAP **ВЫЗЫВАЕМЫЙ** сервис



1. Добавляете SOAP контроллер на вызываемый сервис (JAX-WS для Java EE, Spring WS для спринга), рестоный контроллер удалять пока не надо. По этому шагу есть куча примеров в интернете.
2. Скачиваете mule ESB, оформляете пробную версию на месяц
3. Заходите на <https://anypoint.mulesoft.com/designcenter>, регаетесь там, создаете новую спецификацию API, внимательно и полностью делаете спецификацию со всеми типами данных ошибками и так далее. Нужно делать очень внимательно и проверять, чтобы все типы данных соответствовали (например если это дробное число, нужно поставить тип number, а в advanced параметрах уточнить, что это float). Если на этом этапе сделать ошибки, mule дальше может не заработать.

Еще важно, чтобы на каждый URL был включен метод OPTIONS, это надо, чтобы CORS работал. Когда все сделали, публикуете спецификацию.

Мой пример https://anypoint.mulesoft.com/exchange/7320a393-d3ee-413c-9a9a-eff41fe87eca/soa_lab4/minor/2.0/

4. Запускаете Mule, создаете новый проект. Во вкладке API Implementation выбираете Download RAML from Design Center. Вводите ссылку на свой API.
5. Mule создаст листенер и потоки на каждый запрос из спецификации. Заходите в конфиг листенера, ставите нужный порт, также можно настроить https.
6. В каждый поток добавляете web service consumer. Для него создаете новый конфиг, где указываете адрес первого сервиса и ссылку на его WSDL

Web Service Consumer Config
Default configuration

General Advanced Notes Help

Name: Web_Service_Consumer_Config

Connection

General Security Transport Advanced

Soap version: SOAP11 (Default)

Mtom enabled:

Encoding:

Connection

Wsdl location: http://localhost:25080//SoapTicketServiceImplService?wsdl

Service: SoapTicketServiceImplService

Port: SoapTicketServiceImplPort

Address: http://localhost:25080/SoapTicketServiceImplService

После указания конфига ставите в каждом потоке нужную функцию

General

Operation: getTickets

7. Перед каждым консьюмером ставите message transform. Настраиваете маппинг параметров на аргументы вызовов

```

queryParams: Object
  orderBy: Array<String>=i
  id: Number?
  name: String?
  createDate: String?
  price: Number?
  comment: String?
  type: String?
  event: String?
  pageNumber: Number?
  pageSize: Number?
  queryString: String

```

```

Xml<getTickets>
  > getTickets: Object
    > arg0: Object?

```

```

1 @ %dw 2.0
2 output application/xml skipNullOn="everywhere"
3 ns ns0 http://endpoint.soap.vgorash.com/
4 ---
5 {
6   ns0#getTickets: {
7     arg0: {
8       pageNumber: attributes.queryParams.pageNumber,
9       pageSize: attributes.queryParams.pageSize,
10      id: attributes.queryParams.id,
11      name: attributes.queryParams.name,
12      createDate: attributes.queryParams.createDate,
13      price: attributes.queryParams.price,
14      comment: attributes.queryParams.comment,
15      type: attributes.queryParams.type,
16      events: attributes.queryParams.event,
17      orderBy: attributes.queryParams.orderBy
18     }
19   }
20 }

```

Module updates available
Review and update modules used in your p

ВАЖНО: если какие-то аргументы не обязательны, нужно дописать после output type **skipNullOn="everywhere"**

ЕЩЕ ВАЖНО: у меня был баг, когда я маппил параметр из request body, в аргумент передавалось пустое значение. Решение вот такое:

```

1 @ %dw 2.0
2 output application/xml
3 ns ns0 http://endpoint.soap.vgorash.com/
4 ---
5 {
6   ns0#addTicket: payload mapObject (v, k, i) -> {arg0: v}
7 }

```

8. После консьюмера ставите transform message. Возвращаемый тип выбираете в зависимости от варианта.

The screenshot shows the MuleSoft Studio interface. On the left, the 'Payload' structure is expanded, showing 'body: Object?' and 'addTicketResponse: Object'. The 'Attributes' section shows 'protocolHeaders: Object' and 'additionalTransportData: Object'. On the right, the 'transform message' configuration is shown with the following XML snippet:

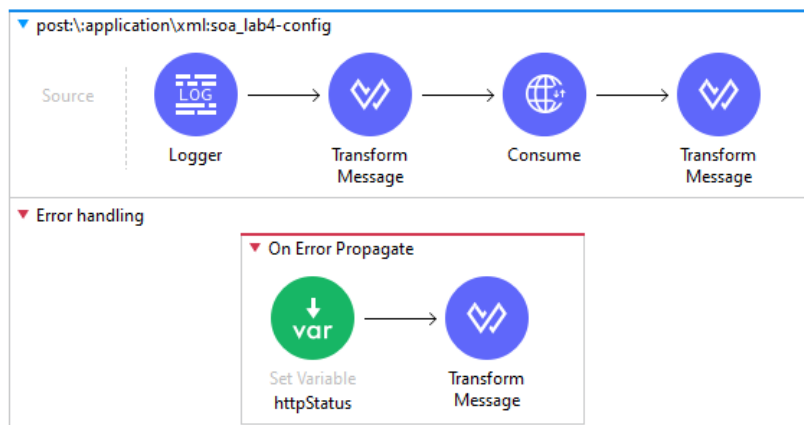
```
1 <?xml version='2.0'>
2   <output application/xml>
3     <ns ns0='http://endpoint.soap.vgorash.com/'>
4       <---
5     <ticket>
6       <payload.body.ns0#addTicketResponse.return>
```

9. Добавляете обработку ошибок: нужно поставить соответствующий код возврата и передать сообщение

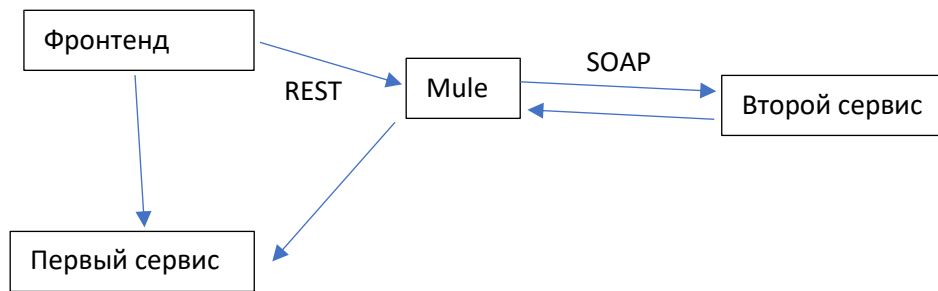
The screenshot shows the MuleSoft Studio interface. The 'General' tab is selected, showing the 'Display Name' as 'httpStatus'. The 'Settings' section shows the 'Name' as 'httpStatus' and the 'Value' as '404'. Below this, the 'Output Payload' is shown with the following XML snippet:

```
1 <?xml version='2.0'>
2   <output application/json>
3     <---
4     <error.detailedDescription>
```

10. В итоге все должно выглядеть примерно так



Если у вас на SOAP **ВЫЗЫВАЮЩИЙ** сервис



Насколько я понимаю, это должно выглядеть примерно так. Но тут лучше спросить у препода. Если я прав, то работа состоит из двух частей.

1. Переписать API второго сервиса на SOAP. Сделать REST прослойку как я описал выше.
2. Заставить второй сервис обращаться к первому тоже через SOAP, добавить в Mule какой-то конвертер soap в рест. Как это сделать я к сожалению не знаю