Ещё раз здравствуйте, начну с первой части задания, с критического взгляда на схему представленной сущности.

Во-первых, вызывает вопрос, почему в таблице два поля отвечающих за идентификацию «id» и «documentId»? Если «documentId» это поле связка, ссылка на идентификатор в другой таблице, то понятно к чему оно здесь. Но в исходных данных об это не говорится, значит можно выкинуть это поле, поскольку появляется излишнее дублирование данных. Также стоит отметить, что присваивать такому важному полю тип «nullable» и оставлять его без значения по умолчанию - не очень хорошая идея, лучше генерировать универсальный идентификатор автоматически, чем оставлять его пустым, в фазе выполнения присвою этому полю генерируемое значение.

Во-вторых, поле «documentDate» имеет формат строки, что, исходя из названия поля, неверное присвоение типа данных. У поля отвечающего за дату, формат данных должен быть соответственно «Date» или «TimeStamp». Логично присвоить ему один из соответствующих форматов. Да, дату можно передавать в виде строки, каждый раз можно парсить и форматировать дату в unix-время, передавая количество секунд с момента начала эпохи unix, но ведь намного удобнее использовать для этого специально предназначенные типы данных.

В-третьих, поля «dictionaryValueId» и «dictionaryValueName» оба имеют тип данных «UUID», что наталкивает на мысль о том, что и то и другое планируется использоваться как универсальный уникальный идентификатор. Исходя из названия и типа данных, оба поля несут в себе разный функционал, но имеют один и тот же тип данных. Может ввести в заблуждение не совпадение названия и типа данных, имеет смысл оставить как есть одно поле - «dictionaryValueId», а «dictionaryValueName» сделать в формате строки, тогда сходу понятно за что отвечает каждое поле. Снова идентификационное поле остается типа «nullable, следует сделать его обязательным и генерируемым в случае пустого значения, по аналогии с полем «id».

В-четвертых, поле «sortOrder», если у нас есть несколько порядков сортировки, то это поле составлено верно, если у нас всего два варианта сортировки, например, «по возрастанию» и «по убыванию», то имеет смысл установить тип данных «Boolean» и переключать сортировку между этими двумя вариантами или придать полю значение «null», если сортировка не нужна.

Далее, переменные внутри схемы сущности, в частности «testId», который является универсальным уникальным идентификатором, что на мой взгляд является лишним, ведь уже существует главное поле «id», котрой однозначно идентифицирует запись в таблице БД. Возможно, что это поле является ссылочным, но об этом нигде не говорится, поэтому не следует множить сущности без необходимости и избавиться от него. «testName» имеет значение по умолчанию и является переменным, не может являться пустым полем, к нему вопросов нет.

По итогу должна получиться следующая схема сущности:

```
@Entity
data class TestEntity(
    @Id
    @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "uuid2")
    val id: UUID = UUID.randomUUID(),
    val documentDate: Date? = null,
    @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "uuid2")
    val dictionaryValueId: UUID = UUID.randomUUID(),
    val dictionaryValueName: String? = null,
    val sortOrder: String? =null
) {
    var testName: String = "Test"
}
```

Перейдем к реализации. По заданию нужно реализовать liquibase-скрипт, создающий таблицу по описанной схеме данных, далее представлен результат написания скрипта:

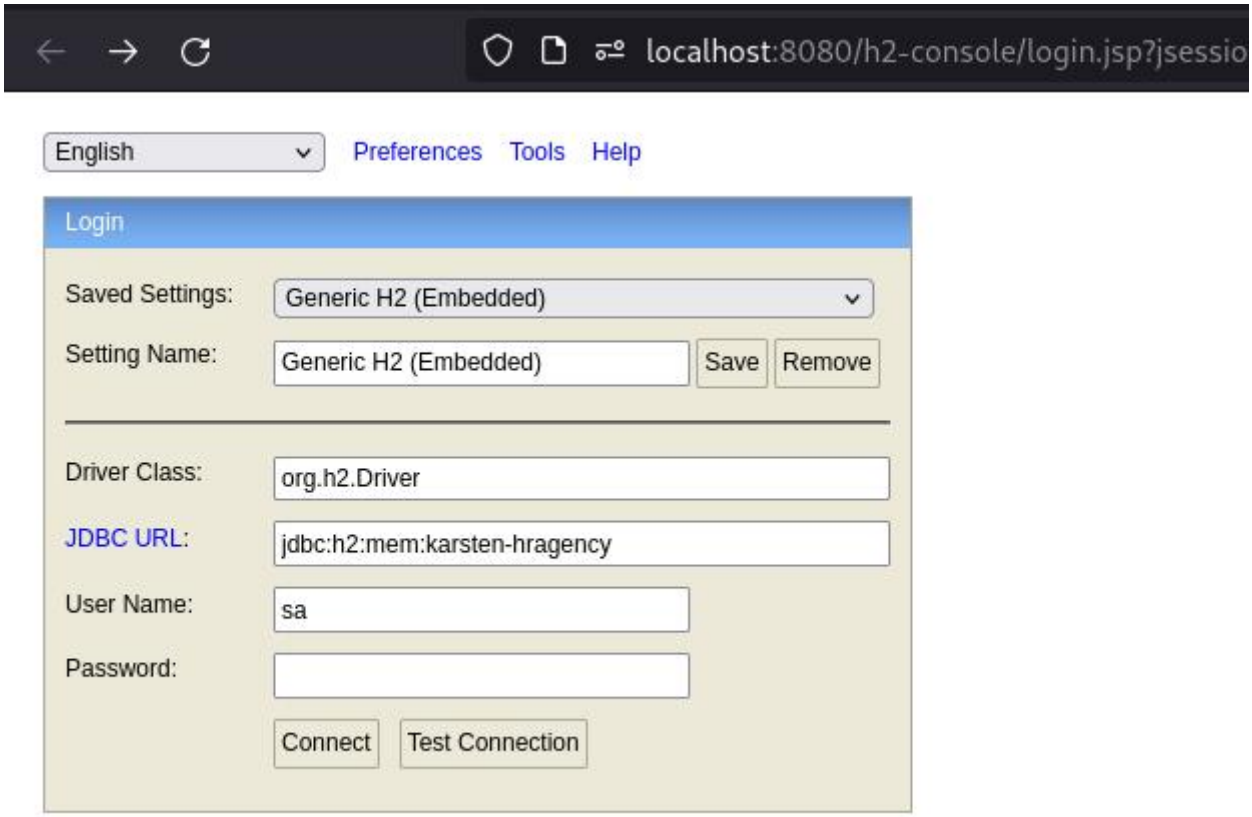
```yaml
databaseChangeLog:
 - changeSet:
    id: 1
    author: karsten
    changes:
     - createTable:
        tableName: test_entity
        columns:
         - column:
            defaultValueComputed: RANDOM_UUID()
            name: id
            type: UUID
            constraints:
             primaryKey: true
             nullable: false
         - column:
            name: document_date
            type: TIMESTAMP
            constraints:
             nullable: true
         - column:
            defaultValueComputed: RANDOM_UUID()
            name: dictionary_value_id
            type: UUID
            constraints:
             nullable: true
         - column:
            name: dictionary_value_name
            type: VARCHAR(50)
            constraints:
             nullable: true
         - column:
            name: sort_order
            type: VARCHAR(50)
            constraints:
             nullable: true
         - column:
            name: test_name
            type: VARCHAR(50)
            constraints:
             nullable: true
```

Проверить правильность выполнения скрипта можно заглянув в БД, для целей тестирования внутри нашего проекта, создадим пустую БД под управлением СУБД H2, заполним её тестовыми данными выполнив следующие sql-запросы:

```sql
INSERT INTO test_entity (document_date, dictionary_value_name, sort_order) VALUES (DATE '2004-09-27', 'First dictionary', 'ascending');
INSERT INTO test_entity (document_date, dictionary_value_name, sort_order) VALUES (DATE '2004-09-27', 'Second dictionary', 'descending');
```

```
INSERT INTO test_entity (document_date, dictionary_value_name, sort_order) VALUES (DATE '2004-09-27', 'Third dictionary', null);
```

И заглянем в консоль администратора, предварительно не забыв включить её в файле конфигурации проекта:



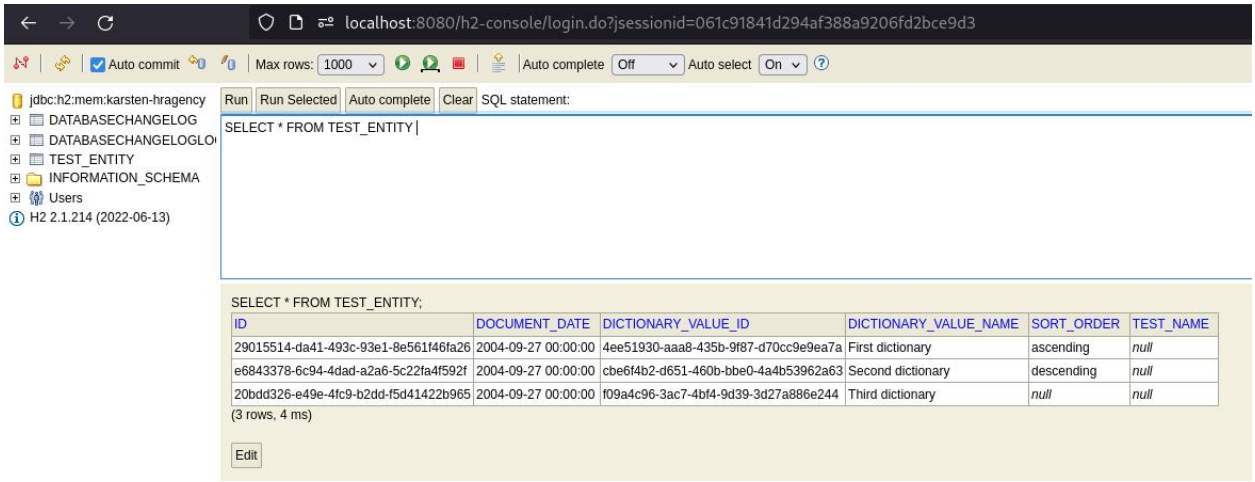
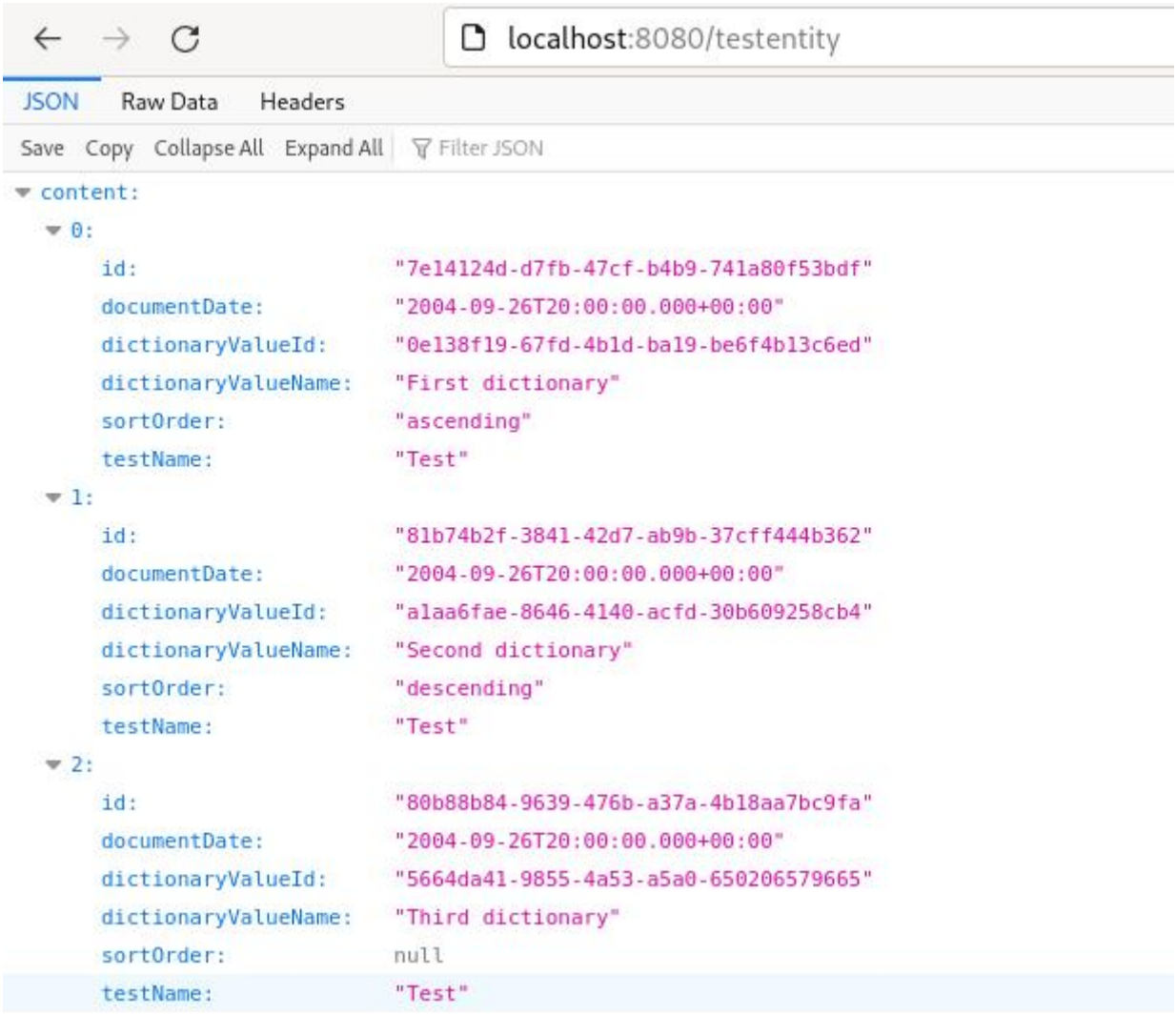


Таблица успешно создана в нашей тестовой базе данных. Перейдем ко второй части задания, создадим контроллер, с базовым набором CRUD-операций. Реализация сущности уже была представлена в листинге выше, далее приведена реализация контроллера:

```
@RestController
class EntityController(
    @Autowired  val entityService: TestEntityService
) {
```

```
    companion object {
        const val PARAM_PATH_ID: String = "id"
        const val URI_PATH_ENTITY: String = "/testentity"
        const val URI_PATH_ENTITY_ID: String = "$URI_PATH_ENTITY/{$PARAM_PATH_ID}"
    }

    @GetMapping(URI_PATH_ENTITY)
    @ResponseStatus(HttpStatus.OK)
    fun getAllEntities(@PageableDefault(value = 50)  pageable: Pageable):
Page<TestEntityDTO> =
        entityService.findAll(pageable)

    @GetMapping(URI_PATH_ENTITY_ID)
    @ResponseStatus(HttpStatus.OK)
    fun getEntityById(@PathVariable(PARAM_PATH_ID) id: UUID): Optional<TestEntityDTO> =
        entityService.findById(id)

    @PostMapping(URI_PATH_ENTITY)
    @ResponseStatus(HttpStatus.CREATED)
    fun postEntity(@RequestBody entity: EntityRequestBody): TestEntityDTO =
        entityService.save(entity)

    @PutMapping(URI_PATH_ENTITY_ID)
    @ResponseStatus(HttpStatus.ACCEPTED)
    fun putEntity(@PathVariable(PARAM_PATH_ID) id: UUID,
                  @RequestBody entity: EntityRequestBody): TestEntityDTO =
        entityService.update(id, entity)

    @PatchMapping(URI_PATH_ENTITY_ID)
    @ResponseStatus(HttpStatus.ACCEPTED)
    fun patchEntity(@PathVariable(PARAM_PATH_ID) id: UUID,
                    @RequestBody entity: EntityRequestBody): TestEntityDTO =
        entityService.patch(id, entity)

    @DeleteMapping(URI_PATH_ENTITY_ID)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    fun deleteEntity(@PathVariable(PARAM_PATH_ID) id: UUID) =
        entityService.deleteEntity(id)
}
```

Проверим корректность работы контроллера выполнив GET-запрос к списку всех сущностей, сделаем это самым тривиальным способом, откроем нужную страницу в браузере:

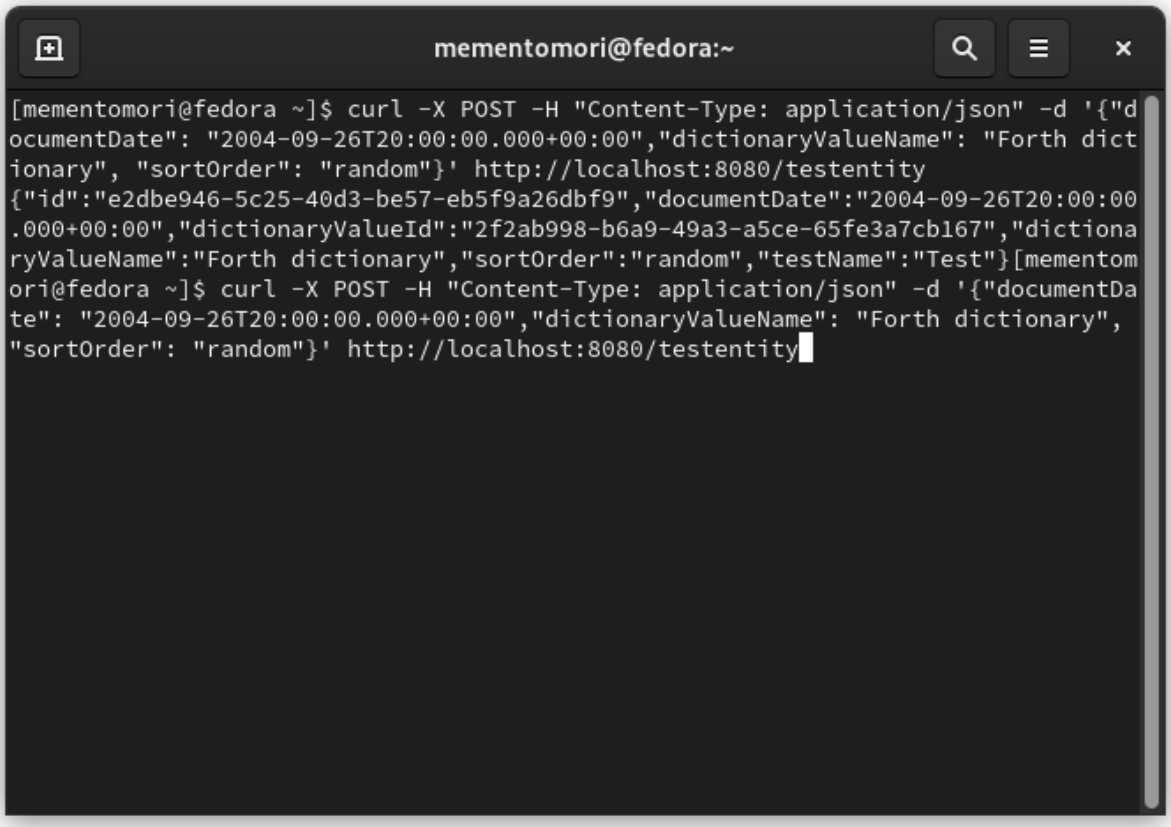


Результат аналогичный выводимому в консоли администратора БД, тестовые данные остаются неизменными.

Протестируем POST-запросы с помощью утилиты «curl», выполним тестовый запрос:

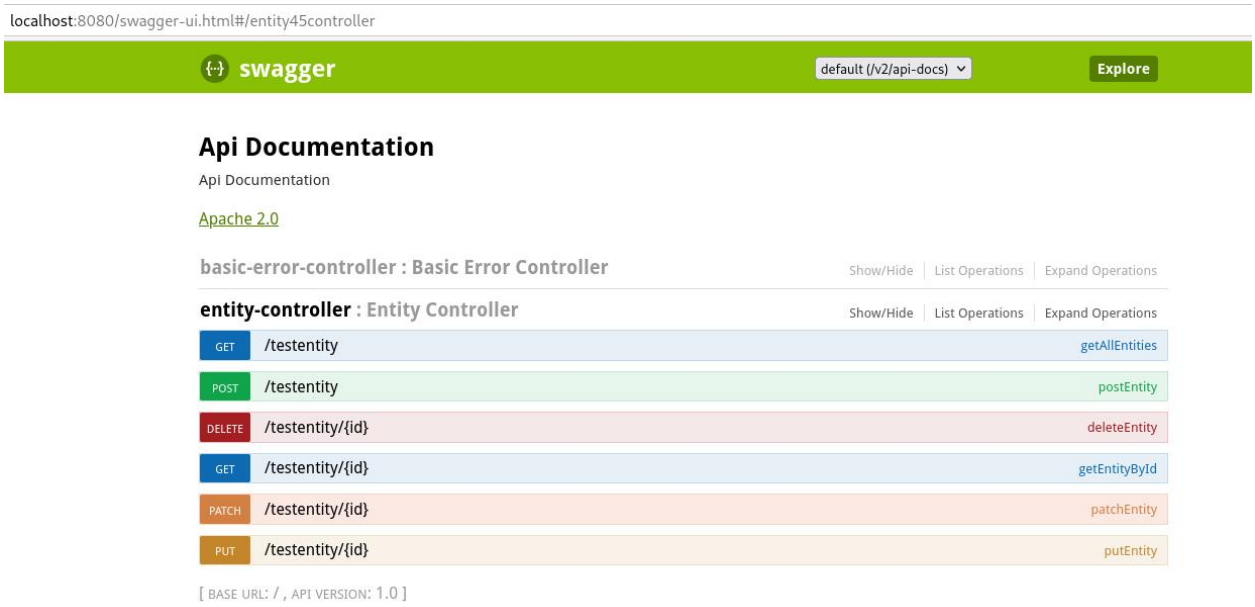
curl -X POST -H "Content-Type: application/json" -d '{"documentDate": "2004-09-26T20:00:00.000+00:00","dictionaryValueName": "Forth dictionary", "sortOrder": "random"}' http://localhost:8080/testentity

И получим следующий результат:



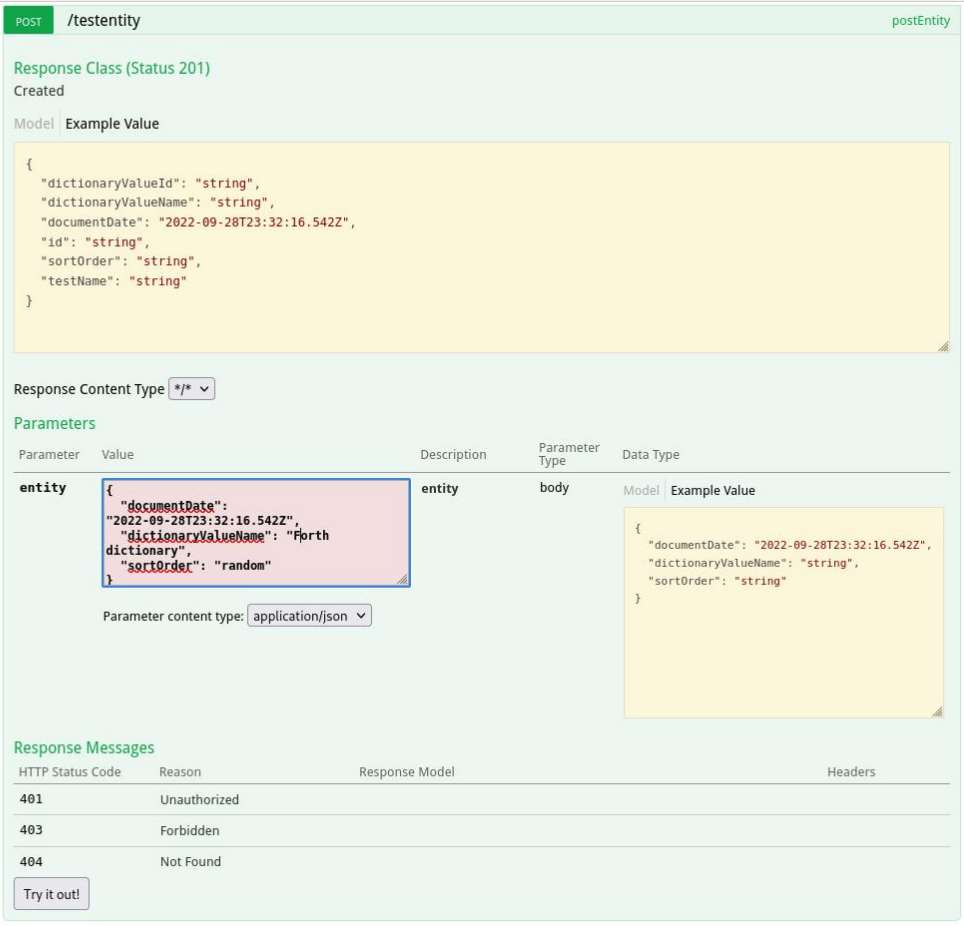
Новая запись успешно создана, в ответ, сервис возвращает созданную запись.

Для наглядности подключим к нашему проекту «springfox swagger ui» и сгенерируем минимальную документацию к нашему API. Проверим результат перейдя по адресу localhost:8080/swagger-ui.html#:



Проверим работу сервиса прямо внутри «swagger ui» сделав аналогичный POST-запрос, который ранее выполнялся с помощью curl :

localhost:8080/swagger-ui.html#!/entity45controller/postEntityUsingPOST



И ответ от сервиса:

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
   "documentDate": "2022-09-28T23:32:16.542Z", \
   "dictionaryValueName": "Forth dictionary",\
   "sortOrder": "random" \
 }' 'http://localhost:8080/testentity'
```

Request URL

```
http://localhost:8080/testentity
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body


```
{
  "id": "d9c3c83d-7078-4978-ba0d-e6261a54ea35",
  "documentDate": "2022-09-28T23:32:16.542+00:00",
  "dictionaryValueId": "3ed8e767-dd05-459f-b4a9-c409a8da6fb9",
  "dictionaryValueName": "Forth dictionary",
  "sortOrder": "random",
  "testName": "Test"
}
```


Response Code

```
201
```

Response Headers


```
{
  "connection": "keep-alive",
  "content-type": "application/json",
  "date": "Wed, 28 Sep 2022 23:47:53 GMT",
  "keep-alive": "timeout=60",
  "transfer-encoding": "chunked"
}
```


Запись добавлена в таблицу БД, сервис отвечает HTTP кодом 201 и возвращает созданную запись.

Более подробно изучить исходный код можно в моем репозитории на github:

https://github.com/mementomorri/liquibase-spring-data

Спасибо за предоставленную возможность проявить мои знания и навыки на практике.

С уважением, Алексей Карстен.