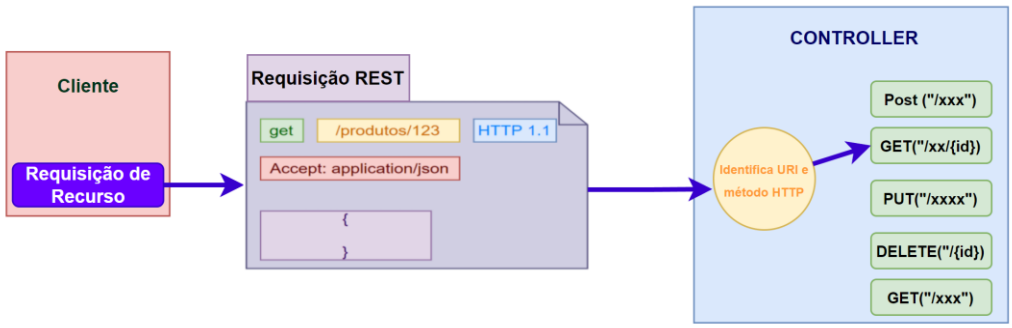
Rest Controllers

Os **controladores** são responsáveis por, olha só, **controlar o acesso a determinados recursos REST**, geralmente cada **Recurso** da aplicação tem seu **controlador.**

A ideia é que a **requisição REST** chegue ao **endpoint** de um **determinado controlador** que vai saber **ler essa requisição** e chamar a resposta de acordo com o **verbo HTTP e a URI** informadas pela requisição **(*outras coisas como cabeçalho e corpo da requisição podem ser necessárias):***



ResponseEntity

O **controller** deve devolver requisições que possuem **status de resposta *(é um padrão de REST)***, o Java tem um objeto para se lidar com essas respostas, esse objeto é o **ResponseEntity.**

Ele é uma espécie de **“envelope”** para se trabalhar com respostas **REST**, ele é um objeto que encapsula o **header**, o **status da reposta**, o **corpo da resposta** e etc... **Tudo que pode ser manipulável em uma resposta REST:**

```java
@PostMapping
@Transactional
public ResponseEntity<TopicoDto> cadastrarTopico(@RequestBody @Valid TopicoForm topicoForm,
                                                 UriComponentsBuilder uriBuilder){

    Topico topico = topicoForm.converterParaEntidade(cursoRepository);
    topicoRepository.save(topico);

    URI uri = uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId()).toUri();
    return ResponseEntity.created(uri).body(new TopicoDto(topico));
}
```

```java
@PutMapping("/{id}")
@Transactional
public ResponseEntity<TopicoDto> atualizarTopico(@PathVariable Long id, @RequestBody @Valid TopicoAtualizacaoForm form,
                                                 UriComponentsBuilder uriBuilder){

    Optional<Topico> topicoAtualizado = form.atualizar(id, topicoRepository);
    if(topicoAtualizado.isEmpty()){
        return ResponseEntity.notFound().build();
    }

    return ResponseEntity.ok(new TopicoDto(topicoAtualizado.get()));
}
```

Métodos **HTTP Post** quando bem sucedidos retornam o status **201 CREATED**, essa requisição tem como boa **prática devolver a URI do recurso que acabou de ser criado**, dá pra construir essa URI com ajuda de um objeto do **Spring.**

Esse Objeto é o **URIBuilder** ele é um objeto do próprio **Spring Web**, então **ele consegue ser injetado automaticamente**, bastando declara-lo como **parâmetro do método:**

```java
@PostMapping
@Transactional
public ResponseEntity<TopicoDto>  cadastrarTopico(@RequestBody @Valid TopicoForm topicoForm,
                                                  UriComponentsBuilder uriBuilder){

    Topico topico = topicoForm.converterParaEntidade(cursoRepository);
    topicoRepository.save(topico);

    URI uri = uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId()).toUri();
    return ResponseEntity.created(uri).body(new TopicoDto(topico));
}
```

Bean Validation

Pode ser que você queira g**arantir a "integridade"** de payload de uma **requisição REST**, por exemplo, o **payload não** pode ter ***campos nulos, ou maior que x tamanho e etc.....***

Para esse tipo de situação existe o **BeanValidation,** ele é do próprio Java, **mas o Spring** é integrado a ela através de uma **dependência**. A ideia é usar **anotações com restrições na classe** que **vai ser o Payload,** e no **método** de resposta do **controlador** que **vai fazer essa validação, avisar ao Spring:**

```java
public class TopicoForm {

    @NotNull @NotEmpty @Length(min = 5)
    private String titulo;

    @NotNull @NotEmpty @Length(min = 10)
    private String mensagem;

    @NotNull @NotEmpty
    private String nomeCurso;
```

```java
    }

    @PostMapping
    @Transactional
    public ResponseEntity<TopicoDto>  cadastrarTopico(@RequestBody @Valid TopicoForm topicoForm,
                                                      UriComponentsBuilder uriBuilder){

        Topico topico = topicoForm.converterParaEntidade(cursoRepository);
        topicoRepository.save(topico);

        URI uri = uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId()).toUri();
        return ResponseEntity.created(uri).body(new TopicoDto(topico));
    }
```

Controller Advice

O **Spring** sabe que tem que **validar o Bean** da requisição, ok, quando o **Bean vier inválido** ele vai lançar uma **Exceção** gigantesca como **resposta**. E se eu quiser **olhar meu controlador** pra **saber quando essa exceção vai ocorrer**, e sabendo quando **ela** ocorre eu posso pegar e tratar **ela**, eu posso fazer isso com **Controller advice.**

Basicamente eu vou ter **uma classe anotada** com **@RestControllerAdvice**, ela funciona como uma espécie de **interceptador** da **camada de conroller**, dentro **dela** eu vou avisar **qual exceção ela vai interceptar *(no caso a exceção do Bean validation),*** quando essa **exceção** for lançada pelo **controller** eu vou **captura-la** e **tratar** para devolver uma resposta melhor:

```java
@RestControllerAdvice
public class ErroDeValidacaoHandler {

    @Autowired
    private MessageSource messageSource;

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public List<ErroFormularioDto> handler(MethodArgumentNotValidException exception){

        List<ErroFormularioDto> erros = new ArrayList<>();
        List<FieldError> errosDeCampo = exception.getBindingResult().getFieldErrors();

        errosDeCampo.stream().forEach(erroCampo ->{
            String mensagem = messageSource.getMessage(erroCampo, LocaleContextHolder.getLocale());
            erros.add(new ErroFormularioDto(erroCampo.getField(), mensagem));
        });
        return erros;
    }
}
```