



DeepL

Подпишитесь на DeepL Pro и переводите документы большего объема.
Подробнее на www.DeepL.com/pro.



Быть RESTful

Разработка качественного API

Резюме: задачи в этом проекте научат вас использовать механизмы Spring для разработки REST-приложений

Содержание

I	Преамбула	2
II	Инструкции	3
III	Правила проекта	5
IV	Упражнение 00 : Безопасность Spring	6
V	Упражнение 01 : JWT	10
VI	Упражнение 02: HATEOAS	12

Глава I

Преамбула

Уровни соответствия API приложения архитектуре REST в соответствии с моделью Ричардсона (каждый последующий уровень основан на предыдущем):

Уровень 0. В качестве транспортного протокола используется HTTP. Для всех взаимодействий используется один URI. Вся необходимая информация передается в виде обычного XML-текста.

Уровень 1. API использует концепцию "ресурса". Каждый ресурс - это отдельный бизнес-объект. Каждый ресурс имеет свой собственный URI. Все взаимодействия описываются одним HTTP-глаголом.

Уровень 2. Все взаимодействия описываются расширенным набором HTTP-глаголов: GET (получение объекта), POST (добавление нового объекта), PUT (обновление существующего объекта) и DELETE. Таким образом, для каждого ресурса определяется CRUD.

Уровень 3. API основан на формате Hypermedia.

Глава II

Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить раствор.
- Теперь для вас существует только одна версия Java - 1.8. Убедитесь, что компилятор и интерпретатор этой версии установлены на вашей машине.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код чаще читают, чем пишут. Внимательно прочитайте [документ](#), в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым [стандартам Oracle](#):
- Комментарии не допускаются в исходном коде вашего решения. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Чтобы ваше решение было оценено, оно должно находиться в вашем GIT-репозитории.
- Ваши решения будут оценивать ваши товарищи по аквариуму.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить свой .gitignore во избежание несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещено выводить предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- У вас есть вопрос? Спросите своего соседа справа. В противном случае попробуйте поговорить с соседом слева.
- Ваше справочное пособие: товарищи / Интернет / Google. И еще кое-что. На любой ваш вопрос есть ответ на Stackoverflow. Узнайте, как правильно задавать вопросы.
- Внимательно прочитайте примеры. В них могут потребоваться вещи, которые не указаны в предмете.
- Для вывода используйте "System.out".

- И да пребудет с вами Сила!
- Никогда не оставляйте на завтра то, что вы можете сделать сегодня ;)


Глава III

Правила проекта

- Решение для каждого упражнения представляет собой отдельный Maven-проект, реализованный на основе Spring Boot.
- Структура проекта - на усмотрение застройщика.
- Каждый проект должен содержать файл data.sql с набором тестовых данных.

Глава IV

Упражнение00 : Spring Security

	Упражнение 00
	Весенняя
	безопасность
Входящий каталог : ex00/	

Вам необходимо разработать API для управления учебным центром. Набор операций должен быть реализован для следующих доменных моделей:

- Пользователь
 - Имя
 - Фамилия
 - Роль (администратор, преподаватель, студент)
 - Вход в систему
 - Пароль
- Курс
 - Дата начала
 - Дата окончания
 - Имя
 - Учителя
 - Студенты
 - Описание
 - Уроки

- Урок
 - Время начала
 - Время окончания
 - День недели
 - Учитель

Задание должно быть выполнено в соответствии с требованиями REST API, например:

- Добавление нового урока в курс с идентификатором 42: POST
/courses/42/lessons

Тело запроса:

```
{
  "Время начала" : "10:00",
  "finishTime" : "12:00",
  "dayOfWeek" : "Monday",
  "teacherId" : 432
}
```

Тело ответа:

```
"урок": {
  "id" : 21,
  "Время начала" : "10:00",
  "finishTime" : "12:00",
  "dayOfWeek" : "Monday",
  "teacher" : {
    "id" : 432,
    "firstName" : "Лучший",
    "lastName" : "Учитель".
  }
}
```

Ниже приведен полный список операций для курсов, которые необходимо реализовать в рамках текущей задачи:

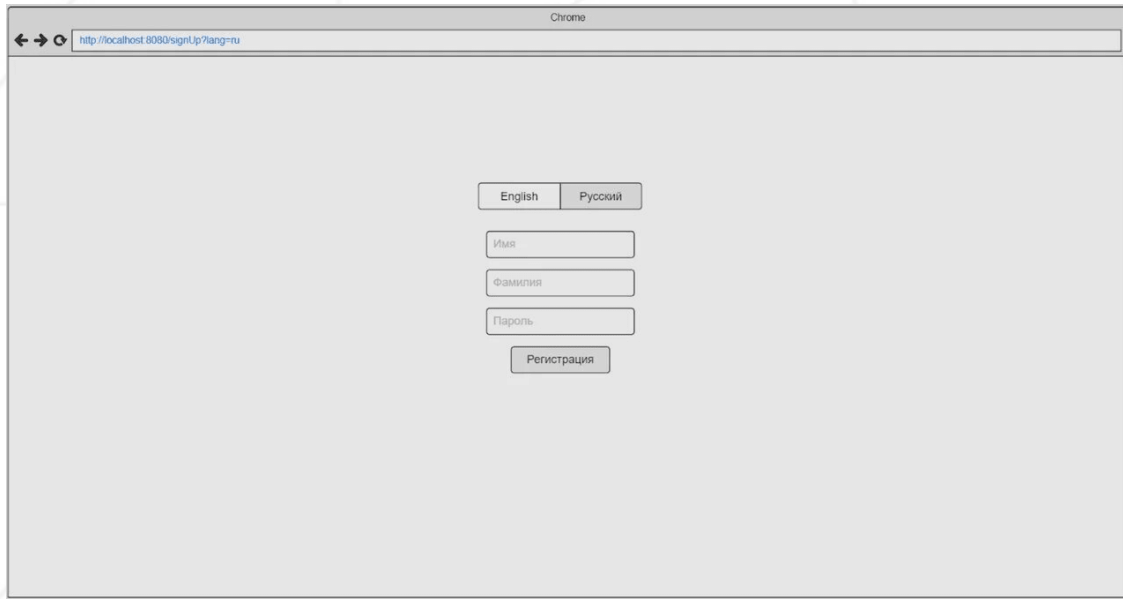
courses-controllers Courses Controllers	
GET	/courses getAllCourses
POST	/courses addCourse
GET	/courses/{course-id} getCourse
PUT	/courses/{course-id} updateCourse
DELETE	/courses/{course-id} deleteCourse
GET	/courses/{course-id}/lessons getLessonsByCourse
POST	/courses/{course-id}/lessons addLessonToCourse
PUT	/courses/{course-id}/lessons/{lesson-id} updateLessonInCourse
DELETE	/courses/{course-id}/lessons/{lesson-id} deleteLessonFromCourse
GET	/courses/{course-id}/students getStudentsByCourse
POST	/courses/{course-id}/students addStudentToCourse
DELETE	/courses/{course-id}/students/{student-id} deleteStudentFromCourse
GET	/courses/{course-id}/teachers getTeachersByCourse
POST	/courses/{course-id}/teachers addTeacherToCourse
DELETE	/courses/{course-id}/teachers/{teacher-id} deleteTeacherFromCourse

Добавление преподавателей и студентов в курс предполагает отправку только идентификатора пользователя в теле запроса (в отличие от добавления урока, где требуется отправка полной информации). Полномасштабная работа с сущностью User осуществляется в отдельном контроллере:

users-controller Users Controller	
GET	/users getAllUsers
POST	/users addNewUser
PUT	/users/{user-id} updateUser
DELETE	/users/{user-id} deleteUser

Дополнительные требования:

- Каждый метод, который извлекает коллекцию объектов, должен поддерживать механизм пагинации.
- Для удобного использования API необходимо интегрировать фреймворк Swagger в ваше приложение: <https://swagger.io/>.
- Для каждого метода также необходимо предоставить документацию с использованием Swagger, например:



Вам необходимо реализовать модульный тест хотя бы для одного GET, POST, PUT и DELETE- метода с использованием MockMvc, например:

```
@ExtendWith(SpringExtension.class)
@AutoConfigureMockMvc
@SpringBootTest
public class CoursesTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private CoursesService coursesService;

    @BeforeEach
    public void setUp() { when(coursesService.delete(1L)).
        then...;
    }

    @Test
    public void deleteCourseTest() throws Exception {
        mockMvc.perform(delete("/courses/1")).andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath(...));
    }
}
```

В случае некорректных операций (добавление несуществующего преподавателя/ученика в курс, указание неправильной роли пользователя и т.д.), вы должны вернуть ответ с кодом 400 и следующим содержанием:


```
"error": {
  "статус" : 400,
  "message" : "Плохой запрос"
}
```

Примечания:

- Spring Data REST не допускается для этой задачи

Глава V

Упражнение 01 : JWT

	Упражнение 01
	JWT
	Входящий каталог : <i>ex01/</i>
	Файлы для сдачи : Центр образования-
	папка Разрешенные функции : н/д

Используйте авторизацию JWT для реализации механизма предоставления доступа к ресурсам на основе ролей.

Теперь каждый запрос должен сопровождаться JWT-токеном со следующей информацией о пользователе:

- Идентификатор пользователя
- Роль пользователя
- Вход пользователя в систему

После авторизации пользователь отправляет POST-запрос на URL `/signUp` с логином и паролем. Если эти данные верны, пользователь получает JWT-токен, подписанный секретным ключом. Ключ хранится в файле `application.properties` приложения. Если данные авторизации неверны, возвращается статус 403.

Каждый запрос пользователя к ресурсам API должен содержать заголовок `Authorization` с пользовательским токеном JWT. В этом случае приложение не должно обращаться к базе данных для авторизации пользователя, поскольку вся необходимая информация хранится в токене.


Операции GET доступны для пользователя с любой ролью. Операции POST, PUT и DELETE доступны только администратору центра.

Примечания:

- Чтобы полностью реализовать аутентификацию с помощью JWT, необходимо реализовать компоненты Spring Security: Authentication, Filter, AuthenticationProvider.
- Spring Data REST не допускается для этой задачи

Глава VI

Упражнение 02 : HATEOAS

	Упражнение 02
HATEOAS	
Входящий каталог : ex02/	
Файлы для сдачи : Кинопапка	
Разрешенные функции : n/a	

Теперь давайте реализуем функциональность учебного центра, используя технологию Spring Data REST. Таким образом, весь API для сущностей User, Course и Lesson будет представлен в формате Hipermedia. Например, для запроса /course GET будет возвращен следующий ответ:

```
{
  "\_embedded": {
    "courses": [
      {
        "title": "Spring Data Rest",
        "description": "Лучший
        фреймворк", "state":
        "Опубликовано",
        "\_links": {
          "self": {
            "href": "http://localhost /courses/1"
          },
          "курс": {
            "href": "http://localhost /courses/1"
          },
          "уроки": {
            "href": "http://localhost /courses/1/lessons".
          },
          "студенты": {
            "href": "http://localhost /courses/1/students".
          }
        }
      }
    ],
    {
      "title": "SQL",
      "описание": "Все о РСУБД",
      "состояние": "Черновик",
      "\_links": {
        "self": {
          "href": "http://localhost /courses/2"
        },
        "курс": {
```



```
    },
    "опубликовать": {
      "href": "http://localhost /courses/2/publish".
    },
    "уроки": {
      "href": "http://localhost /courses/2/lessons".
    },
    "студенты": {
      "href": "http://localhost /courses/2/students".
    }
  }
}
],
"_links": {
  "self": {
    "href": "http://localhost /courses"
  },
  "профиль": {
    "href": "http://localhost /profile /courses".
  },
  "search": {
    "href": "http://localhost /courses/search".
  }
},
"page": {
  "размер": 20,
  "totalElements": 2,
  "totalPages": 1,
  "число": 0
}
}
```

Как видно из примера, для сущности Course необходимо реализовать возможность публикации с помощью POST-запроса /courses/2/publish. Курс, находящийся в состоянии DRAFT, может быть опубликован. После публикации он переходит в состояние PUBLISHED и не может быть опубликован повторно.

Должна быть обеспечена возможность работы с API через HAL Browser.

Также необходимо обеспечить автоматическую генерацию документации adoc для метода публикации курса на основе модульного тестирования этого метода. Пример такой документации:

Course API

Methods

Course publish

You can publish a course with status **DRAFT**

request

```
PUT /courses/1/publish HTTP/1.1
Host: localhost:8080
```

response

```
HTTP/1.1 200 OK
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/hal+json
Content-Length: 93

{
  "title" : "Spring 5",
  "description" : "All about Spring",
  "state" : "PUBLISHED"
}
```

Table 1. response-fields

Path	Type	Description
title	String	Title of course
description	String	Description of course
state	String	State of course