Escola de Ciências e Tecnologia Departamento de Informática

Licenciatura em Engenharia Informática

Unidade curricular Tecnologias Web

Ano letivo 2020/2021

Relatório

Trabalho Prático 2ª parte

Docentes

Professor Pedro Salgueiro

Professor José Saias

Trabalho realizado por:

Pedro Claudino nº 39870

Ludgero Teixeira nº 41348

Évora, fevereiro de 2021

Descrição do trabalho

Para esta 2ª parte do trabalho prático foi pedido a implementação backend da aplicação web de venda de produtos online, implementada na 1ª parte do trabalho, por forma a obter uma aplicação web utilizável, de acordo com a seguinte descrição.

A implementação da mesma rege-se pelas seguintes condições/regras:

Todos os utilizadores, registados ou não, devem conseguir:
- Listar produtos;
- Visualizar detalhes de um produto;
- Pesquisar produtos;

Apenas os clientes registados devem conseguir:
- Criar encomendas (adicionar produtos ao carrinho de compras);
- Listar encomendas;

Os administradores devem conseguir realizar as seguintes tarefas:
- Administradores devem conseguir adicionar produtos;
- Administradores devem conseguir remover produtos;

A aplicação deve permitir:
- Autenticação dos utilizadores;
- Registo de utilizadores (os utilizadores devem conseguir registar-se na plataforma);

As alterações necessárias à 1ª parte do trabalho por forma a permitir que todas estas operações possam ser utilizadas pelos utilizadores da aplicação web são:

- Permitir o registo de utilizadores;
- Permitir a autenticação dos utilizadores;
- Criar encomendas;
- Listar encomendas;

A aplicação web deve ser implementada usando o conceito MVC e deve ser implementada usando o framework Spring, em conjunto com o módulo Spring Data JPA para persistir os dados, usando uma base de dados à sua escolha.
Para facilitar o processo de desenvolvimento deve usar o Gradle para gerir as dependências do projeto, bem como as tarefas de compilação e execução.

Permissões

Existem 2 tipos de permissões (**utilizadores e administradores**), onde os utilizadores autenticados podem aceder a todas as páginas expeto as de "administração" ("/add-product", "/ list-product"), caso haja uma tentativa de acesso por parte de utilizadores cujo "Role" não permita o acesso a estas páginas os mesmos são redirecionados para uma página com uma mensagem de erro.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/", "/index", "/index2","/product", "/add-client", "/new-client", "/advanced-search", "/search", "/erro","/frontend/**").permitAll()
            .antMatchers( ...antPatterns: "/add-product", "/list-product", "/admin").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and() HttpSecurity
        .formLogin() FormLoginConfigurer<HttpSecurity>
            .loginPage("/login")
            .successHandler(successHandler)
            .permitAll()
            .and() HttpSecurity
            .exceptionHandling().accessDeniedPage("/erro") ExceptionHandlingConfigurer<HttpSecurity>
            .and() HttpSecurity
        .logout() LogoutConfigurer<HttpSecurity>
            .permitAll();
}
```

Registo e Autenticação

Para a implementação das funcionalidades de registo e autenticação na nossa aplicação web, começamos por definir uma entidade (**Client.java)** que será estruturada pelas credenciais do utilizador (**username, password, role**).

Foi criada uma classe para a inserção destes mesmos dados da entidade numa base de dados complementada por um ficheiro de html onde os dados são submetidos pelo utilizador através de um form, retornando no fim uma view de confirmação do registo.

A classe **UserDetailsServiceImp.java** foi implementada, com o intuito de verificar se o utilizador se encontra na base de dados que é complementada com a classe **MyUserDetails.java** que retorna as credenciais do utilizador e verifica se a conta de utilizador está ativa, expirada ou bloqueada.

Com a estruturação destas classes temos também 2 páginas html:

- **Login.html** que correspondente ao login onde as credenciais são introduzidas através de um form e de seguida é feita a autenticação das mesmas.

- **NewClient.html** que corresponde ao registo de um utilizador onde são introduzidas as credenciais de registo sendo de seguida guardadas na base de dados.

```java
@Controller
public class NewClient {
    private static final Logger log = LoggerFactory.getLogger(NewClient.class);

    @Autowired
    private ClientRepository repository;

    @PostMapping("/new-client")
    public String newClient(
            @RequestParam(name="username", required=true, defaultValue="") String username,
            @RequestParam(name="password", required=true, defaultValue="") String password,
            @RequestParam(name="role", required=true, defaultValue="") String role,
            Model model)
    {

        String encodedPassword = new BCryptPasswordEncoder().encode(password);

        repository.save(new Client(username, encodedPassword, role));

        log.info("Users found with findAll():");
        log.info("-------------------------------");
        for (Client Client : repository.findAll()) {
            log.info(Client.toString());
        }
        log.info("");

        model.addAttribute( attributeName: "username", username);
        model.addAttribute( attributeName: "role", role);
        return "new-client-view";
    }
```

```java
@Entity
public class Client {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String username;
    private String password;
    private String role;

    protected Client() {}

    public Client( String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }
```

Criação e remoção de produtos

A criação de um produto assemelhasse muito a criação de um utilizador, as classes **Product.java** e **NewProduct.java** seguem a mesma implementação das classes referidas acima divergindo apenas nas variáveis que ambas as classes e entidades têm.

A classe **ListProduct.java** é criada com o intuito de mostrar todos os produtos já introduzidos na base de dados que é por sua vez complementada pela classe **DelProduct.java** que por sua vez irá remover determinado produto.

Com a estruturação destas classes temos também 2 páginas html:

- **NewProduct.html**, que correspondente a inserção de dados de um produto (nome, preço, descrição).

- **list-product-view.html**, que corresponde a mostragem da lista de produtos já introduzidos na base de dados, onde a cada produto corresponde um botão para remover o mesmo.

```java
@Controller
public class DelProduct {

    @Autowired
    private ProductRepository repository;
    private static final Logger log = LoggerFactory.getLogger(DelProduct.class);

    @GetMapping("/del-product")
    public String removeProduct(@RequestParam(name="id") Long id, Model model){
        log.info(id+"");
      repository.deleteById(id);

      return "redirect:/list-product";
    }
}
```

```java
@Controller
public class ListProduct {
    private static final Logger log = LoggerFactory.getLogger(ListProduct.class);

    @Autowired
    private ProductRepository repository;

    @GetMapping("/list-product")
    public String listProduct(Model model)
    {
        List<Product> ProductList = (List<Product>) repository.findAll();

        log.info("Products found with findAll():");
        log.info("-------------------------------");
        for (Product Product : repository.findAll()) {
            log.info(Product.toString());
        }
        log.info("");

        model.addAttribute( attributeName: "ProductList", ProductList);
        return "list-product-view";
    }
}
```

Encomendas

Na classe **Order.java** é feita uma relação entre as encomendas e os produtos, sendo que nesta classe existe uma lista " **Cart** " com o intuito de guardar os produtos na encomenda.

A classe **NewOrder.java** é um controlador onde são adicionados os produtos ao " **Cart** " e onde também é feita a confirmação da encomenda e a limpeza do " **Cart** " após a realização da mesma.

Com a estruturação destas classes temos também 2 páginas html:

- **Cart.html**, mostra os produtos no carrinho.

- **list-order-view.html**, mostra a lista de encomenda realizadas pelo utilizador.

```
    @GetMapping("/add-cart")
    public String addCart(@RequestParam(name="id") Long id, Model model)
    {
        order.getCart().add(repositoryp.findById(id).get());
        model.addAttribute( attributeName: "ProductList", order.getCart());
        return "redirect:/list-cart";
    }

    @GetMapping("/list-cart")
    public String listProduct(Model model)
    {


        log.info("Orders found with findAll():");
        log.info("-------------------------------");
        for (Order Order : repository.findAll()) {
            log.info(Order.toString());
        }
        log.info("");

        model.addAttribute( attributeName: "ProductList", order.getCart());
        return "cart";
    }

    @GetMapping("/order-done")
    public String orderDone(Model model)
    {

        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username;
        if(principal instanceof UserDetails){
            username =((UserDetails)principal).getUsername();
        }else{
            username= principal.toString();
        }

        order.setUsername(username);

        repository.save(order);
        order = new Order();
        return "redirect:/list-cart";
    }
}
```

```java
@Entity
@Table(name="orders")
public class Order {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String username;

    @OneToMany(mappedBy = "order")
    private List<Product> cart;

    public Order(Long id, String username, List<Product> cart) {...}

    public Order() { this.cart= new ArrayList<>(); }
```

Pesquisa de produtos

A pesquisa de produtos é feita recorrendo ao input inserido pelo utilizador na barra de pesquisa, onde irá procurar numa lista de produtos, por nomes de produtos existentes no repositório que coincidam com o input introduzido.

```java
public interface ProductRepository extends CrudRepository<Product, Long> {
    List<Product> findByName(String name);

    Optional<Product> findById(@Param("id") Long id);

    @Query(value = "FROM Product WHERE name LIKE %:name%")
    List<Product> searchProduct(@Param("name") String name);

}
```

```java
@Controller
public class ProductSearch {
    @Autowired
    private ProductRepository repository;
    private static final Logger log = LoggerFactory.getLogger(DelProduct.class);

    @PostMapping("/search-product")
    public String searchProduct(@RequestParam(name="name") String name, Model model){
        log.info(name+"");
        List<Product> product = repository.searchProduct(name);
        model.addAttribute( attributeName: "product", product);


        return "/index2";
    }


}
```

Dificuldades e limitações da Aplicação Web

Apesar de acharmos que a realização deste trabalho tenha sido concluída com sucesso, existem pequenos aspetos da aplicação web que não ficaram concluídas com sucesso.

São estes:

- A remoção dos produtos do “carrinho” de encomenda.
- Na listagem das encomendas a aplicação não discrimina quais os produtos que se encontram inseridos em cada encomenda.

Bibliografia

Para a realização deste trabalho guiamos nos pelos Power-Points cedidos pelos docentes, tal como os exercícios realizados nas aulas práticas.

Outros recursos:

- https://o7planning.org/10605/create-a-java-shopping-cart-web-application-using-spring-mvc-and-hibernate
- https://docs.spring.io/spring-framework/docs/current/reference/html/
- https://www.edureka.co/blog/mvc-architecture-in-java/