SQUARD 06
- 202002721 | Erik Nathan de Oliveira Batista
- 202002509 | Itallo Monteiro Minatti
- 202002088 | Julio Cesar de Barros
- 202003105 | Leonardo Viana Gouveia Vieira de Morais
- 202002790 | Veron Thiago de Paula Ferreira
- 202005574 | Jhonny Glaucio Nascimento de Melo
- 202002043 | Beatriz Lira Martins

APS – ESTRUTURA DE DADOS

1 – Introdução

Nosso projeto é um sistema de cadastro de alunos de uma faculdade, nos baseamos no fato de que o sistema na nossa faculdade é bastante falho.
O objetivo desse trabalho é, implementar o CRUD (Acrônimo do inglês create, read, update, e delete) são as quatro operações básicas de criação, consulta, atualização, e destruição de dados.

Uma coleção de atividades, tais como inserir, suprimir e consultar, encapsulada junto com uma estrutura passiva, como um dicionário (conjunto de verbetes), pode ser considerada um tipo abstrato de dados (TAD).Definido dessa forma, o TAD DICIONÁRIO fica representado no nível de abstração mais alto possível: o nível conceitual. Em um nível de abstração mais baixo, denominado nível de design, a estrutura passiva deve ser representada por um modelo de dados (por exemplo: sequência ou árvore binária de busca) e as operações devem ser especificadas através de procedimentos cuja representação não dependa de uma linguagem de programação.

2 – Implementação

Estrutura de dados:
Para implementação do trabalho foram criados ArrayList para listar os alunos já cadastrados, por matrícula ou por nome. Usamos a estrutura para percorrer as mesmas.

Ex:

```
public interface CursoInterfSer {



    public List<Curso> listar();
```

}

Nós:

Criamos um novo nó com o valor passado, no auxiliar fica no inicio, se a lista estiver vazia, senão vai até o final da lista.Precisamos de dois nós auxiliares, um para que possamos ligar o novo nó ao seu proximo (vamos chamar ele de P)e outro para conectar o nó anterior à P ao novo nó, até que você encontre o local correto para nó ser inserido em ordem crescente vá para o próximo nó. Devemos verificar aqui se já chegou ao final da lista:auxilia != null

```
	}


	/**Inserindo um nó em ordem, passando o valor*/
	public void inserir(T  valor) {
		//Cria um novo nó com o valor passado
		No<T> novo_no = new No<T>(valor);

		/*Precisamos de dois nós auxiliares, um para que possamos ligar o novo nó ao seu proximo
		 * (vou chamar ele de P)e outro para conectar o nó anterior à P ao novo nó*/
		No<T> auxiliar = primeiro;
		No<T> auxiliar2 = null;

		/*Até que você encontre o local correto para nó ser inserido em ordem crescente
		 *vá para o próximo nó. Devemos verificar aqui se já chegou ao final da lista:
		 auxilia != null
		 */
		while((auxiliar != null) && ( auxiliar.obterValor().compareTo( novo_no.obterValor() )) == -1  )
		{
			//auxiliar2 guarda o valor de auxiliar, antes dele pular para o próximo
			auxiliar2 = auxiliar;
			//pula para o próximo
			auxiliar = auxiliar.obterProximo();
		}

		//é o primeiro nó
		if(this.primeiro == null) {
			this.primeiro = novo_no;

		//o nó deve ficar antes do primeiro (é menor que ele)
		}else if(auxiliar == this.primeiro) {

			novo_no.inserirProximo(this.primeiro);
```

```
this.primeiro = novo_no;
```

Listas Encadeadas

São listas onde cada elemento contido em uma lista está armazenado em um TAD chamado elemento de lista. Cada elemento de lista referencia o próximo e só é alocado dinamicamente quando é necessário para referenciar o primeiro elemento utilizamos um TAD cabeça de lista.

Ex:

```
while((auxiliar != null) && (auxiliar.obterValor().compareTo( valor )) != 0  )
    {
        //auxiliar2 guarda o valor de auxiliar, antes dele pular para o próximo
        auxiliar2 = auxiliar;
        //pula para o próximo
        auxiliar = auxiliar.obterProximo();
    }

    //se o nó a ser removido for o primeiro
    if(auxiliar == this.primeiro) {

        No retorno = this.primeiro;
        this.primeiro = this.primeiro.obterProximo();
        return retorno;

    //remove o nó da lista, ligando o nó anterior ao próximo do nó achado
    }else if(auxiliar != null)
        auxiliar2.inserirProximo(auxiliar.obterProximo());
```

```
        //retorna o nó (null ou o nó achado)
        return auxiliar;
```

Login com banco de dados:

Utilizamos a implementação do BCryptPasswordEncoder no projeto, para torna-lo mais seguro encriptando nossas senhas do banco de dados e preservando a confidencialidade das mesmas. Aqui mostraremos parte do código feito:

```
@Override
  protected void configure (HttpSecurity http) throws Exception{
     http.authorizeRequests()
          .antMatchers("/css/**", "/js/**", "/images/**").permitAll()
          .antMatchers("/cadastrar").authenticated()
          .antMatchers("/").authenticated()
          .anyRequest().authenticated()
          .and()
          .formLogin()
          .loginPage("/login")
          .defaultSuccessUrl("/")
          .permitAll()
          .and()
          .logout()
          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
          .logoutSuccessUrl("/login?logout").permitAll();

  }
```

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception{

        auth.inMemoryAuthentication()
                .withUser("user")
                .password(passwordEncoder().encode("user"))
                .authorities("USER")
            .and()
                .withUser("admin")
                .password(passwordEncoder().encode("admin"))
                .authorities("ADMIN");

    }

}
```

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código está dividido entre três arquivos principais: controller, model, e interface Service. O tipo abstrato de dados está localizado no model.

Criamos características na pasta modelo todas as classes são abstrações, as ações estão localizadas na pasta service e na pasta model.

A IDE utilizada foi o intellij, usando o sistema operacional Windows para executá-lo. Para executá-lo basta apertar o botão “Run” na IDE do intellij.

Tecnologias utilizadas no projeto: Spring boot, Spring Security, JPA repository, Thymeleaf.