REST API в Spring Boot

- Ориентированность на ресурсы
  - Дополнительные действия
  - Префикс /api
- Коды ответа

Spring Boot в основном используется для создания API. В этом уроке вы узнаете, что это такое, зачем нужно и как с этим работать.

По отношению к веб-приложениям, API — это набор маршрутов, созданных для использования другими сервисами или приложениями. API обычно возвращают данные в JSON-формате, чтобы их было удобно читать и обрабатывать.

Например, такой API есть у GitHub. Оно активно используется в редакторах с интеграцией с GitHub API, а также в онлайн-инструментах для создания десктопных приложений и даже консольных утилит:

```
// gh — это имя утилиты в корне проекта
gh pr status

Relevant pull requests in cli/cli

Current branch
There is no pull request associated with [fix-homepage-bug]

Created by you
You have no open pull requests

Requesting a code review from you
#100 Fix footer on homepage [fix-homepage-footer]
✓ Checks passing - Review pending
```

Способов создавать API много. Самым распространенным и базовым считается REST API. Это архитектурный подход, очень сильно полагающийся на принципы работы самого HTTP. Далее мы будем изучать особенности этого подхода и строить наш код на его основе.

Ориентированность на ресурсы

В REST-подходе маршруты строятся вокруг ресурсов. Действия над ними в основном определяются HTTP-методами. Так выглядят маршруты типичного CRUD:

Метод	Маршрут	Описание
GET	/users	Список пользователей
GET	/users/{id}	Пользователь
POST	/users	Создание нового пользователя
PUT	/users/{id}	Обновление пользователя
DELETE	/users/{id}	Удаление пользователя

Здесь ресурс — это пользователь. Обычно во фреймворках берут имя ресурса, ставят его во множественное число и делают частью маршрута. А вот в коде все по-другому: обработчики одного ресурса помещают в общий класс, называемый **контроллером**. Сам класс располагают в соответствующей директории:

```
# Гипотетический пример
src/main/java/io/hexlet/spring/controller
├── PostsController.java
├── PagesController.java
└── UsersController.java
```

С точки зрения Spring Boot контроллером считается любой класс с аннотацией `@RestController`. Для пользователя он будет выглядеть так:

```
@RestController
public class UsersController {
    // Здесь расположен код обработчиков
}
```

Это не означает, что все обработчики, связанные с каким-то ресурсом, находятся ровно в одном контроллере. У ресурсов бывают разные представления, которые требуют других обработчиков для одних и тех же действий. Представьте себе ссылку */companies/google/users* для маршрута */companies/{id}/users*. Такой маршрут может

выводить сотрудников конкретной компании, у которых есть собственное представление и собственные действия. Здесь для работы с этими пользователями в коде будет свой контроллер:

```
# Пример с вложенной структурой
src/main/java/io/hexlet/blog/controller
├── PostsController.java
├── PagesController.java
├── UsersController.java
└── companies
    └── UsersController.java
```

Дополнительные действия

CRUD — это базовый набор операций, но в реальных проектах встречаются запуски процессов, переводы сущностей в новые состояния и другие дополнительные действия. Предположим, что мы хотим опубликовать статью на сайте. Чтобы это сделать, нужен дополнительный обработчик со своим маршрутом:

```
// Внутри PostsController
// PATCH используется для частичного обновления
@PatchMapping('/posts/{id}/publish')
// Здесь код обработчика
```

Префикс */api*

Этот пункт не относится напрямую к понятию REST, но считается общепринятой практикой. Если маршрут считается частью API, то его располагают под префиксом */api*. Например, маршрут */users* превращается в */api/users*. То же самое происходит со всеми остальными маршрутами. Иногда этого недостаточно и вводится версионирование: */api/v1/users*.

В рамках курса мы остановимся на варианте */api/users*. Чтобы добиться такого результата, можно добавить этот префикс в каждый маршрут, но получится дублирование. Упростить эту задачу можно с помощью аннотации `@RequestMapping`, которой можно пометить контроллер:

```
import org.springframework.web.bind.annotation.RequestMapping;
// Остальные импорты

@RestController
@RequestMapping("/api")
public class UsersController {
    @GetMapping("/users")
    public String index() { /* код */ }
}
```

Если контроллеров больше одного, то придется задавать эту аннотацию в каждом контроллере.

Spring Boot дает возможность указать префикс на уровне всего приложения, но так лучше не делать, потому что пропадет возможность создавать что-то вне */api*, и перестанет работать главная страница.

Коды ответа

Другой важный элемент в REST API — это коды ответа. В этой части REST API полностью полагается на правильное использование стандарта HTTP. Например, для созданного ресурса правильный код ответа — это 201, а не 200. Если произошла внутренняя ошибка — это 500, а если ошибка валидации — это 422. Подробнее об ошибках можно прочитать здесь.

В Spring Boot коды возврата можно задавать несколькими способами. Иногда это происходит через `ResponseEntity`, но в большинстве ситуаций для этого достаточно аннотации `@ResponseStatus`:

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
// Остальные импорты

@RestController
@RequestMapping("/api")
public class UsersController {
    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    public User create() { /* код */ }
}
```

Далее →