**SOAP**

Используется поверх http. Язык xml.

**REST**

Не протокол, а архитектурный стиль со своими принципами.

Используется протокол http и глаголы (GET,POST,PUT,DELETE). RESTful-системы в идеале должны удовлетворять семантики глаголов, но на практике может встречаться метод POST, который выполняет функции не только POST, но и GET.

Доступ к ресурсам: id объекта – это часть url, например /book/3/

Использование JSON для передачи сообщений.

Запрос REST должен содержать все то, чтобы сервер вернул нам ответ. Это забота клиента. Мы не обязываем сервер помнить, что он нам отвечал до этого.

**Контракт**

Контракт как интерфейс в JAVA, т.е. когда реализация интерфейса обязует класс реализовать методы этого интерфейса. Так и контракт для сервисов – это когда сервис в ответ на какой-то запрос обязуется возвращать сообщение определенного вида с определенными полями, определенного типа.

Контракт для SOAP – это WSDL и язык описания структуры XML Schema.

Контракт для REST – это методы GET, PUT, POST, DELETE + URL и язык описания структуры JSON Schema.

**Contract Test**

Проверка соблюдения условий контрактов.

Контрактные тесты – является разновидностью между интеграционными и unit-тестами.

**Spring**

Фраемворк для разработки серверных web-приложений. Состоит из разных модулей, который можно комбинировать как конструктор.

Группа библиотек Spring Cloud, в которой есть Spring Cloud Contract.

**Spring Cloud Contract**

Ручка или endpoint – это некий API-метод, пример:

|  |
| --- |
| @GetMapping("/bets/{userId}")  public ResponseEntity<List<Bet>> getBets(@PathVariable("userId") String userId) {  List<Bet> bets = service.getByUserId(userId);  if (bets.isEmpty()) {  return ResponseEntity.noContent().build();  }  return ResponseEntity.ok(bets);  } |

Для GET аннотация @GetMapping + указание url.

Для POST аннотация @PostMapping + указание url.

@PathVariable – path-параметры, заданные в url аннотации.

Аннотация навешивается над методом.

Java-объект автоматически Spring-ом преобразуется в JSON и вернется в виде JSON.

Хорошая практика выносить логику в сервисы. Наш контролер должен быть максимально тупой (по феншую).

Для того, чтобы написать контрактный тест на Spring Cloud Contract нужно использовать либо groovyDSL, либо yaml, т.е. файл, в котором будет прописана конфигурация нашего теста. Описаник контракта на языке yaml:

|  |
| --- |
| org.springframework.cloud.contract.spec.Contract.make {  request {  method 'GET'  urlPath '/bets/2'  }  response {  status 200  headers {  header('Content-Type', 'application/json')  }  body('''  {  "sport":"football",  "amount": 1  }  '''  )  }  } |

Будет сгенерирован следующий тест. По умолчанию будет использована библиотека RestAssured.

|  |
| --- |
| public class UserControllerTest extends ContractBase {  @Test  public void validate\_get\_200() throws Exception {  // given:  MockMvcRequestSpecification request = given();  // when:  ResponseOptions response = given().spec(request)  .get("/bets/2");  // then:  assertThat(response.statusCode()).isEqualTo(200);  assertThat(response.header("Content-Type")).isEqualTo("application/json");  // and:  DocumentContext parsedJson = JsonPath.parse(response.getBody().asString());  assertThatJson(parsedJson).field("['amount']").isEqualTo(1);  assertThatJson(parsedJson).field("['sport']").isEqualTo("football");  } |

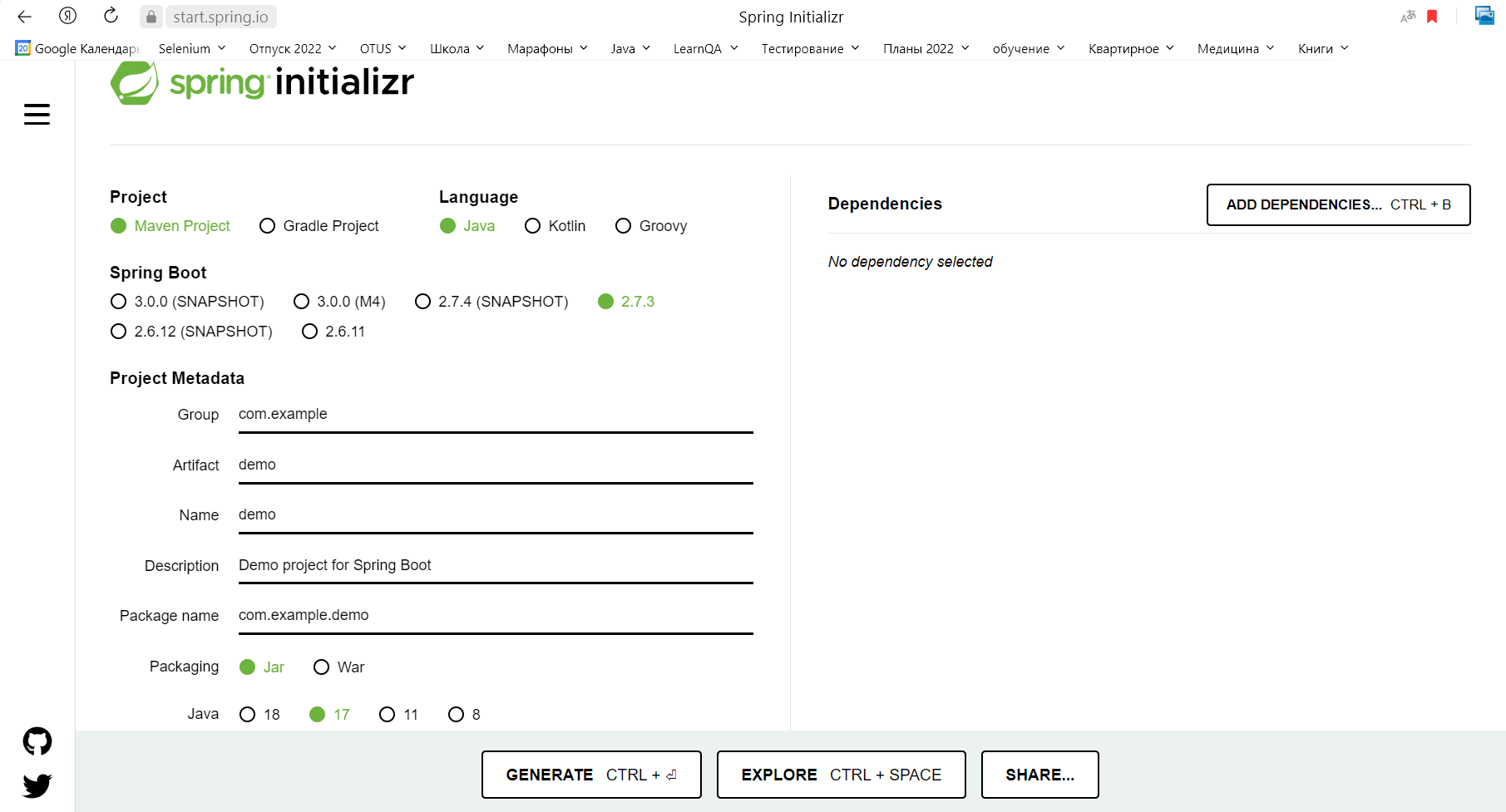
**Практика**

Фейковое API: <https://reqres.in/>

Будем использовать:

1. Заглушка проекта Spring
2. http-клиент: retrofit (основной для Android). Им будем отправлять запросы. Для него подключаем Jackson.
3. Junit

Генератор проекта: <https://start.spring.io/>



Пишем два теста

1. Получение пользователя по id
2. Создание пользователя

Retrofit состоит из двух частей (в папке api):

1. Интерфейс, в кот.мы описываем наши endpoint
2. Класс, где мы конструируем наш http-клиент, которому мы будем скармливать наш интерфейс