Validation

On Spring

Dependency

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Exemple on Entity

```
@Entity
public class UserApp {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;

   @NotBlank(message = "Name is mandatory")
   private String name;

   @NotBlank(message = "Email is mandatory")
   private String email;

}
```

on Controller

```
@RestController
@RequestMapping("/user")
public class UserController {

   @PostMapping
   ResponseEntity<String> addUser(@Valid @RequestBody UserApp user) {
      // persisting the user
      return ResponseEntity.ok("User is valid");
   }

}
```

On Fail

Quando o argumento (parâmetro) não "passa" na validação, é lançada uma excessão do tipo MethodArgumentNotValidException.

Então, precisamos escrever um tratamento para essa exception.

On Fail

Exemplo de tratamento de exception:

```
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(MethodArgumentNotValidException.class)
public Map<String, String> handleValidationExceptions(
  MethodArgumentNotValidException ex) {
    Map<String, String> errors = new HashMap<>();
    ex.getBindingResult().getAllErrors().forEach((error) -> {
        String fieldName = ((FieldError) error).getField();
        String errorMessage = error.getDefaultMessage();
        errors.put(fieldName, errorMessage);
    });
    return errors;
}
```

Outras anotações

@NotNull: uma CharSequence, Collection, Map, ou Array é valida se não for nula, mas pode estar vazia.

@NotEmpty: uma CharSequence, Collection, Map, ou Array é válida se não é nula e seu tamanho é maior que zero.

@NotBlank: uma String é valida se não é nula e seu tamanho, removidos os espaços em branco, é maior que zero.

@Email: uma String em formato válido de um e-mail.

@Min e @Max: um valor numerico é valido quando está acima ou abaixo de um certo valor informado.

@Pattern: uma String é valida quando combina com um certo padrão.

Exemplo

```
class Input {

  @Min(1)
  @Max(10)
  private int numberBetweenOneAndTen;

  @Pattern(regexp = "^[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}$")
  private String ipAddress;

}
```

Validando tipos primitivos

É possível validar também tipos primitivos em um Controller. Neste caso a exception disparada por padrão é ConstraintViolationException com retorno padrão 500 (Internal Server Error). Por isso, podemos indicar o tipo de exception que queremos que seja disparada em caso de falha.

Exemplo

```
@RestController
@Validated
class ValidateParametersController {

 @GetMapping("/validatePathVariable/{id}")
 ResponseEntity<String> validatePathVariable( @PathVariable("id") @Min(5) int id) {
  return ResponseEntity.ok("valid");
 }
```

E no Service?

```
@Service
@Validated
class ValidatingService{

   void validateInput(@Valid Input input){
     // do something
   }

}
```