Updating Resources

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Updating resources

- PUT for full updates
- PATCH for partial updates

Upserting

- Creating a resource with PUT or PATCH

PUT vs. PATCH

PUT is for full updates

- All resource fields are either overwritten or set to their default values

PATCH is for partial updates

- Allows sending over change sets via JsonPatchDocument

Demo

Updating a resource (part 1)

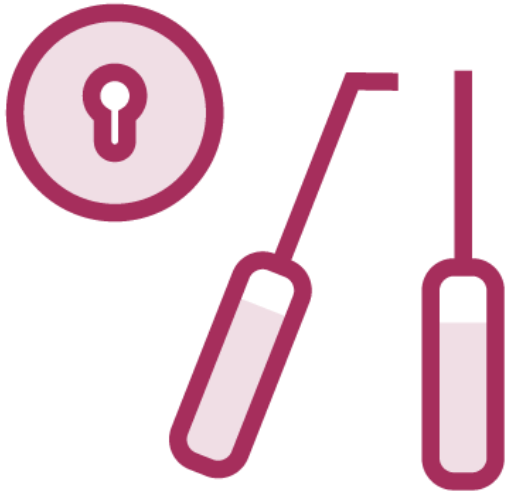
The Repository Pattern

An abstraction that reduces complexity and aims to make the code, safe for the repository implementation, persistence ignorant

Advantages of the Repository Pattern

Less code duplication

Less error-prone code



Better testability of the consuming class

Persistence ignorant

Switching out the persistence technology is not the main purpose. Choosing the best one for each repository method is.

The Repository Pattern

We're working on a contract, not on an implementation

Always have a set of methods matching the required functionality and call them, even if they don't do anything in the current implementation

Demo

Updating a resource (part 2)

Demo

Validating input when updating a resource with PUT

Updating Collection Resources

Sending a PUT request to a collection resource like http://host/api/authors/{authorId}/courses is allowed

- The course's resource would be overwritten with the new collection

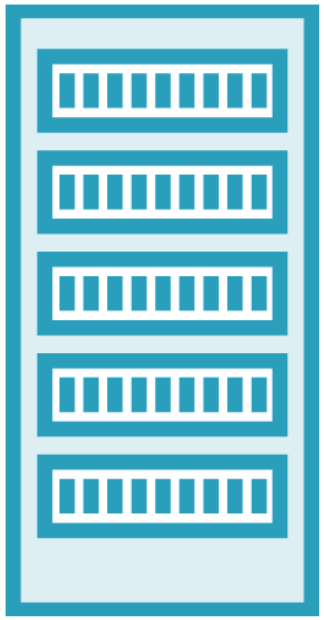
It's rarely implemented because it can be very destructive

Upserting

http://myapi/authors

http://myapi/authors/{guid}

http://myapi/authors/1



Upserting

Server is responsible for URI generation	Consumer is responsible for URI generation
PUT request must go to an existing URI	PUT request can be sent to an unexisting URI, because the consumer is allowed to create it
If it doesn't exist, a 404 is returned	If it doesn't exist, the resource is created
POST must be used for creation as we cannot know the URI in advance	PUT can be used for creation: upsert

Demo

Upserting with PUT

Partially Updating a Resource

HTTP PATCH is for partial updates

The request body of a patch request is described by RFC 6902 (JSON Patch) https://tools.ietf.org/html/rfc6902

PATCH requests should be sent with media type “application/json-patch+json”

```
[
    {
        "op": "replace",
        "path": "/title",
        "value": “new title"
    }
    ,
    {
        "op": “remove",
        "path": "/description"
    }
]
```

- array of operations
- “replace” operation
- “title” property gets value “new title”
- “remove” operation
- “description” property is removed (set to its default value)

JSON Patch Operations

Add

```
{"op": "add",
"path": "/a/b",
"value": "foo"}
```

Remove

```
{"op": "remove",
"path": "/a/b"}
```

Replace

```
{"op": "replace",
"path": "/a/b",
"value": "foo"}
```

JSON Patch Operations

Copy

```
{"op": "copy",
"from": "/a/b",
"path": "/a/c"}
```

Move

```
{"op": "move",
"from": "a/b",
"path": "/a/c"}
```

Test

```
{"op": "test",
"path": "/a/b",
"value": "foo"}
```

Demo

Partially updating a resource

Demo

Validating input when updating a resource with PATCH

Demo

Returning ValidationProblems from controller actions

Demo

Upserting with PATCH

Demo

Validating input when upserting with PATCH

Summary

PUT for full updates

- 200 Ok or 204 No content
- Not safe
- Idempotent

PATCH for partial updates

- JSON Patch standard
- 200 Ok or 204 No content
- Not safe
- Not idempotent

Summary

Upserting

- Possible when the consumer of the API is allowed to create the resource URI

Summary

Validation rules may be different between creating and updating a resource

When patching, validate the patched DTO