# REST

> **Parameterized Type Reference**
>
> when passing a **List** *(OR any type that takes in a parameter)* in the request body always **DO NOT USE TYPE.class** use:
> `.bodyToMono(new ParametertizedTypeReference<List<Type>> () {})`

## Server side

### full example

```
@RestController
@RequestMapping(path="/persons", consumes = {MediaType.APPLICATION_JSON_VALUE}, produces = {MediaType.APPLICATION_JSON_VALUE})
public class PersonResource {

    private final PersonService personService;

    public PersonResource(PersonService personService) {
        this.personService = personService;
    }

    // TODO Part 1: Implement the specified endpoints here
    @GetMapping
    public ResponseEntity<List<Person>> getAllPersons(@RequestParam(name = "AD", defaultValue = "Descending") String AD, @RequestParam( name = "FD",  defaultValue = "ID") String FD){
        PersonSortingOptions sortOp = new PersonSortingOptions( );
        switch(AD){

            case "Descending" -> sortOp.setSortingOrder(PersonSortingOptions.SortingOrder.DESCENDING);
            default -> sortOp.setSortingOrder(PersonSortingOptions.SortingOrder.ASCENDING);
        }
        switch(FD){
            case "ID" -> sortOp.setSortField(PersonSortingOptions.SortField.ID);
            case "FN" -> sortOp.setSortField(PersonSortingOptions.SortField.FIRST_NAME);
            case "LN" -> sortOp.setSortField(PersonSortingOptions.SortField.LAST_NAME);
            case "BD" -> sortOp.setSortField(PersonSortingOptions.SortField.BIRTHDAY);
            default -> sortOp.setSortField(PersonSortingOptions.SortField.ID);
        }
        return ResponseEntity.ok(personService.getAllPersons(sortOp));
    }
    @PostMapping
    public ResponseEntity<Person> createPerson(@RequestBody Person person){
        if(person.getId()==null) {
            return ResponseEntity.ok(personService.savePerson(person));
        }
        return ResponseEntity.badRequest().build();
    }
    @PutMapping("/{id}")
    public ResponseEntity<Person> updatePerson(@RequestBody Person person, @PathVariable UUID id){
        if(person.getId().equals(id)){
            return ResponseEntity.ok(personService.savePerson(person));
        }
        return ResponseEntity.badRequest().build();
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deletePerson(@PathVariable UUID id){
        if(id!= null){
            personService.deletePerson(id);
        }
        return ResponseEntity.noContent().build();
    }

}
```

## Response

#response

```
return ResponseEntity.ok(ObjectToReturn);

return ResponseEntity.badRequest.build();

//For Deletes
return ResponseEntity.noContent().build();
```

## Request

#request

### Path Variable

#### server side

```
@GetMapping("/onTopOfBaseURI/{parameter}")
public ResponseEntity<ResponseType> function(@PathVariable Type parameter){}
```

#### client side

```
RestTemplate restTemplate = new RestTemplate();
String url = "http://localhost:8080/users/{id}";
Long userId = 42L;

ResponseEntity<String> response = restTemplate.getForEntity(url, String.class, userId);
System.out.println(response.getBody());
```

or something simpler by concatenating

> **RestTemplate vs WebClient**
>
> `getForEntity() / getForObject()` is used with **RestTemplate**
> `.get()...retrieve.bodyToMono().subscribe(something -> {})` is used with **WebClient**
> **THE SUBSCRIBE IS OPTIONAL FOR A REACTIVE ACTION**

## Client side

### Full example

```
public class PersonController {
    private final WebClient webClient = WebClient.builder()
            .baseUrl("http://localhost:8080/")
                .defaultHeader(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
                .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
                .build();
    private final List<Person> persons = new ArrayList<>();

    public void addPerson(Person person, Consumer<List<Person>> personsConsumer) {
```

```
        // TODO Part 2: Make an http post request to the server
        webClient.post().uri("persons").bodyValue(person).retrieve().bodyToMono(Person.class).subscribe(newPerson ->{
            persons.add(newPerson);
            personsConsumer.accept(persons);
        });
    }

    public void updatePerson(Person person, Consumer<List<Person>> personsConsumer) {
        // TODO Part 2: Make an http put request to the server
        webClient.put().uri("persons/" + person.getId()).bodyValue(person).retrieve().bodyToMono(Person.class).subscribe(newPerson-> {
            persons.replaceAll(oldPerson -> oldPerson.getId().equals(newPerson.getId())? newPerson : oldPerson);
            personsConsumer.accept(persons);
        });
    }

    public void deletePerson(Person person, Consumer<List<Person>> personsConsumer) {
        // TODO Part 2: Make an http delete request to the server
        webClient.delete().uri("persons/"+ person.getId()).retrieve().toBodilessEntity().subscribe(v->{
           persons.remove(person);
           personsConsumer.accept(persons);
        });
    }

    public void getAllPersons(PersonSortingOptions sortingOptions, Consumer<List<Person>> personsConsumer) {
        // TODO Part 2: Make an https get request to the server
        String AD;
        String FD;
        switch(sortingOptions.getSortingOrder()){
            case ASCENDING -> AD = "Ascending";
            case DESCENDING -> AD = "Descending";
            default -> AD = "Descending";
        }
        switch(sortingOptions.getSortField()){
            case ID -> FD = "ID";
            case FIRST_NAME -> FD = "FN";
            case LAST_NAME -> FD = "LN";
            case BIRTHDAY -> FD = "BD";
            default -> FD = "ID";
        }
        webClient.get().uri(uriBuilder -> uriBuilder.path("persons").queryParam("AD",AD).queryParam("FD", FD).build()).retrieve().bodyToMono(new ParameterizedTypeReference<List<Person>>()
{}).onErrorStop().subscribe(newPersons -> {
            persons.clear();
            persons.addAll(newPersons);
            personsConsumer.accept(persons);
        });
    }
}
```

### better simple example

```
    public void updatePerson(Person person, Consumer<List<Person>> personsConsumer) {
        // TODO Part 2: Make an http put request to the server
        webClient.put().uri("persons/" + person.getId()).bodyValue(person).retrieve().bodyToMono(Person.class).subscribe(newPerson-> {
            persons.replaceAll(oldPerson -> oldPerson.getId().equals(newPerson.getId())? newPerson : oldPerson);
            personsConsumer.accept(persons);
        });
    }
```

### URI Builder

#uri

**DO NOT USE FOR SIMPLE PATH VARIABLES**

```
webClient.get().uri(uriBuilder -> uriBuilder.path("persons").queryParam("AD",AD).queryParam("FD", FD).build()).retrieve().bodyToMono(new ParameterizedTypeReference<List<Person>>()
{}).onErrorStop().subscribe(newPersons -> { persons.clear(); persons.addAll(newPersons); personsConsumer.accept(persons); });
```

## Object Mapper

#json

**Most of the time isn't used manually**

`return ResponseEntity.ok(user);` uses it by default to serialize
**OR** within @RequestBody

if needed though

```
ObjectMapper objectMapper = new ObjectMapper();
String jsonString = objectMapper.writeValueAsString(user);

User user = objectMapper.readValue(jsonString, User.class);
```