Partial Updates (PATCH) in Spring Boot

https://medium.com/@henrickkakutalua/partial-updates-patch-in-spring-boot-63ff35582250

Using JsonNullable to update only the fields you want

The Problem

Last year (2019), i was fortunate to work both as mobile and backend developer for Box Office Experience application, with a wonderful and talented team.

One of the annoying things i ended up facing in both roles, was dealing with endpoints to update resources in this application, either with PATCH or PUT HTTP methods. With the somewhat limited knowledge we had at the time about Spring Boot, we ended up creating update endpoints that required the client to always specify all the fields, even when not required.

For example, imagine that we have a set of endpoints used to manipulate a Person resource with the JSON representation below:

```
{
  "id": 1,
  "first_name": "Henrick",
  "last_name": "Kakutalua",
  "birthday": "2020-09-23T00:00:00+00:01",
  "bio": "Lorem ipsum dolor amet consectur",
  "image_url": "https://image/profile_photo.jpg"
}
```

Then suppose you only need to update the first_name and last_name attributes. The logical solution would be to only specify values for these two attributes. Right?

Well, we could do that if we evaluate in our server-side code which values are not null, and update the resource accordingly.

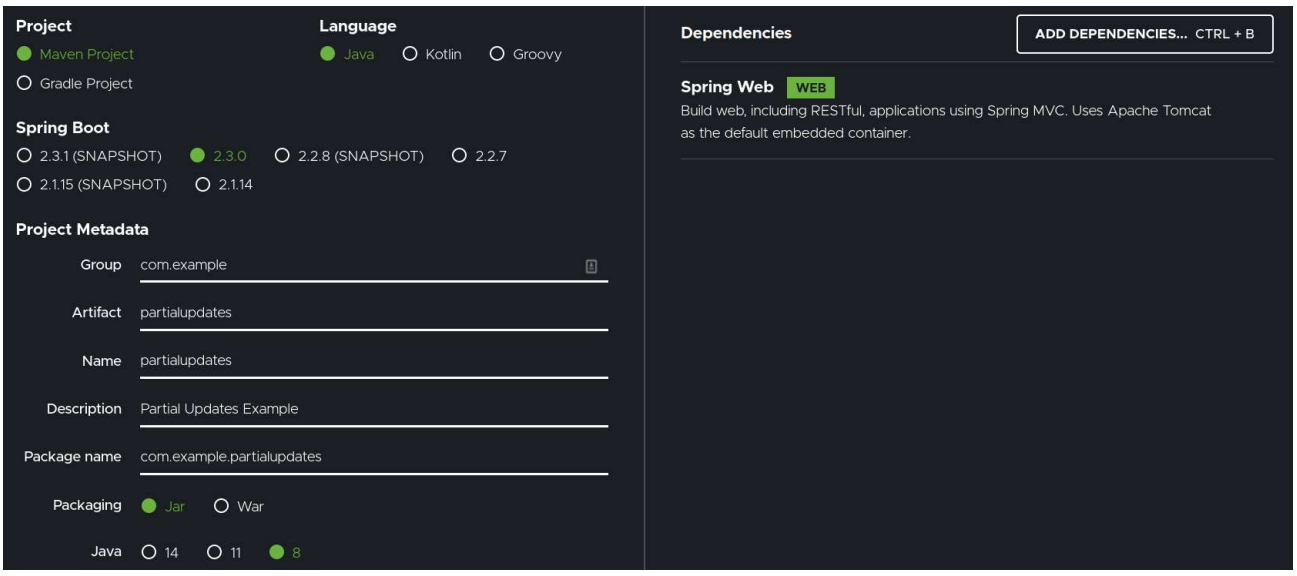
Let's create a simple Spring Boot project so we can start implementing this.

Take these ... as a sign to relax, to pause a little bit on the reading, OK?

Creating The Project

If you prefer to go right to the code, checkout theGithub repository here.

Go to Spring Initializer with this link, the link already contains the dependencies and settings needed for the project. It should be similar to the image below when you open it.



Spring Initializer

Click on Generate (Ctrl + Enter), to generate and download the project. Decompress the project and open in your favourite IDE.

You will notice that is quite a simple project, containing a PartialupdatesApplication annotated with @SpringBootApplication as expected.

Domain and Repository class for Person

Let's first create the Person domain class.

```
package com.example.partialupdates.domain;

import java.time.OffsetDateTime;
import java.util.Objects;
import java.util.Optional;

public class Person {
    private final long id;
    private String firstName;
    private String lastName;
    private OffsetDateTime birthday;
    private String bio;
    private String imageUrl;

    public Person(long id, String firstName, String lastName, OffsetDateTime birthday,
String bio, String imageUrl) {
        if (firstName == null) {
            throw new IllegalArgumentException("firstName can't be null");
        }
        if (lastName == null) {
            throw new IllegalArgumentException("lastName can't be null");
        }
        if (birthday == null) {
            throw new IllegalArgumentException("birthday can't be null");
        }

        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthday = birthday;
        this.bio = bio;
        this.imageUrl = imageUrl;
    }

    public long getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public Person setFirstName(String firstName) {
        if (firstName == null) {
            throw new IllegalArgumentException("firstName can't be null");
        }

        this.firstName = firstName;
        return this;
    }

    public String getLastName() {
        return lastName;
    }

    public Person setLastName(String lastName) {
        if (lastName == null) {
```

```
            throw new IllegalArgumentException("lastName can't be null");
        }

        this.lastName = lastName;
        return this;
    }

    public OffsetDateTime getBirthday() {
        return birthday;
    }

    public Person setBirthday(OffsetDateTime birthday) {
        if (birthday == null) {
            throw new IllegalArgumentException("birthday can't be null");
        }

        this.birthday = birthday;
        return this;
    }

    public Optional<String> getBio() {
        return Optional.ofNullable(bio);
    }

    public Person setBio(String bio) {
        this.bio = bio;
        return this;
    }

    public Optional<String> getImageUrl() {
        return Optional.ofNullable(imageUrl);
    }

    public Person setImageUrl(String imageUrl) {
        this.imageUrl = imageUrl;
        return this;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return id == person.id;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

Quite verbose, right? You can notice that we are enforcing some fields to be required (i.e. non-nullable) with preconditions in the constructor and setters. Preconditions are contracts that enforce how the state of an object ought to be before before being initialized. We're enforcing firstName, lastName and birthday to be non-nullable.

Let's create an in-memory repository for this domain class, PeopleRepository:

```
package com.example.partialupdates.repositories;

import com.example.partialupdates.domain.Person;
import org.springframework.stereotype.Repository;

import java.time.OffsetDateTime;
import java.util.*;
import java.util.function.UnaryOperator;
```

```
@Repository
public class PeopleRepository {
    private static final ArrayList<Person> people = new ArrayList<>();

    public PeopleRepository() {
        Person person1 = new Person(
                1, "Henrick", "Kakutalua", OffsetDateTime.now(),
                "Lorem ipsum", "https://image/profile_photo.jpg");
        Person person2 = new Person(
                2, "Alexandre", "Juca", OffsetDateTime.now(),
                "Lorem ipsum", "https://image/profile_photo.jpg");
        Person person3 = new Person(
                3, "Pedro", "Massango", OffsetDateTime.now(),
                "Lorem ipsum", "https://image/profile_photo.jpg");
        Person person4 = new Person(
                4, "Ilton", "Ingui", OffsetDateTime.now(),
                "Lorem ipsum", "https://image/profile_photo.jpg");
        Person person5 = new Person(
                5, "Aristoteles", "Lopes", OffsetDateTime.now(),
                "Lorem ipsum", "https://image/profile_photo.jpg");

        people.addAll(Arrays.asList(person1, person2, person3, person4, person5));
    }

    public List<Person> getAll() {
        return Collections.unmodifiableList(people);
    }

    public Optional<Person> getById(long id) {
        return people.stream()
                .filter(p -> p.getId() == id)
                .findFirst();
    }

    public void save(Person person) {
        if (person == null) {
            throw new IllegalArgumentException("person can't be null");
        }

        if (!people.contains(person)) {
            throw new RuntimeException(String.format("Person with id '%s' does not
exists", person.getId()));
        }

        int indexOfPerson = people.indexOf(person);
        people.remove(person);
        people.add(indexOfPerson, person);
    }
}
```

There is already preloaded data, so you don't have to worry about populating the repository with a POST endpoint.

Partial Updates with Null Evaluation

The first and ingeniously obvious strategy to partially update a resource would be to evaluate which request body fields are not null, and update the resource with these modified fields.

We'll create a controller containing two endpoints, one to retrieve all people and the second to update a specific person. The controller should be quite straightforward with the data being stored in-memory.

Before we create our controller, we'll need two data transfer objects that contains data that is retrieved when getting people and data that is updated when updating people, respectively.

```
package com.example.partialupdates.controllers.dtos;
```

```
import java.time.OffsetDateTime;

public class PersonReadDto {
    private final long id;
    private final String firstName;
    private final String lastName;
    private final OffsetDateTime birthday;
    private final String bio;
    private final String imageUrl;

    public PersonReadDto(long id, String firstName, String lastName, OffsetDateTime
birthday, String bio, String imageUrl) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthday = birthday;
        this.bio = bio;
        this.imageUrl = imageUrl;
    }

    public long getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public OffsetDateTime getBirthday() {
        return birthday;
    }

    public String getBio() {
        return bio;
    }

    public String getImageUrl() {
        return imageUrl;
    }
}
```

```
package com.example.partialupdates.controllers.dtos;

import java.time.OffsetDateTime;

public class PersonUpdateDto {
    private String firstName;
    private String lastName;
    private OffsetDateTime birthday;
    private String bio;
    private String imageUrl;

    protected PersonUpdateDto() { }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
```

```
    public OffsetDateTime getBirthday() {
        return birthday;
    }

    public String getBio() {
        return bio;
    }

    public String getImageUrl() {
        return imageUrl;
    }
}
```

Ok, now, finally, let's create a PeopleController containing the endpoints to retrieve and update people. As you can see below, our update endpoint evaluates if a DTO property is not null before assigning it to the entity.

```
package com.example.partialupdates.controllers;

import com.example.partialupdates.controllers.dtos.PersonReadDto;
import com.example.partialupdates.controllers.dtos.PersonUpdateDto;
import com.example.partialupdates.domain.Person;
import com.example.partialupdates.repositories.PeopleRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/people")
public class PeopleController {
    private final PeopleRepository peopleRepository;

    public PeopleController(PeopleRepository peopleRepository) {
        if (peopleRepository == null)
            throw new IllegalArgumentException("peopleRepository can't be null");

        this.peopleRepository = peopleRepository;
    }

    @GetMapping
    public ResponseEntity<List<PersonReadDto>> getAllPeople() {
        List<Person> people = peopleRepository.getAll();
        List<PersonReadDto> personReadDtos = people.stream()
                .map(p -> new PersonReadDto(
                        p.getId(),
                        p.getFirstName(),
                        p.getLastName(),
                        p.getBirthday(),
                        p.getBio().orElse(null),
                        p.getImageUrl().orElse(null)))
                .collect(Collectors.toList());

        return ResponseEntity.ok(personReadDtos);
    }

    @PatchMapping("{id}")
    public ResponseEntity<Void> updatePerson(@Valid @PathVariable("id") long id,
                                             @RequestBody PersonUpdateDto personUpdateDto)
{
        Optional<Person> personOptional = peopleRepository.getById(id);
        if (!personOptional.isPresent()) {
            return ResponseEntity.notFound().build();
        }
```

```
        Person person = personOptional.get();
        if (personUpdateDto.getFirstName() != null) {
            person.setFirstName(personUpdateDto.getFirstName());
        }
        if (personUpdateDto.getLastName() != null) {
            person.setLastName(personUpdateDto.getLastName());
        }
        if (personUpdateDto.getBirthday() != null) {
            person.setBirthday(personUpdateDto.getBirthday());
        }
        if (personUpdateDto.getBio() != null) {
            person.setBio(personUpdateDto.getBio());
        }
        if (personUpdateDto.getImageUrl() != null) {
            person.setImageUrl(personUpdateDto.getImageUrl());
        }

        peopleRepository.save(person);

        return ResponseEntity.noContent().build();
    }
}
```

Let's run the project and test our partial update mechanism.

```
mvn spring-boot:run
```

Let's look at all people registered in our application

```
curl --location --request GET 'http://localhost:8080/api/people' | json_pp
```

```
[
    {
        "id": 1,
        "firstName": "Henrick",
        "lastName": "Kakutalua",
        "birthday": "2020-06-02T22:10:16.016+01:00",
        "bio": "Lorem ipsum",
        "imageUrl": "https://image/profile_photo.jpg"
    },
    {
        "id": 2,
        "firstName": "Alexandre",
        "lastName": "Juca",
        "birthday": "2020-06-02T22:10:16.016+01:00",
        "bio": "Lorem ipsum",
        "imageUrl": "https://image/profile_photo.jpg"
    },
...
```

We have Henrick, Alexandre, among others. Change Henrick's first name, last name and birthday.

```
curl --location --request PATCH 'http://localhost:8080/api/people/1' \
--header 'Content-Type: application/json' \
--data-raw '{
"firstName": "Clark",
"lastName": "Kent",
"birthday": "1956-09-23T00:00:00+01:00"
}'
```

If you retrieve again all people, you'll notice that the changes were indeed made.

Now, let's try to update the remaining properties of Superman, bio and imageUrl.

```
curl --location --request PATCH 'http://localhost:8080/api/people/1' \
--header 'Content-Type: application/json' \
--data-raw '{
  "bio": "Superman was born at Krypton",
  "imageUrl": "https://new.image.url"
}'
```

NICE!! If you try to get all people again, you'll notice that Henrick is no more, Superman is born!

But if you remember, we defined the Person entity as having non-nullable properties such as firstName, lastName and birthday, as well as nullable ones such as bio and imageUrl.

What if we need to set bio and imageUrl to null? Doing a PATCH request like the one below won't update the properties as we wish, as we only update an entity property when the input is not null :-(.

```
curl --location --request PATCH 'http://localhost:8080/api/people/1' \
--header 'Content-Type: application/json' \
--data-raw '{
  "bio": null,
  "birthday": null
}'
```

Well, you can think: "we can remove null evaluation for properties that can be nullable". And that would solve(-ish) the problem.

```
Person person = personOptional.get();
if (personUpdateDto.getFirstName() != null) {
    person.setFirstName(personUpdateDto.getFirstName());
}
if (personUpdateDto.getLastName() != null) {
    person.setLastName(personUpdateDto.getLastName());
}
if (personUpdateDto.getBirthday() != null) {
    person.setBirthday(personUpdateDto.getBirthday());
}
person.setBio(personUpdateDto.getBio());
person.setImageUrl(personUpdateDto.getImageUrl());

peopleRepository.save(person);

return ResponseEntity.noContent().build();
```

If you update bio and imageUrl to null and retrieve again all people, you'll notice that our change was indeed successful. But unfortunately we introduced a new problem: if we omit bio and birthday from the request body (which is reasonable if you don't want to update them) they will be changed to NULL nonetheless.

That's why this first strategy —null evaluation — is inadequate for our purpose. We need to omit properties we don't want to update, and if we do so, their input will be null in the DTO, and the entity corresponding fields set to null unintendedly.

Partial Updates with JsonNullable

The second strategy that we will try, which is more sophisticate, is by using the wrapper class JsonNullable from the jackson-databind-nullable library.

JsonNullable will allow our application to handle DTO fields in three possible states:

- **non-null value:** a value like "abcd", 76 and []
- **null value:** well, a null value.
- **undefined value:** an omitted value.

Add jackson-databind-nullable as a dependency to pom.xml

```
<dependency>
   <groupId>org.openapitools</groupId>
   <artifactId>jackson-databind-nullable</artifactId>
   <version>0.2.1</version>
</dependency>
```

Create a JacksonConfiguration class, we'll setup ObjectMapper there, so Jackson can correctly serialize and deserialize classes containing JsonNullable wrapped fields.

```
package com.example.partialupdates.config;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.openapitools.jackson.nullable.JsonNullableModule;
import org.springframework.context.annotation.Configuration;

import javax.annotation.PostConstruct;

@Configuration
public class JacksonConfiguration {
    private final ObjectMapper objectMapper;

    public JacksonConfiguration(ObjectMapper objectMapper) {
        this.objectMapper = objectMapper;
    }

    @PostConstruct
    ObjectMapper jacksonObjectMapper() {
        objectMapper.registerModule(new JsonNullableModule());
        return objectMapper;
    }
}
```

Note that we're using @PostConstruct annotation so we can configure Jackson's ObjectMapper after Spring Boot, because we don't want to lose all useful setup made previously by Spring Boot, such as Date and time serialization, for example.

Now, modify the PersonUpdateDto fields to be wrapped with JsonNullable class:

```
package com.example.partialupdates.controllers.dtos;

import org.openapitools.jackson.nullable.JsonNullable;

import javax.validation.constraints.NotNull;
import java.time.OffsetDateTime;

@SuppressWarnings("FieldMayBeFinal")
public class PersonUpdateDto {
    @NotNull
    private JsonNullable<String> firstName = JsonNullable.undefined();
    @NotNull
    private JsonNullable<String> lastName = JsonNullable.undefined();
    @NotNull
    private JsonNullable<OffsetDateTime> birthday = JsonNullable.undefined();
    private JsonNullable<String> bio = JsonNullable.undefined();
    private JsonNullable<String> imageUrl = JsonNullable.undefined();

    protected PersonUpdateDto() { }

    public JsonNullable<String> getFirstName() {
        return firstName;
    }

    public JsonNullable<String> getLastName() {
        return lastName;
    }

    public JsonNullable<OffsetDateTime> getBirthday() {
```

```
        return birthday;
    }

    public JsonNullable<String> getBio() {
        return bio;
    }

    public JsonNullable<String> getImageUrl() {
        return imageUrl;
    }
}
```

Note that we added the @NotNull annotation to firstName, lastName and birthday fields. This will make the application return a 400 error if their values are not omitted and explicitly set to null.

Update PeopleController to evaluate if the values of the fields above are not omitted and update the entity accordingly:

```
@RestController
@RequestMapping("/api/people")
public class PeopleController {
    ...

    @PatchMapping("{id}")
    public ResponseEntity<Void> updatePerson(@PathVariable("id") long id,
                                             @Valid @RequestBody PersonUpdateDto
personUpdateDto) {
        Optional<Person> personOptional = peopleRepository.getById(id);
        if (!personOptional.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        Person person = personOptional.get();
        if (personUpdateDto.getFirstName().isPresent()) {
            person.setFirstName(personUpdateDto.getFirstName().get());
        }
        if (personUpdateDto.getLastName().isPresent()) {
            person.setLastName(personUpdateDto.getLastName().get());
        }
        if (personUpdateDto.getBirthday().isPresent()) {
            person.setBirthday(personUpdateDto.getBirthday().get());
        }
        if (personUpdateDto.getBio().isPresent()) {
            person.setBio(personUpdateDto.getBio().get());
        }
        if (personUpdateDto.getImageUrl().isPresent()) {
            person.setImageUrl(personUpdateDto.getImageUrl().get());
        }

        peopleRepository.save(person);

        return ResponseEntity.noContent().build();
    }
}
```

JsonNullable's method isPresent evaluates if the wrapped value exists, this value can be a non-null or null value. If the value is undefined/omitted, isPresent will return false.

If you make a request omitting the nullable fields, the correspondent properties in the entity won't be changed to null.

```
curl --location --request PATCH 'http://localhost:8080/api/people/1' \
--header 'Content-Type: application/json' \
--data-raw '{
  "firstName": "Clark",
  "lastName": "Kent"
}'
```

So, that solves completely the problem we had with the first strategy, when a field is omitted.

Conclusion and Bonus

As we saw, we can use jackson-databind-nullable library to effectively do partial updates. But the code we used to evaluate if the fields are present are not the best example of readability.

Let's improve that by making a helper class JsonNullableUtils that will improve our code readability:

```
package com.example.partialupdates.utils;

import org.openapitools.jackson.nullable.JsonNullable;

import java.util.function.Consumer;

public final class JsonNullableUtils {
    private JsonNullableUtils() {}

    public static <T> void changeIfPresent(JsonNullable<T> nullable, Consumer<T> consumer)
{
        if (nullable.isPresent()) {
            consumer.accept(nullable.get());
        }
    }
}
```

The code is very much straightforward, the changeIfPresent method takes a JsonNullable instance, evaluates if there's a value within it and passes the value to the Consumer<T> instance.

Consumer<T> is just a reference to a method that contains one parameter of type T and returns nothing. We can pass either method references or lambdas as an argument to a parameter of type Consumer<T>. See the updated code below for PeopleController:

```
package com.example.partialupdates.utils;

import org.openapitools.jackson.nullable.JsonNullable;

import java.util.function.Consumer;

public final class JsonNullableUtils {
    private JsonNullableUtils() {}

    public static <T> void changeIfPresent(JsonNullable<T> nullable, Consumer<T> consumer)
{
        if (nullable.isPresent()) {
            consumer.accept(nullable.get());
        }
    }
}
```

Nice!! That's way better. Hope you have enjoyed this article.

Please don't forget to share and clap as many time as you want to help this article reach out more people.

References

https://github.com/OpenAPITools/jackson-databind-nullable