



# THE DEVELOPER'S CONFERENCE

Como modelar e desenvolver **{APIs}**  
para **/evoluir** sua plataforma

Guilherme de Cleva Farto  
P&D e Arquitetura TOTVS

# Agenda



- Objetivos
- Tecnologias adotadas
  - Spring Boot Framework
  - Springfox e Swagger UI
  - MongoDB
  - Spring Initializr
- Desenvolvimento prático
- Próximos passos

# Objetivos



- Explorar os conceitos de APIs por meio da modelagem e do desenvolvimento prático com Java e Spring Boot
- Documentar as APIs (e endpoints/métodos) com Swagger
- Experimentar recursos
  - Uso de Lombok
  - Segurança com filtro para *Basic Authentication* e *Header*
  - Spring Data com *MongoRepository* e *Query*
  - Exceções customizadas e *HTTP Status*
- Próximos passos em APIs

# Tecnologias adotadas



- Java 8
- Apache Maven
- Lombok
- Spring Boot Framework
- Springfox e Swagger UI
- MongoDB

# Spring Boot Framework



## ➤ Spring Boot

- Framework modular para facilitar o desenvolvimento em Java
  - Inversão de controle e Injeção de dependências
  - Auto configuration

## ➤ Spring Ecosystem

- Spring WebServices, Spring Data, Spring Cloud, Spring Security, Spring AMQP, Spring Batch, Spring LDAP, Spring REST Docs, ...

# Spring Boot Framework



## ➤ Objetivos

- Otimizar experiência de início (rápido) de um projeto/arquitetura
- Minimizar configurações por XML (uso de [anotações](#) e [.properties](#))
- Fornecer requisitos não-funcionais previamente configurados
  - Métricas, segurança, *health check* de servidor, ...
- Possibilitar modo *standalone* e/ou servidor externo (Tomcat)
  - [Embedded Web Servers](#)
  - [Deployable War File](#)

# Springfox e Swagger UI



- Importância da documentação de APIs?
- Desafios
  - Evolução constante
  - Versionamento
  - Developer Experience (DX)
  - Para quem é a API?

# Springfox e Swagger UI



- O que seria interessante para nossas plataformas?
  - Torna possível a **geração de documentações de APIs por meio de metadados e anotações** no próprio código
  - Permite que usuários **executem e testem** as APIs por meio de interface de usuário padronizada



# Springfox e Swagger UI



## ➤ Swagger

- Plataforma agnóstica para documentação de APIs
- Expõe uma interface Web para execução e testes das APIs documentadas ([Swagger UI](#))
- Alternativas: Apiary, RAML, API Blueprint, ...

## ➤ Springfox

- Permite a injeção de metadados em classes (e.g., VO e Controllers) para gerar documentações de APIs REST
- Garante consistência entre código (dev), APIs (REST) e docs
- Renderiza os metadados (e anotações) em uma sintaxe JSON

# MongoDB



- Banco de dados NoSQL open source para dados não estruturados baseado em documentos (JSON)
- Utilizado para exemplificar o uso de **MongoRepository** e de **@Query** para consultas
- Inicialização da base de dados
  - `mongod --dbpath=C:/tdc_poa_2018/db`
  - `mongo`
- <https://docs.mongodb.com/>

# Spring Initializr



- Provê uma interface Web para geração de projetos a partir de templates (profiles) com dependências do Spring Boot
- Otimiza e padroniza a criação de novos projetos com Spring Boot e subprojetos (Web, Security, Data, JPA, ...)
- <https://start.spring.io/>

## ➤ Tutorial

- Geração do projeto com Spring Initializr
- Configuração de “pom.xml” para Springfox
- Configuração de “application.properties”
- Desenvolvimento de “HelloWorldController”

# 1. Uso do Spring Initializr



Spring Initializr x +

← → ↻ https://start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.1.0

### Project Metadata

Artifact coordinates

Group

br.com.tdc

Artifact

workshop-apis

### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web × DevTools × MongoDB × Lombok ×

Generate Project alt + ↵

## 2. Dependências no “pom.xml”



THE  
DEVELOPER'S  
CONFERENCE

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.8.0</version>
  <scope>compile</scope>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.8.0</version>
  <scope>compile</scope>
</dependency>
```

### 3. Configuração de “`application.properties`”



THE  
DEVELOPER'S  
CONFERENCE

```
# Server config
server.port=9090
server.servlet.context-path=/tdc

# MongoDB
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=tdc
```

## 4. Criação de “HelloWorldController”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.controllers;

@Controller
@RequestMapping("/api/hello")
public class HelloWorldController {

    @RequestMapping(
        method = RequestMethod.GET,
        produces = MediaType.TEXT_PLAIN_VALUE)
    @ResponseBody
    public String hello() {
        return "Hello, TDC";
    }
}
```



## 4. Criação de “HelloWorldController”

### \* Uso de @PathVariable



THE  
DEVELOPER'S  
CONFERENCE

```
@RequestMapping(  
    value = "/{name}",  
    method = RequestMethod.GET,  
    produces = MediaType.TEXT_PLAIN_VALUE)  
@ResponseBody  
public String getHelloPath(@PathVariable("name") String name) {  
    return "Hello, " + name + "!";  
}
```

## 4. Criação de “HelloWorldController”

### \* Uso de @RequestParam



THE  
DEVELOPER'S  
CONFERENCE

```
@RequestMapping(  
    value = "/withParam",  
    method = RequestMethod.GET,  
    produces = MediaType.TEXT_PLAIN_VALUE)  
@ResponseBody  
public String getHelloParam(@RequestParam("name") String name) {  
    return "Hello, " + name + "!";  
}
```

## 4. Criação de “HelloWorldController”

### \* Uso de @RequestHeader



THE  
DEVELOPER'S  
CONFERENCE

```
@RequestMapping(  
    value = "/withHeader",  
    method = RequestMethod.GET,  
    produces = MediaType.TEXT_PLAIN_VALUE)  
@ResponseBody  
public String getHelloHeader(@RequestHeader("name") String name) {  
    return "Hello, " + name + "!";  
}
```

## ➤ Testes de requisição

- GET <http://localhost:9090/tdc/api/hello>
- GET <http://localhost:9090/tdc/api/hello/Guilherme>
- GET <http://localhost:9090/tdc/api/hello/withParam?name=Guilherme>
- GET <http://localhost:9090/tdc/api/hello/withHeader>
  - (+ header “name” = “Guilherme”)

# ➤ Configuração Swagger

## 5. Criação de “SwaggerConfig”



```
package br.com.tdc.workshopapis.swagger;

@Configuration
@EnableSwagger2
public class SwaggerConfig {

    public static final Contact DEFAULT_CONTACT = new Contact("TDC Porto Alegre 2018",
        "http://thedevelopersconference.com.br", "guilherme.farto@gmail.com");

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(this.getApiInfo())
            .useDefaultResponseMessages(false)
            .select()
            .apis(Predicates.and(
                RequestHandlerSelectors.basePackage("br.com.tdc.workshopapis.controllers")))
            .build();
    }
}
```

## 5. Criação de “SwaggerConfig”



```
private ApiInfo getApiInfo() {  
    return new ApiInfoBuilder()  
        .title("TDC - Workshop APIs")  
        .description("Workshop de APIs no TDC Porto Alegre 2018")  
        .version("1.0.0")  
        .contact(DEFAULT_CONTACT).build();  
}  
  
@Bean  
public UiConfiguration uiConfig() {  
    return UiConfigurationBuilder.builder().filter(true)  
        .docExpansion(DocExpansion.LIST).defaultModelRendering(ModelRendering.EXAMPLE)  
        .deepLinking(true)  
        .defaultModelsExpandDepth(1).defaultModelExpandDepth(1)  
        .tagsSorter(TagsSorter.ALPHA).operationsSorter(OperationsSorter.ALPHA)  
        .displayRequestDuration(true)  
        .showExtensions(true).build();  
}  
}
```



THE  
DEVELOPER'S  
CONFERENCE

## ➤ Acessar documentação gerada

- <http://localhost:9090/tdc/v2/api-docs>
- <http://localhost:9090/tdc/swagger-ui.html>



# TDC - Workshop APIs 1.0.0

[ Base URL: localhost:9090/tdc ]

<http://localhost:9090/tdc/v2/api-docs>

Workshop de APIs no TDC Porto Alegre 2018

[TDC Porto Alegre 2018 - Website](#)

[Send email to TDC Porto Alegre 2018](#)

## hello-world-controller Hello World Controller

GET /api/hello hello

GET /api/hello/{name} getHelloPath

GET /api/hello/withHeader getHelloHeader

GET /api/hello/withParam getHelloParam

## 6. Criação de “HomeController”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.controllers;

@Controller
public class HomeController {

    @RequestMapping("/")
    public String home1() {
        return "redirect:/swagger-ui.html";
    }

    @RequestMapping("/swagger")
    public String home2() {
        return "redirect:/swagger-ui.html";
    }
}
```

## 7. Criação de annotation “**ApiTDC**”

\* Para filtro no **SwaggerConfig**



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface ApiTDC {

}
```

## 7. Criação de annotation “**ApiTDC**”

\* Para filtro no **SwaggerConfig**



THE  
DEVELOPER'S  
CONFERENCE

```
@ApiTDC
@Controller
@RequestMapping("/api/hello")
public class HelloWorldController {
```

SwaggerConfig.java

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(this.getApiInfo())
        .useDefaultResponseMessages(false)
        .select()
        .apis(Predicates.and(
            RequestHandlerSelectors.basePackage("br.com.tdc.workshopapis.controllers"),
            RequestHandlerSelectors.withClassAnnotation(ApiTDC.class)))
        .build();
}
```



## ➤ REST API para Usuário

- UserTypeEnum e UserVO
- UserRepository
- UserBO
- UserController
  - Exceções customizadas e HTTP Status

## 8. Criação de “UserTypeEnum” e “UserVO”



THE  
DEVELOPER'S  
CONFERENCE

UserTypeEnum.java

```
package br.com.tdc.workshopapis.enums;

public enum UserTypeEnum {

    ADMIN("admin"), USER("user");

    private final String value;

    private UserTypeEnum(String value) {
        this.value = value;
    }

    @JsonValue
    public String getValue() {
        return value;
    }
}
```

## 8. Criação de “UserTypeEnum” e “UserVO”



UserTypeEnum.java

```
public static UserTypeEnum findByValue(String value) {
    for (UserTypeEnum item : UserTypeEnum.values()) {
        if (value.equalsIgnoreCase(item.getValue())) {
            return item;
        }
    }

    return null;
}

public static class UserTypeEnumConverter extends PropertyEditorSupport {

    @Override
    public void setAsText(String value) throws IllegalArgumentException {
        setValue(UserTypeEnum.findByValue(value));
    }
}
}
```

## 8. Criação de “UserTypeEnum” e “UserVO”



UserVO.java

```
package br.com.tdc.workshopapis.models;

@ApiModel(value = "User", description = "Representa uma entidade de usuário")
@Document(collection = "users")
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
public class UserVO {

    @Id
    @Field("login")
    @ApiModelProperty(value = "Login do usuário", example = "user.test")
    private String login;
```



## 8. Criação de “UserTypeEnum” e “UserVO”



UserVO.java

```
@ApiModelProperty(value = "Nome do usuário", example = "User Test")
private String name;

@ApiModelProperty(value = "E-mail do usuário", example = "user.test@gmail.com")
private String mail;

@ApiModelProperty(value = "Data de criação do usuário")
private Date createdAt;

@ApiModelProperty(value = "Status do usuário", example = "true")
private boolean status;

@ApiModelProperty(value = "Tipo do usuário", example = "admin")
private UserTypeEnum userType;

}
```

## 9. Criação de “UserRepository” \* Com **MongoRepository**



```
package br.com.tdc.workshopapis.repositories;

@Repository
public interface UserRepository extends MongoRepository<UserV0, String> {

    UserV0 findByLogin(String login);

    @Query("{ 'userType' : ?0 }")
    List<UserV0> filter(UserTypeEnum userType);

    @Query(value = "{ 'login' : ?0 }", delete = true)
    Long deleteByLogin(String login);

}
```

## 10. Criação de “UserBO”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.business;

@Service
public class UserBO {

    @Autowired
    private UserRepository repository;

    public List<UserVO> getAll() {
        return repository.findAll();
    }

    public List<UserVO> filterUsers(UserTypeEnum userType) {
        if (userType == null) {
            return this.getAll();
        }

        return repository.filter(userType);
    }
}
```

## 10. Criação de “UserBO”

\* (continuação)



THE  
DEVELOPER'S  
CONFERENCE

```
public UserVO get(String login) {
    UserVO user = repository.findByLogin(login);

    return user;
}

public boolean hasUser(String login) {
    return this.get(login) != null;
}

public UserVO insert(UserVO user) {
    if (this.hasUser(user.getLogin())) {
        return null;
    }

    return repository.insert(user);
}
```

## 10. Criação de “UserBO”

\* (continuação)



THE  
DEVELOPER'S  
CONFERENCE

```
public UserVO update(UserVO user) {  
    if (!this.hasUser(user.getLogin())) {  
        return null;  
    }  
  
    return repository.save(user);  
}  
  
public Long delete(String login) {  
    if (!this.hasUser(login)) {  
        return null;  
    }  
  
    return repository.deleteByLogin(login);  
}  
  
public void deleteAll() {  
    repository.deleteAll();  
}  
}
```

## 11. Criação de exceções customizadas

- \* “**NotFoundException**”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.exceptions;

@ResponseStatus(code = HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public NotFoundException(String message) {
        super(message);
    }

    @Override
    public synchronized Throwable fillInStackTrace() {
        return this;
    }
}
```

## 11. Criação de exceções customizadas

### \* “EntityExistsException”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.exceptions;

@ResponseStatus(code = HttpStatus.CONFLICT)
public class EntityExistsException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public EntityExistsException(String message) {
        super(message);
    }

    @Override
    public synchronized Throwable fillInStackTrace() {
        return this;
    }
}
```

## 12. Criação de “UserController”



```
package br.com.tdc.workshopapis.controllers;

@Api(tags = { "users" })
@ApiTDC
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserBO bo;

    @ApiOperation("Recupera listagem de usuários")
    @ApiResponses({
        @ApiResponse(code = 200, message = "Usuários recuperados com sucesso"),
        @ApiResponse(code = 401, message = "Falha na autenticação") })
    @RequestMapping(method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
    @ResponseBody
    public List<UserVO> getUsers() {
        return bo.getAll();
    }
}
```



## 12. Criação de “UserController” \* (continuação)



```
@ApiOperation("Filtra listagem de usuários")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuários filtrados com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(value = "/filter",
    method = RequestMethod.GET,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public List<UserVO> filterUsers(
    @ApiParam(value = "Tipo de usuário", required = false)
    @RequestParam(name = "userType", required = false) UserTypeEnum userType) {
    return bo.filterUsers(userType);
}
```

## 12. Criação de “UserController” \* (continuação)



```
@ApiOperation("Recupera usuário pelo {login}")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuário recuperado com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(value =("/{login}",
    method = RequestMethod.GET,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public UserVO getUser(
    @ApiParam(value = "Login do usuário", required = true)
    @PathVariable(value = "login", required = true) String login) {
    UserVO user = bo.get(login);

    if (user == null) {
        throw new NotFoundException(MessageFormat.format("User {0} not found", login));
    }

    return user;
}
```

## 12. Criação de “UserController” \* (continuação)



THE  
DEVELOPER'S  
CONFERENCE

```
@ApiOperation("Insere usuário")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuário inserido com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(
    method = RequestMethod.POST,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public UserVO insertUser(@RequestBody UserVO user) {
    String login = user.getLogin();

    user = bo.insert(user);

    if (user == null) {
        throw new EntityExistsException(MessageFormat.format("User {0} already exists", login));
    }

    return user;
}
```

## 12. Criação de “UserController” \* (continuação)



```
@ApiOperation("Atualiza usuário pelo {login}")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuário atualizado com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(value =("/{login}", method = RequestMethod.PUT,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public UserVO updateUser(
    @ApiParam(value = "Login do usuário", required = true)
    @PathVariable(value = "login", required = true) String login,
    @RequestBody UserVO user) {
    user.setLogin(login);
    user = bo.update(user);

    if (user == null) {
        throw new NotFoundException(MessageFormat.format("User {0} not found", login));
    }

    return user;
}
```

## 12. Criação de “UserController” \* (continuação)



```
@ApiOperation("Remove usuário pelo {login}")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuário removido com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(value =("/{login}",
    method = RequestMethod.DELETE,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public void deleteUser(
    @ApiParam(value = "Login do usuário", required = true)
    @PathVariable(value = "login", required = true) String login) {
    Long result = bo.delete(login);

    if (result == null) {
        throw new NotFoundException(MessageFormat.format("User {0} not found", login));
    }
}
```

## 12. Criação de “UserController” \* (continuação)



THE  
DEVELOPER'S  
CONFERENCE

```
@ApiOperation("Remove todos os usuários")
@ApiResponses({
    @ApiResponse(code = 200, message = "Usuários removidos com sucesso"),
    @ApiResponse(code = 401, message = "Falha na autenticação") })
@RequestMapping(value = "/deleteAll",
    method = RequestMethod.DELETE,
    produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public void deleteAll() {
    bo.deleteAll();
}

@InitBinder
public void initBinder(final WebDataBinder webdataBinder) {
    webdataBinder.registerCustomEditor(UserTypeEnum.class, new UserTypeEnumConverter());
}
}
```

## ➤ Segurança com Filter

- “x-api-key” (header)
- Basic authentication

## 13. Criação de filtro para autenticação

### \* “**ApiFilter**”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.filters;

public class ApiFilter implements Filter {

    public static final String AUTHORIZATION = "authorization";
    public static final String X_API_KEY = "x-api-key";

    public static final String VALID_API_KEY = "tdc";
    public static final String VALID_BASIC_AUTH = "dGRjOnRkYw=="; // b64("tdc:tdc")

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        if (isValid(req)) {
            chain.doFilter(request, response);
        } else {
            cancel(res);
        }
    }
}
```



## 13. Criação de filtro para autenticação

### \* “**ApiFilter**”



THE  
DEVELOPER'S  
CONFERENCE

```
private boolean isValid(HttpServletRequest request) {  
    String apiKeyHeader = request.getHeader(ApiFilter.X_API_KEY);  
    String authorizationHeader = request.getHeader("Authorization");  
  
    if (apiKeyHeader != null && !"".equals(apiKeyHeader)) {  
        return ApiFilter.VALID_API_KEY.equals(apiKeyHeader);  
    } else if (authorizationHeader != null && !"".equals(authorizationHeader)) {  
        return ApiFilter.VALID_BASIC_AUTH.equals(authorizationHeader.replace("Basic ", ""));  
    }  
  
    return false;  
}  
  
private void cancel(HttpServletRequest response) throws IOException {  
    response.reset();  
    response.sendError(HttpStatus.UNAUTHORIZED, "The authorization is invalid");  
}  
}
```

## 13. Criação de filtro para autenticação

### \* “**FilterConfiguration**”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis;

@Configuration
public class FilterConfiguration {

    @Bean
    public FilterRegistrationBean<ApiFilter> apiFilter() {
        FilterRegistrationBean<ApiFilter> registrationBean = new FilterRegistrationBean<>();

        registrationBean.setFilter(new ApiFilter());
        registrationBean.addUrlPatterns("/api/users/*");

        return registrationBean;
    }
}
```

## 13. Criação de filtro para autenticação

### \* “SwaggerConfig”



```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(this.getApiInfo())
        .useDefaultResponseMessages(false)
        .securitySchemes(securitySchemes())
        .securityContexts(securityContext())
        // ...
}

private List<SecurityScheme> securitySchemes() {
    SecurityScheme securitySchemeApiKey =
        new ApiKey(ApiFilter.X_API_KEY, ApiFilter.X_API_KEY, "header");
    SecurityScheme securitySchemeBasic = new BasicAuth(ApiFilter.AUTHORIZATION);

    return Arrays.asList(securitySchemeApiKey, securitySchemeBasic);
}
```

## 13. Criação de filtro para autenticação

### \* “SwaggerConfig”



```
private List<SecurityContext> securityContext() {  
    return Arrays.asList(  
        SecurityContext.builder().securityReferences(defaultAuth())  
            .forPaths(PathSelectors.any()).build());  
}  
  
private List<SecurityReference> defaultAuth() {  
    AuthorizationScope[] authorizationScopes = new AuthorizationScope[0];  
  
    SecurityReference securityReferenceApiKey = SecurityReference.builder()  
        .reference(ApiFilter.X_API_KEY).scopes(authorizationScopes).build();  
    SecurityReference securityReferenceBasic = SecurityReference.builder()  
        .reference(ApiFilter.AUTHORIZATION).scopes(authorizationScopes).build();  
  
    return Arrays.asList(securityReferenceApiKey, securityReferenceBasic);  
}
```



THE  
DEVELOPER'S  
CONFERENCE

# ➤ Springfox Plugins

- Metadados e extensões
- [Documentação oficial](#)

## 14. Extensões para Swagger

### \* “DatabaseMetadata”



THE  
DEVELOPER'S  
CONFERENCE

```
package  
br.com.tdc.workshopapis.annotations;  
  
@Target(ElementType.FIELD)  
@Retention(RetentionPolicy.RUNTIME)  
public @interface DatabaseMetadata {  
  
    String column() default "";  
  
}
```

## 14. Extensões para Swagger

\* “UserVO”



THE  
DEVELOPER'S  
CONFERENCE

```
public class UserVO {  
  
    @Id  
    @Field("login")  
    @ApiModelProperty(value = "Login do usuário", example = "user.test")  
    @DatabaseMetadata(column = "LOGIN")  
    private String login;  
  
    @ApiModelProperty(value = "Nome do usuário", example = "User Test")  
    @DatabaseMetadata(column = "NAME")  
    private String name;  
  
    @ApiModelProperty(value = "E-mail do usuário", example = "user.test@gmail.com")  
    @DatabaseMetadata(column = "MAIL")  
    private String mail;  
}
```

## 14. Extensões para Swagger

### \* “CustomModelPropertyBuilderPlugin”



THE  
DEVELOPER'S  
CONFERENCE

```
package br.com.tdc.workshopapis.swagger.plugins;

@Component
@Order(SwaggerPluginSupport.SWAGGER_PLUGIN_ORDER + 1002)
public class CustomModelPropertyBuilderPlugin implements ModelPropertyBuilderPlugin {

    @Override
    public void apply(ModelPropertyContext context) {
        context.getBuilder().allowEmptyValue(null);

        if (context != null && context.getBeanPropertyDefinition().isPresent()) {
            Optional<DatabaseMetadata> annotation = Optional.fromNullable(context.getBeanPropertyDefinition()
                .get().getField().getAnnotation(DatabaseMetadata.class));

            if (annotation.isPresent()) {
                ObjectVendorExtension rootExtension = new ObjectVendorExtension("x-database-metadata");
                rootExtension.addProperty(new StringVendorExtension("column", annotation.get().column()));

                context.getBuilder().extensions(Collections.singletonList(rootExtension));
            }
        }
    }
}
```



## 14. Extensões para Swagger

### \* “CustomModelPropertyBuilderPlugin”



THE  
DEVELOPER'S  
CONFERENCE

```
@Override
public boolean supports(DocumentationType documentationType) {
    return true;
}
}
```

## 14. Extensões para Swagger

### \* “CustomModelPropertyBuilderPlugin”



THE  
DEVELOPER'S  
CONFERENCE

#### Models

##### User ▾ {

<i>description:</i>	<i>Representa uma entidade de usuário</i>
<code>createdAt</code>	<code>string(\$date-time)</code> Data de criação do usuário
<code>login</code>	<code>string</code> <i>example: user.test</i> <i>x-database-metadata: OrderedMap { "column": "LOGIN" }</i> Login do usuário
<code>mail</code>	<code>string</code> <i>example: user.test@gmail.com</i> <i>x-database-metadata: OrderedMap { "column": "MAIL" }</i> E-mail do usuário
<code>name</code>	<code>string</code> <i>example: User Test</i> <i>x-database-metadata: OrderedMap { "column": "NAME" }</i> Nome do usuário

> Dúvidas?

# Atividades



- Implementar uma API para eventos (workshop ou palestra)
  - Enumeração TipoEvento (workshop, palestra, ...)
  - Classe CoordenadorEventoVO
    - Login, nome e e-mail
  - Classe EventoVO
    - Código, título, descrição, tipo do evento, coordenador, data, quantidade de vagas, ...
  - Classe EventoRepository (persistência de dados)
  - Classe EventoBO (regras de negócio)
  - Classe EventoController (APIs REST)

## ➤ Próximos passos

- O que (continuar) estudando?

# Próximos passos



- API economy
- Microservices
- API rate limiting, tracking e testes
- Autenticação com JWT e/ou OAuth2
- Hypermedia as the Engine of Application State (HATEOAS)
- Open Data Protocol (OData)
- Subscrição de eventos
  - WebSockets, WebHooks, REST Hooks, Pub-Sub e Server Sent Events
- GraphQL

# Obrigado!



## Guilherme de Cleva Farto



[guilherme.farto@gmail.com](mailto:guilherme.farto@gmail.com)



<https://www.linkedin.com/in/guilherme-farto/>



<https://github.com/guilhermefarto>



<https://twitter.com/gclevea>



# THE DEVELOPER'S CONFERENCE