DESAFIO DIAZERO

Cadastro Incidentes

Kevelly Parente

API REST

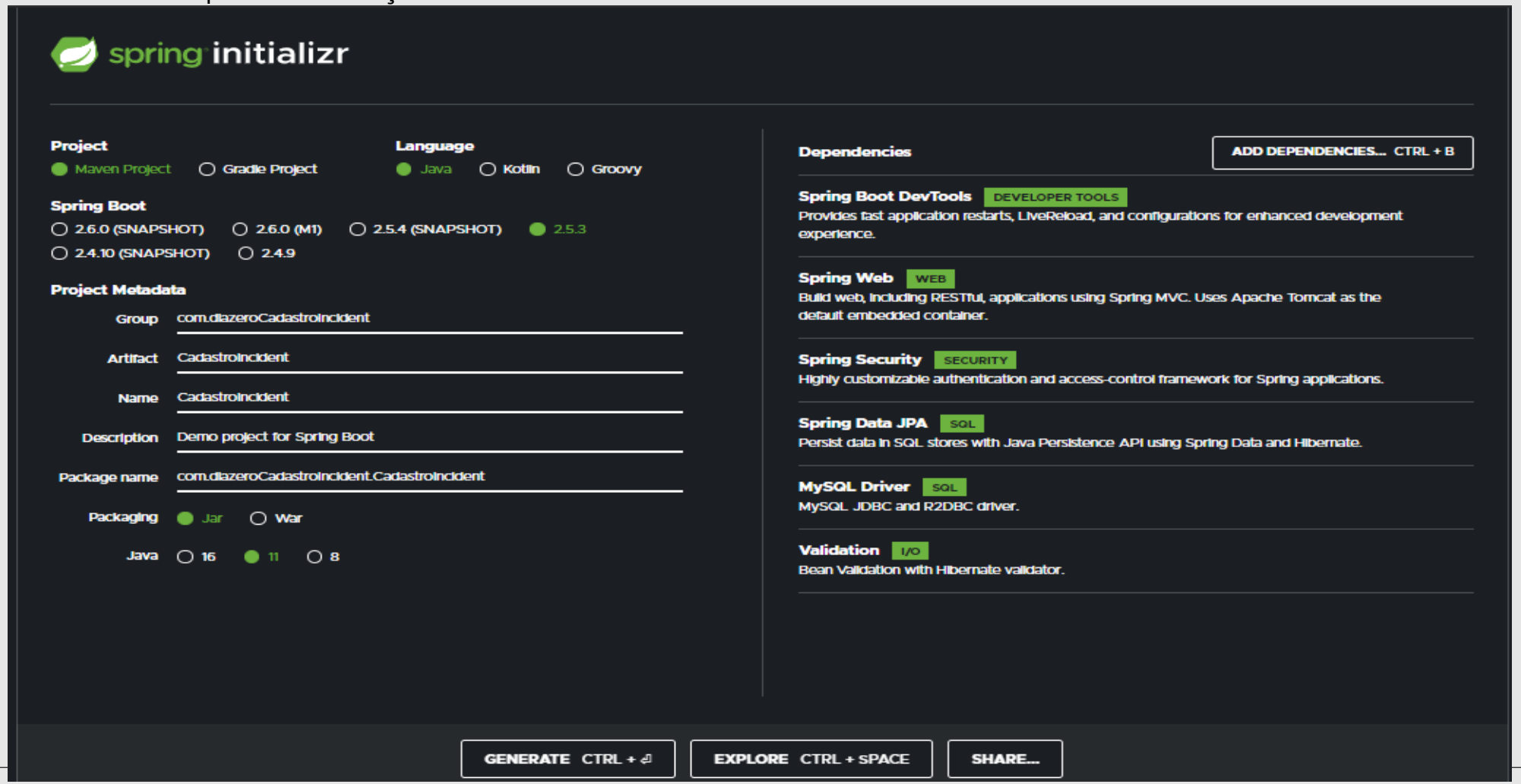
Construindo um API REST em que é possível cadastrar incidentes utilizando Java e o framework Spring.

Kevelly Braga

27/07/2021

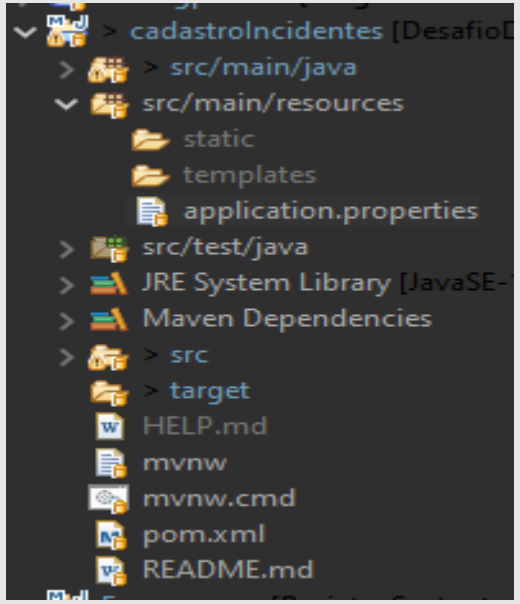
INICIANDO NOSSA API REST...

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:



PRÓXIMOS PASSOS:

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




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

```

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:

```java
@Entity
@Table(name = "incidente")

public class Incident {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long idIncident;

    @NotNull
    @Size(min = 3, max = 100)
    private String name;

    @NotNull
    @Size(min = 10, max = 500)
    private String description;

    @Temporal(TemporalType.TIMESTAMP)
    private Date creatAt = new java.sql.Date(System.currentTimeMillis());

    @Temporal(TemporalType.TIMESTAMP)
    private Date updateAt = new java.sql.Date(System.currentTimeMillis());

    @Temporal(TemporalType.TIMESTAMP)
    private Date closedAt = new java.sql.Date(System.currentTimeMillis());

    @ManyToOne
    @JsonIgnoreProperties ("usuario")
        private User user;
```

```java
@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.

```java
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);

}
```

```java
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, onde está toda nossa regra de negócio. Por exemplo, para cadastrar um incidente é preciso que tenha as seguintes informações: nome e descrição.

```java
@Service
public class IncidentService {

    private @Autowired IncidentRepository repositoryI;

    public Optional<Incident> registerIncident (Incident newIncident){

        Optional<List<Incident>> nameExisting = Optional.ofNullable(repositoryI.
                findAllByNameContainingIgnoreCase(newIncident.getName()));
        if (nameExisting.isPresent()) {
            return Optional.empty();
        }


    Optional<List<Incident>> descriptionExisting = Optional.ofNullable(repositoryI.
            findAllByDescriptionContainingIgnoreCase(newIncident.getName()));
    if (descriptionExisting.isPresent()) {
        return Optional.empty();
    }
    else {

    }
    return null;

}
}
```

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

```java
@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().getName());
                loginUser.get().setSenha(presentUser.get().getPassword());

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

Agora é a hora do Controller, o lugar em que administramos 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.

```java
@RestController
@RequestMapping ("/incident")
@CrossOrigin ("*")
public class IncidentController {

    @Autowired
    private IncidentRepository repository;

    @GetMapping
    public ResponseEntity<List<Incident>> GetAll (){
        return ResponseEntity.ok(repository.findAll());
    }


    @GetMapping ("/{idIncident}")
    public ResponseEntity<Incident> GetById (@PathVariable long idIncident) {

        return repository.findById(idIncident)
                .map(resp -> ResponseEntity.ok(resp))
                .orElse(ResponseEntity.notFound().build());
    }
    @GetMapping ("/name/{name}")
    public ResponseEntity<List<Incident>> GetByName (@PathVariable String name) {
        return ResponseEntity.ok(repository.findAllByNameContainingIgnoreCase(name));
    }
    @GetMapping ("/description/{description}")
    public ResponseEntity<List<Incident>> GetByDescriprion (@PathVariable String description) {
        return ResponseEntity.ok(repository.findAllByDescriptionContainingIgnoreCase(description));
    }
    @PostMapping
    public ResponseEntity<Incident> registerIncident (@Valid @RequestBody Incident incident){
        LocalDateTime date = LocalDateTime.now();
        return ResponseEntity.status(HttpStatus.CREATED).body(repository.save(incident));
    }
```

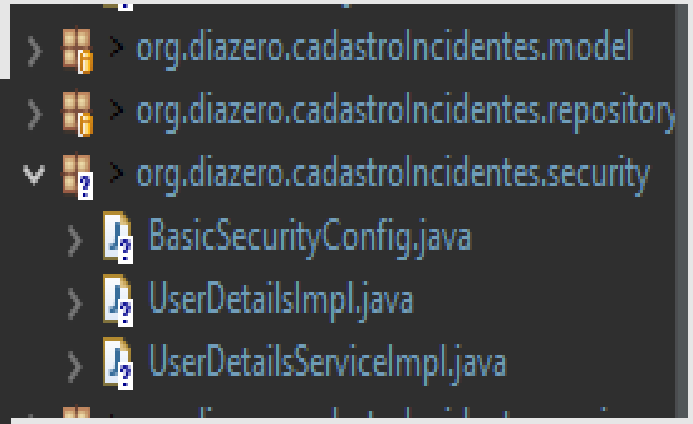
```
    return ResponseEntity.ok(repository.findAllByDescriptionContainingIgnoreCase(description));
}
@PostMapping
public ResponseEntity<Incident> registerIncident (@Valid @RequestBody Incident incident){
    LocalDateTime date = LocalDateTime.now();
    return ResponseEntity.status(HttpStatus.CREATED).body(repository.save(incident));
}


@PutMapping
 ResponseEntity<Incident> maintenanceIncident (@Valid @RequestBody Incident incidentUpdate){
    LocalDateTime date = LocalDateTime.now();
    Optional<Incident> existingIncident = repository.findById(incidentUpdate.getIdIncident());
    if (existingIncident.isPresent()) {
    existingIncident.get().setName(incidentUpdate.getName());
    existingIncident.get().setDescription(incidentUpdate.getDescription());
    existingIncident.get().setUpdate(incidentUpdate.getUpdate());
    return ResponseEntity.status(HttpStatus.CREATED).body(repository.save(existingIncident.get()));

    } else {
        return ResponseEntity.notFound().build();
    }
}

@DeleteMapping ("/{idIncident}")
public void removeIncidents (@PathVariable long idIncident) {
    repository.deleteById(idIncident);
}
```

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.

```
stController
ossOrigin(origins = "*", allowedHeaders = "*")
questMapping("/user")
lic 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.




```java
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() //restringir 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

}
```

```java
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;
    }
```

```java
@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.