



DESAFIO DIAZERO

Cadastro Incidentes

Kevelly Braga Parente

API REST

**Construindo um API REST em que é possível
cadastrar incidentes utilizando Java e o
framework Spring.**

Kevelly Braga

INICIANDO NOSSA API REST...

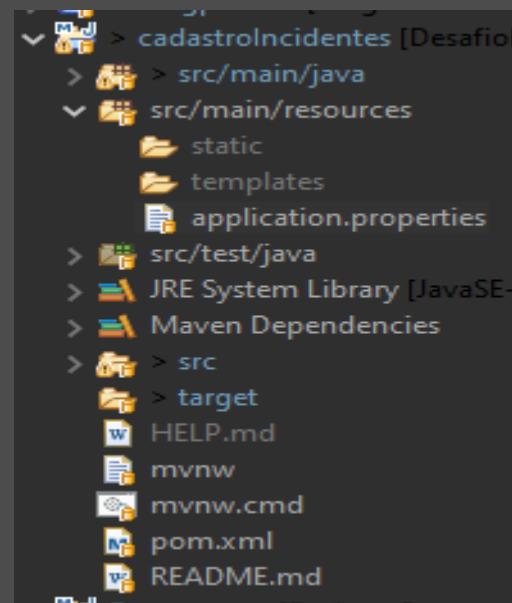
Começamos criando nosso projeto em Spring , para o mesmo, utilizei o Spring Initializr (<https://start.spring.io/>). Nele fiz as principais configurações, defini a versão do Java e adicionei as dependências necessárias para a criação do API REST.

Desse jeitinho:

The screenshot shows the Spring Initializr web application interface. On the left, the 'Project' section is set to 'Maven Project'. Under 'Language', 'Java' is selected. In the 'Spring Boot' section, '2.5.3' is chosen. The 'Project Metadata' section includes fields for Group (com.dlazero.CadastroIncident), Artifact (CadastroIncident), Name (CadastroIncident), Description (Demo project for Spring Boot), Package name (com.dlazero.CadastroIncident.CadastroIncident), and Packaging (Jar). Below these, Java version options 16, 11, and 8 are shown, with 11 selected. On the right, the 'Dependencies' section lists several optional add-ons: 'Spring Boot DevTools' (selected), 'Spring Web' (selected), 'Spring Security', 'Spring Data JPA', 'MySQL Driver', and 'Validation'. A button 'ADD DEPENDENCIES... CTRL + B' is located at the top right of the dependencies panel. At the bottom, there are buttons for 'GENERATE' (CTRL + ⌘), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'.

PRÓXIMOS PASSOS:

- Logo após criar as dependências, importei o projeto para a IDE ECLIPSE;
- Configurei e fiz a conexão com o banco de dados, MYSQL, a partir do *application.properties*.



```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.datasource.url = jdbc:mysql://localhost:3306/cadastro_Incidente?user=root&password=root&createDatabaseIfNotExist=true&serverTimezone=UTC&useSSL=false
3 spring.datasource.username=root
4 spring.datasource.password=root
5 spring.jpa.show-sql=true
6
```

Vale ressaltar que utilizei arquitetura de microsserviços para a criação do API REST e, por isso, trabalhei com camadas. O que representa uma aplicação que trabalha de forma autônoma e independente, e realizando, assim, processos específicos e objetivos.

A primeira camada criada foi a model, em que fiz a criação das entidades Incident e User e elas têm um relacionamento @OneToMany e @ManyToOne

Seguem as entidades criadas:

```
--  
21  @Entity  
22  @Table(name = "incidente")  
23  
24  public class Incident {  
25  
26●    @Id  
27    @GeneratedValue(strategy = GenerationType.IDENTITY)  
28    private long idIncident;  
29  
30●    @NotNull  
31    @Size(min = 3, max = 100)  
32    private String name;  
33  
34●    @NotNull  
35    @Size(min = 10, max = 500)  
36    private String description;  
37  
38●    @Temporal(TemporalType.TIMESTAMP)  
39    private Date createdAt = new java.sql.Date(System.currentTimeMillis());  
40  
41    private LocalDateTime updatedAt;  
42  
43    private LocalDateTime closedAt;  
44  
45●    @ManyToOne  
46    @JsonIgnoreProperties ("usuario")  
47    private User user;  
48  
49●    public long getIdIncident() {  
50        return idIncident;  
51    }  
52  
53
```

```
@Entity
@Table(name = "tb_usuario")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @NotNull
    @Size(min = 2, max = 100)
    private String name;
    @NotNull
    @Size(min = 5, max = 100)
    private String user;
    @NotNull
    @Size(min = 5, max = 100)
    private String password;

    @OneToMany(mappedBy = "usuario", cascade = CascadeType.REMOVE)
    @JsonIgnoreProperties("usuario")
    private List<Incident> incident;
```

Agora é a vez de mostrar os repositórios criados que são: `IncidentRepository` e `UserRepository`.

```
package org.diazero.cadastroIncidentes.repository;

import java.util.List;

@Repository
public interface IncidentRepository extends JpaRepository<Incident, Long> {
    public List<Incident> findAllByNameContainingIgnoreCase (String name);
    public List<Incident> findAllByDescriptionContainingIgnoreCase (String description);
}
```

```
package org.diazero.cadastroIncidentes.repository;

import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    public Optional<User> findByUser (String User);
}
```

Mostrando agora o Service, em que está toda nossa regra de negócio. Por exemplo, para fazer a manutenção e remoção de um *incidente* é preciso que o mesmo esteja cadastrado.

P.S Quando essas regras são feitas, gera-se a data do momento em questão.

```
13 @Service
14 public class IncidentService {
15
16     private @Autowired IncidentRepository repositoryI;
17
18     public Optional<Incident> registerIncident(Incident newIncident) {
19         return Optional.ofNullable(repositoryI.save(newIncident));
20     }
21
22     public Optional<Incident> maintenanceIncident(Incident incidentUpdated) {
23         LocalDateTime date = LocalDateTime.now();
24         Optional<Incident> existingIncident = repositoryI.findById(incidentUpdated.getIdIncident());
25         if (existingIncident.isPresent()) {
26             existingIncident.get().setName(incidentUpdated.getName());
27             existingIncident.get().setDescription(incidentUpdated.getDescription());
28             existingIncident.get().setUpdatedAt(date);
29             return Optional.ofNullable(repositoryI.save(existingIncident.get()));
30         } else {
31             return Optional.empty();
32         }
33     }
34
35     public Optional<Incident> closeIncident(Incident closeIncident) {
36         LocalDateTime date = LocalDateTime.now();
37         Optional<Incident> existingIncident = repositoryI.findById(closeIncident.getIdIncident());
38         if (existingIncident.isPresent()) {
39             existingIncident.get().setName(closeIncident.getName());
40             existingIncident.get().setDescription(closeIncident.getDescription());
41             existingIncident.get().setClosedAt(date);
42             return Optional.ofNullable(repositoryI.save(existingIncident.get()));
43         } else {
44             return Optional.empty();
45         }
46     }
```

Continuando na Service, agora a de usuário em que não é possível cadastrar mais de um usuário com o mesmo e-mail, por exemplo.

```
@Service
public class UserService {
    @Autowired
    private UserRepository repository;

    public Optional<User> UserRegister(User newUser) {
        BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
        String senhaCriptografada = encoder.encode(newUser.getPassword());
        newUser.setPassword(senhaCriptografada);
        return Optional.ofNullable(repository.save(newUser));
    }

    public Optional<UserLogin> login(Optional<UserLogin> loginUser) {
        BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
        Optional<User> presentUser = repository.findByUser(loginUser.get().getUsuario());
        if (presentUser.isPresent()) {
            if (encoder.matches(loginUser.get().getSenha(), presentUser.get().getPassword())) {
                String auth = loginUser.get().getUsuario() + ":" + loginUser.get().getSenha();
                byte[] encodedAuth = org.apache.tomcat.util.codec.binary.Base64
                    .encodeBase64(auth.getBytes(Charset.forName("US-ASCII")));
                String token = "Basic " + new String(encodedAuth);
                loginUser.get().setToken(token);
                loginUser.get().setId(presentUser.get().getId());
                loginUser.get().setNome(presentUser.get().getNome());
                loginUser.get().setSenha(presentUser.get().getPassword());

                return loginUser;
            }
        }
        return null;
    }
}
```

Agora é a hora do Controller, o lugar em que administrarmos e manipulamos as URL's. Utilizei o Controller para criação do método cadastrar incidentes, utilizando o POST; Para criação do método de manutenção dos incidentes, para isso utilizei o PUT, e para a criação do método de remoção dos incidentes, e para isso utilizei o DELETE.

```
24
25  @RestController
26  @RequestMapping ("/incident")
27  @CrossOrigin ("*")
28  public class IncidentController {
29
30    @Autowired
31    private IncidentRepository repository;
32
33    @Autowired
34    private IncidentService service;
35
36    @GetMapping
37    public ResponseEntity<List<Incident>> GetAll (){
38      return ResponseEntity.ok(repository.findAll());
39    }
40
41
42    @GetMapping ("/{idIncident}")
43    public ResponseEntity<Incident> GetById (@PathVariable long idIncident) {
44
45      return repository.findById(idIncident)
46        .map(resp -> ResponseEntity.ok(resp))
47        .orElse(ResponseEntity.notFound().build());
48    }
49    @GetMapping ("/{name}/{name}")
50    public ResponseEntity<List<Incident>> GetByName (@PathVariable String name) {
51      return ResponseEntity.ok(repository.findAllByNameContainingIgnoreCase(name));
52    }
53    @GetMapping ("/{description}/{description}")
54    public ResponseEntity<List<Incident>> GetByDescriprion (@PathVariable String description) {
55      return ResponseEntity.ok(repository.findAllByDescriptionContainingIgnoreCase(description));
56    }
... .
```

```
57
58    @PostMapping
59    public ResponseEntity<Optional<Incident>> registerIncident(@Valid @RequestBody Incident incident) {
60
61        return Optional.ofNullable(ResponseEntity.status(HttpStatus.CREATED).body(service.registerIncident(incident)))
62            .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
63    }
64
65    @PutMapping
66    ResponseEntity<Optional<Incident>> maintenanceIncident(@Valid @RequestBody Incident incidentUpdate) {
67        return Optional
68            .ofNullable(
69                ResponseEntity.status(HttpStatus.ACCEPTED).body(service.maintenanceIncident(incidentUpdate)))
70            .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
71    }
72
73    @PutMapping
74    ResponseEntity<Optional<Incident>> removeIncident(@Valid @RequestBody Incident closeIncident) {
75        return Optional
76            .ofNullable(ResponseEntity.status(HttpStatus.ACCEPTED).body(service.closeIncident(closeIncident)))
77            .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
78    }
79
80    @DeleteMapping("/{idIncident}")
81    public void removeIncidents(@PathVariable long idIncident) {
82        repository.deleteById(idIncident);
83    }
84
85}
86
```

O Controller de Usuários foi feito para fazer, também, métodos de: cadastro de novos usuários, login dos usuários, atualização dos dados do usuário e a opção de apagar o cadastro por meio de um ID.

```
@RestController
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserService userService;
    @Autowired
    private UserRepository repositoryU;

    @PostMapping("/login")
    public ResponseEntity<UserLogin> Autentication(@RequestBody Optional<UserLogin> user) {
        return userService.login(user).map(resp -> ResponseEntity.ok(resp))
            .orElse(ResponseEntity.status(HttpStatus.UNAUTHORIZED).build());
    }

    @PostMapping("/register")
    public ResponseEntity<Optional<User>> Post(@RequestBody User newUser) {
        return ResponseEntity.status(HttpStatus.CREATED).body(userService.UserRegister(newUser));
    }
}
```

```
@GetMapping
ResponseEntity<List<User>> getAllUsuario() {
    List<User> listaDeUsuarios = repositoryU.findAll();

    if (!listaDeUsuarios.isEmpty()) {
        return ResponseEntity.status(HttpStatus.OK).body(listaDeUsuarios);
    } else {
        return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
    }
}

@GetMapping("id/{id}")
ResponseEntity<User> getUsuarioById(@PathVariable Long id) {
    return repositoryU.findById(id).map(resp -> ResponseEntity.ok().body(resp))
        .orElse(ResponseEntity.notFound().build());
}

@PutMapping
public ResponseEntity<User> put(@RequestBody User usuario) {
    return ResponseEntity.status(HttpStatus.OK).body(repositoryU.save(usuario));
}

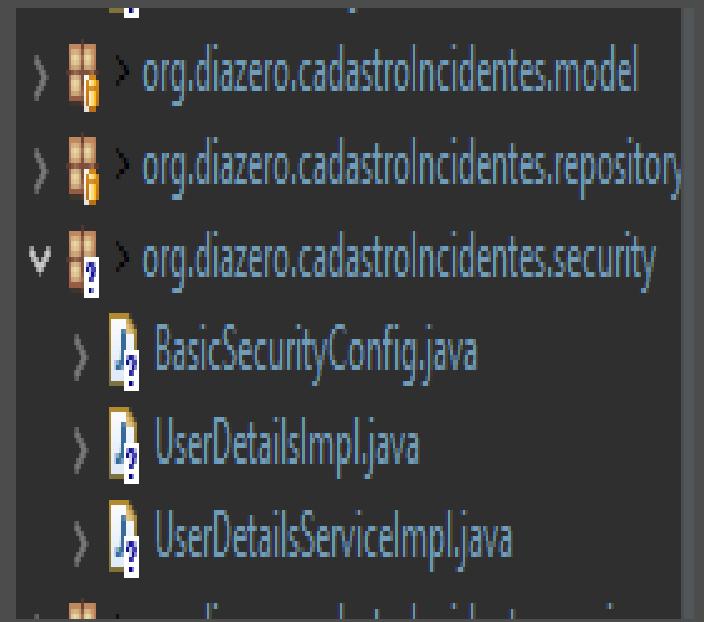
@DeleteMapping("/{id}")
public void delete(@PathVariable long id) {
    repositoryU.deleteById(id);
}
```

Para encerrar nossa aplicação, criei a camada de segurança implementada pelo Spring Security para autenticação do login e senha de nossos usuários a partir de um token de acesso.

```
import org.springframework.beans.factory.annotation.Autowired;

@EnableWebSecurity
public class BasicSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception{
        auth.userDetailsService(userDetailsService);
    }
    @Bean
    public PasswordEncoder passwordEncoder () {
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests() //restrinjir acesso a tudo
            .antMatchers("/usuarios/logar").permitAll()
            .antMatchers("/**").permitAll()
            .antMatchers("/usuarios/cadastrar").permitAll()
            .anyRequest().authenticated()
            .and().httpBasic() // configura a autenticação por http basic
            .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and().cors()
            .and().csrf().disable(); //adiciona suporte csrf = falsificação de solicitação entre sites
    }
}
```



```
import java.util.Collection;□

public class UserDetailsImpl implements UserDetails {
    private static final long serialVersionUID = 1L;

    private String userName;
    private String password;
    private List<GrantedAuthority> authorities;

    public UserDetailsImpl(User user) {
        this.userName = user.getName();
        this.password = user.getPassword();
    }

    public UserDetailsImpl() {
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // TODO Auto-generated method stub
        return authorities;
    }

    @Override
    public String getPassword() {
        // TODO Auto-generated method stub
        return password;
    }
}
```

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    @Autowired
    private User userRepository;
    @Override
    public UserDetails loadUserByUsername (String userName) throws UsernameNotFoundException {
        Optional<User> user = userRepository.findByName(userName);
        user.orElseThrow(() -> new UsernameNotFoundException( userName + " not found"));
        return user.map(UserDetailsImpl:: new).get ();
    }
}
```

**E assim finalizo o API.
Me coloco a disposição qualquer dúvida ou sugestão.
Abraços.**