LEI

Relatório

CONSTRUÇÃO DE SISTEMAS DE SOFTWARE

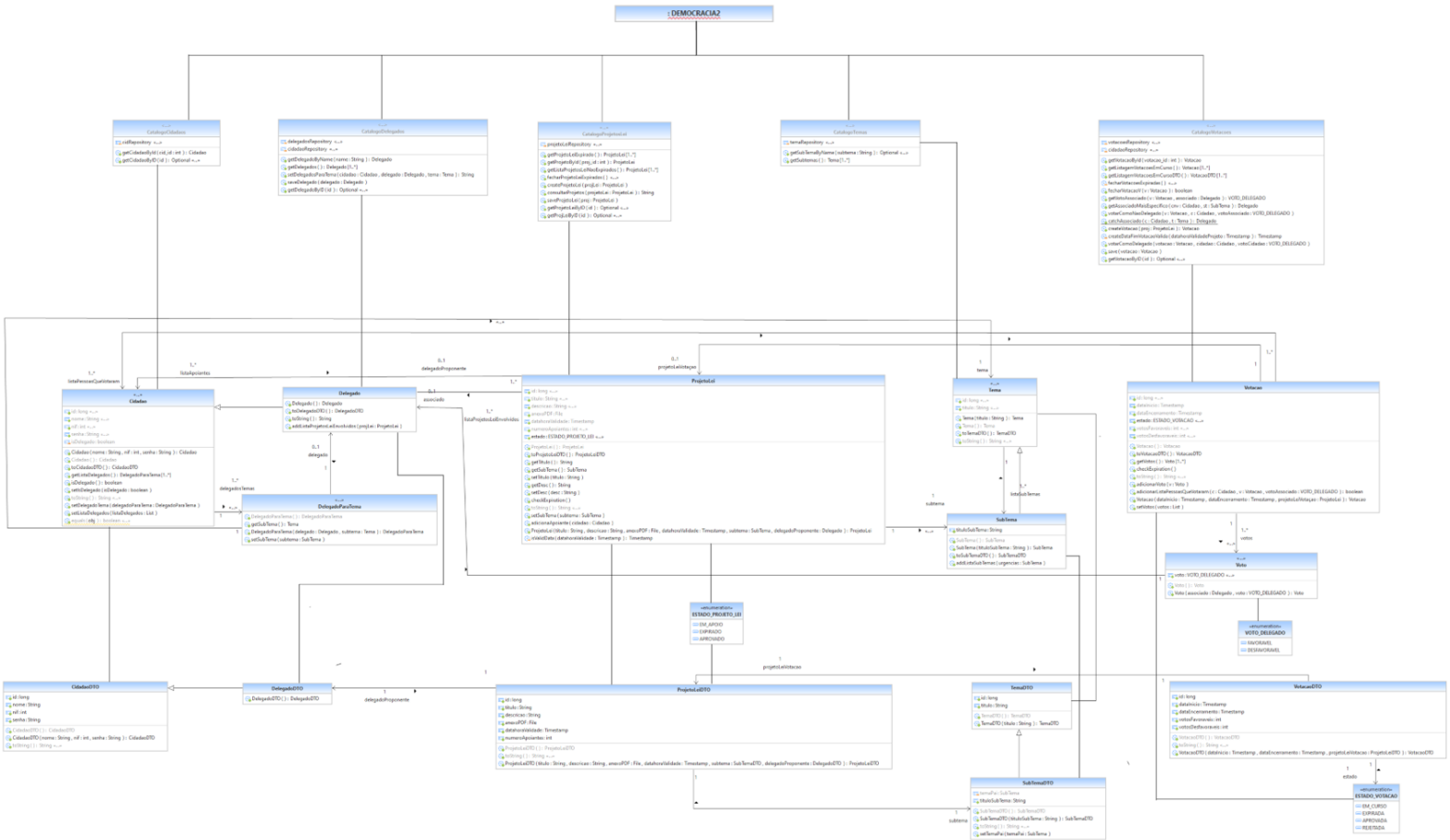
Trabalho realizado por:

Paulo Bolinhas fc56300

João Santos fc57103

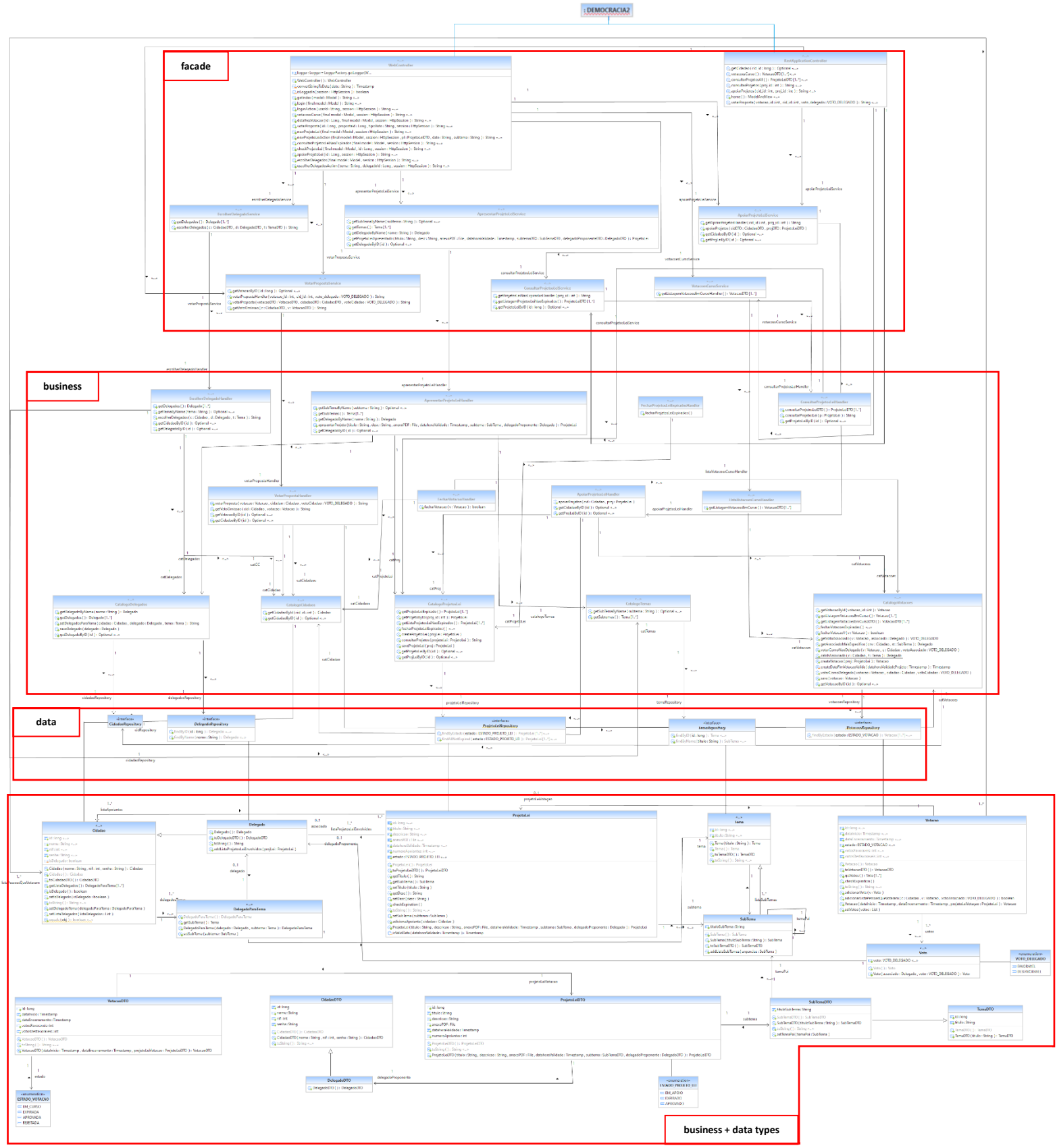
Autores:

- João Gonçalo Santos, fc57103
- Paulo Bolinhas, fc56300



Autores:

- João Gonçalo Santos, fc57103
- Paulo Bolinhas, fc56300



API REST

```java
@RestController
@RequestMapping("/api")
public class RestApplicationController {

    @Autowired private VotacoesCursoService votacoesCursoService;
    @Autowired private ConsultarProjetosLeiService consultarProjetosLeiService;
    @Autowired private ApoiarProjetoLeiService apoiarProjetoLeiService;
    @Autowired private VotarPropostaService votarPropostaService;

    @GetMapping("/cidadao/{cid_id}")
    Optional<CidadaoDTO> getCidadao(@PathVariable long cid_id) {
        return apoiarProjetoLeiService.getCidadaoByID(cid_id);
    }

    @GetMapping("/votacoesCurso")
    List<VotacaoDTO> votacoesCurso() {
        return votacoesCursoService.getListagemVotacoesEmCursoHandler();
    }

    @GetMapping("/consultarProjetos")
    List<ProjetoLeiDTO> consultarProjetosAll() {
        return consultarProjetosLeiService.getListagemProjetosLeiNaoExpirados();
    }

    @GetMapping("/consultarProjetos/{proj_id}")
    String consultarProjetos(@PathVariable int proj_id) {
        return consultarProjetosLeiService.getProjetosLeiNaoExpiradosHandler(proj_id);
    }

    @GetMapping("/apoiarProjetos/{cid_id}/{proj_id}")
    String apoiarProjetos(@PathVariable int cid_id, @PathVariable int proj_id) {
        return apoiarProjetoLeiService.getApoiarProjetosHandler(cid_id, proj_id);
    }

    @GetMapping("/votarProposta/{votacao_id}/{cid_id}/{voto_delegado}")
    String votarProposta(@PathVariable int votacao_id, @PathVariable int cid_id, @PathVariable VOTO_DELEGADO voto_delegado) {
        return votarPropostaService.votarPropostaHandler(votacao_id, cid_id, voto_delegado);
    }

    @RequestMapping(value = "/index")
    public ModelAndView home() {
        return new ModelAndView("index");
    }
}
```

A API REST consiste num controlador Spring Boot denominado RestApplicationController (definido com a anotação @RestController), que é responsável por lidar com os pedidos HTTP recebidos e fornecer as respostas adequadas. O controlador é mapeado para o caminho base "/api".

A arquitetura interna da API é organizada em camadas, onde cada camada possui uma responsabilidade específica. Assim, temos o RestController, que vai usar os serviços dos casos de uso específicos da API (injetados no controlador), anotados com @Service, para chamar os handlers, anotados com @Component, desses mesmos casos de uso. As intâncias de cada serviço e handler são anotadas com o @Autowired (de modo a serem instanciadas automaticamente). Importante referir que, nestas chamadas, apenas usamos objetos DTOS, pois queremos encapsular apenas os atributos necessários a serem mostrados na API.

VotacoesCursoService: Este serviço é responsável por lidar com as operações relacionadas ao caso de uso “Listar as votações em curso”. A função votacoesCurso() retorna uma lista de objetos VotacaoDTO, que contém as votações que ainda estão em curso (não expiradas). Assim, ao aceder ao link localhost:8080/api/votacoesCurso, através da anotação @GetMapping("/votacoesCurso"), é recebida uma resposta no formato json, com essas votações.

ConsultarProjetosLeiService: Este serviço lida com as operações relacionadas ao caso de uso “Consultar projectos de lei”. A função consultarProjetosAll() retorna uma lista de objetos ProjetoLeiDTO, que contém todos os projetos de lei não expirados. A função consultarProjetos() recebe o ID do projeto e retorna uma string que representa o projeto escolhido. É através dos @GetMapping("/consultarProjetos") e @GetMapping("/consultarProjetos/{proj_id}"), respectivamente, que podemos aceder aos links localhost:8080/api/consultarProjetos e localhost:8080/api/consultarProjetos/{proj_id} e receber as devidas

respostas. No caso da segunda função, no campo “proj_id” apenas é preciso substituir com o projeto do id desejado. Tal é permitido pois no argumento da função passamos a anotação @PathVariable que atribui o valor do campo ao argumento passado na função. Importante referir que depois, já no serviço obtemos o projeto de lei desejado por esse id.

ApoiarProjetoLeiService: Este serviço trata das operações relacionadas ao caso de uso “Apoiar projectos de lei”. A função getCidadao() retorna um objeto Optional<CidadaoDTO>, que representa informações sobre o cidadão com base no ID fornecido. A função apoiarProjetos() permite dado o id do cidadão e o id do projeto de lei, fazer com que o cidadão passe a apoiar esse projeto.

VotarPropostaService: Este serviço lida com as operações relacionadas ao caso de uso “Votar numa proposta”. A função votarProposta() recebe o ID da votação, o ID do cidadão e o tipo de voto delegado e retorna uma string que representa o resultado da votação.

É importante salientar também que são usados Optional<object>, já nos serviços, para lidar com as exceções (existe ou não objeto para o id dado), através do método .isPresent(), como podemos ver nos casos abaixo:

```
public String getProjetosLeiNaoExpiradosHandler(int proj_id) {
        Optional<ProjetoLei> pl = consultarProjetosLeiHandler.getProjetoLeiByID((long) proj_id);
        if(pl.isPresent()) {
            return consultarProjetosLeiHandler.consultarProjetosLei(consultarProjetosLeiHandler.getCatProj().getProjetoById(proj_id));
        } else {
            return "Não existem projetos com o ID especificado";
        }
}
```

```
public String getApoiarProjetosHandler(int cid_id, int proj_id) {
    Optional<ProjetoLei> pl = apoiarProjetosLeiHandler.getProjLeiByID((long) proj_id);
    Optional<Cidadao> c = apoiarProjetosLeiHandler.getCidadaoByID((long) cid_id);
    if(pl.isPresent() && c.isPresent()) {
        apoiarProjetosLeiHandler.apoiarProjetos(apoiarProjetosLeiHandler.getCatCidadao().getCidadaoById(cid_id), apoiarProjetosLeiHandler.getCatProj().getProjetoById(proj_id));
        return "Projeto Apoiado";
    } else {
        return "Não existem projetos ou cidadaos com o ID especificado";
    }
}
```

```
public String votarPropostaHandler(int votacao_id, int cid_id, VOTO_DELEGADO voto_delegado) {
    Optional<Votacao> v = votarPropostaHandler.getVotacaoByID((long) votacao_id);
    Optional<Cidadao> c = votarPropostaHandler.getCidadaoByID((long) cid_id);
    if(v.isPresent() && c.isPresent()) {
        return votarPropostaHandler.votarProposta(votarPropostaHandler.getCatVotacoes().getVotacaoById(votacao_id), votarPropostaHandler.getCatCidadao().getCidadaoById(cid_id), voto_delegado);
    } else {
        return "Não existem cidadaos ou votacoes com o ID especificado";
    }
}
```

Além dos serviços mencionados acima, o controlador rest também possui um método home() mapeado para a rota "/index", que retorna um objeto ModelAndView, representando a página inicial.

ThymeLeaf

A WebController é uma classe responsável por lidar com pedidos HTTP e controlar o fluxo da aplicação web com rendering Server-Side.

> Primeiramente convém referir que:
>
> @PostMapping: Mapeia um pedido HTTP POST para um método de manipulação de objetos. Usamos nos métodos acionados ao clicar num botão.
>
> HttpSession: Fornece acesso à sessão do user num aplicativo da web. Usamos para separar as ações dos diferentes cidadãos e entre cidadãos e delegados.
>
> Model: Passa dados do controlador para a template durante o processo de renderização. Usamos para visualizar as nossas templates html.

Assim, a nossa classe WebController é constituída por:

Serviços:

A WebController depende de vários serviços para realizar as operações específicas de cada caso de uso. Estes são injetados na classe através da anotação @Autowired. Estes serviços encapsulam a lógica de negócio e fornecem funcionalidades específicas para a aplicação.

Métodos:

isLoggedIn(HttpSession session): É um método privado que verifica se o usuário está autenticado. Recebe um objeto HttpSession como parâmetro e verifica se o atributo "userId" está presente na sessão.

getIndex(Model model): Mapeado para o link com URL "/" através da anotação @RequestMapping e retorna a template "login" (login.html, que contém o design do endpoint). É utilizado como página inicial da nossa app.

login(Model model): Mapeado para o link com URL "/login" através da anotação @GetMapping e retorna a template "login". Este método é chamado quando o user acessa a página de login.

loginAction(String userId, HttpSession session): Mapeado para o link com URL "/login" através da anotação @PostMapping, trata da ação de login do user. Verifica se o user com o ID fornecido existe no sistema e, em caso afirmativo, armazena o ID do user na sessão e redireciona para a página "/votacoesCurso". Caso contrário, retorna a template "errorLogin".

votacoesCurso(Model model, HttpSession session): Mapeado para o link com URL "/votacoesCurso" através da anotação @GetMapping, retorna a template "votacoesCurso_list" que exibe a lista de votações em curso. Verifica se o user está autenticado e, se sim, adiciona a lista de votações ao modelo. Caso contrário, redireciona para a página de login.

detalhesVotacao(Long id, Model model, HttpSession session): Mapeado para o link com URL "/votacoesCurso/{id}" através da anotação @GetMapping, retorna a template "votacao_detail" com os detalhes de uma votação específica. Verifica se o user está autenticado, procuara o user atual na sessão e procura a votação correspondente ao ID fornecido. Se a votação existir, adiciona-a ao modelo juntamente com o voto de omissão do user. Caso contrário, retorna a template "error404" (html com exceção lançada).

votarProposta(Long id, Long propostaId, String tipoVoto, HttpSession session): Mapeado para o link com URL "/votacoesCurso/{id}/votar" através da anotação @PostMapping, trata a ação de votar em uma proposta de votação. Obtém a votação correspondente ao ID fornecido, busca o usuário atual na sessão e registra o voto na proposta de acordo com o tipo de voto selecionado. Em seguida, redireciona de volta para a página de detalhes da votação.

newProjetoLei(Model model, HttpSession session): Mapeado para o link com URL "/projetoLei/new" através da anotação @GetMapping, retorna a template "projetoLei_add" para adicionar um novo projeto de lei. Verifica se o user está autenticado e, em caso afirmativo, adiciona um novo objeto ProjetoLei ao modelo, juntamente com a lista de temas disponíveis. Caso contrário, redireciona para a página de login.

newProjetoLeiAction(Model model, HttpSession session, ProjetoLeiDTO pl, String date, String subtema): Mapeado para o link com URL "/projetoLei/new" através da anotação @PostMapping, trata o caso de adicionar um novo projeto de lei. Obtém os dados do projeto de lei fornecidos no formulário, como o título, descrição, data e subtema. Verifica se o user está autenticado e se é um delegado. Se todas as verificações passarem, adiciona o projeto de lei à BD e redireciona para a página de detalhes do projeto de lei adicionado. Caso contrário, retorna a template "errorNaoDelegado" (html com exceção lançada).

consultarProjetosLeiNaoExpirados(Model model, HttpSession session): Mapeado para o link com URL "/projetosLeiNaoExpirados" através da anotação @GetMapping, retorna a template "projetosLeiNaoExpirados_list" que exibe a lista de projetos de lei não expirados. Verifica se o user está autenticado e, se sim, adiciona a lista de projetos de lei ao modelo. Caso contrário, redireciona para a página de login.

checkProjetoLei(Model model, Long id, HttpSession session): Mapeado para o link com URL "/projetosLei/{id}" através da anotação @GetMapping, retorna a visualização "projetoLei_detail" com os detalhes de um projeto de lei específico. Verifica se o user está autenticado, procura o projeto de lei correspondente ao ID fornecido e, se existir, adiciona-o ao modelo. Caso contrário, retorna a template "error404".

apoiarProjetoLei(Long id, HttpSession session): Mapeado para o link com URL "/projetosLei/{id}/apoio" através da anotação @PostMapping, trata a ação de apoiar um projeto de lei. Obtém o user atual da sessão, procura o projeto de lei correspondente ao ID fornecido e regista o apoio do user ao projeto de lei. Em seguida, redireciona de volta para a página de detalhes do projeto de lei.

escolherDelegados(Model model, HttpSession session): Mapeado para o link com URL "/escolherDelegados" através da anotação @GetMapping, retorna a template "escolher_delegados" para escolher delegados. Verifica se o user está autenticado e, em caso afirmativo, adiciona a lista de delegados e temas disponíveis ao modelo. Caso contrário, redireciona para a página de login.

escolherDelegadosAction(String tema, Long delegadoId, HttpSession session): Mapeado para o link com URL "/escolherDelegados" através da anotação @PostMapping, trata a ação de escolher delegados. Obtém o user atual da sessão, o tema e o delegado selecionado e regista a escolha do delegado para o tema. Em seguida, redireciona de volta para a página de escolher delegados.

JAVA FX

A classe Democracia2 é a classe principal da nossa aplicação desktop com JavaFX. Em resumo, a classe Democracia2 configura e exibe a janela principal da aplicação JavaFX. Ela carrega o layout da interface do user a partir do arquivo FXML (login.fxml e inicia o controlador do login. Esta classe representa o ponto de partida da aplicação e define a estrutura básica da interface do user.

O LoginController é responsável por fazer a gestão a lógica de autenticação e login da nossa app. Ele lida com a interação do user no formulário de login, valida o ID deste, faz uma solicitação para obter informações do user com base no ID e, se o login for bem-sucedido, exibe a próxima tela do menu principal.

A classe MainMenuController é responsável pela lógica da interface do menu principal da aplicação. Esta facilita a navegação entre diferentes telas da aplicação, lista de projetos e lista de votaçoes, carregando os arquivos FXML correspondentes e inicializando os controladores associados a eles. Ela também faz a gestão da passagem do modelo de dados e outras informações relevantes entre os controladores. Aqui, podemos também fazer logout da aplicação, através de uma chamada goBack() na qual se perde o id do user atual e somos redirecionados à página de login.

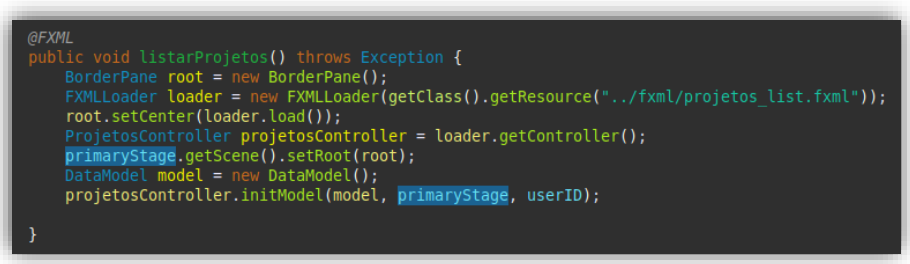
O ProjetosController é responsável por apresentar uma template que exibe uma tabela de projetos. O controlador recupera os dados do projeto de um endpoint da API, preenche a tabela com os dados e permite que o user clique em um projeto para visualizar seus detalhes.

Já o ProjetoDetailsController é responsável por tratar da interação com um projeto específico. O controlador recebe um projeto como parâmetro, define os valores dos vários campos com base nos dados do projeto e fornece a funcionalidade para oferecer suporte a interações, neste caso, a do caso de uso pedido, apoiar projeto.

Quanto ao VotacoesController e ao VotacoesDetailsController, são semelhantes ao ProjetosController e ProjetoDetailsController, mas lidam com dados relacionados às votações. O VotacoesController recupera dados de votação de um endpoint da API, preenche uma tabela com os dados e permite que o usuário clique em uma votação para visualizar seus detalhes. O VotacoesDetailsController define os valores dos vários campos com base nos dados da votação e fornece a funcionalidade para oferecer suporte a interações, neste caso, a do caso de uso pedido, votar proposta, a favor ou contra.

Deixar referido que "@FXML" é uma anotação usada no contexto do JavaFX para marcar elementos de interface gráfica (UI) definidos num arquivo FXML.

Para navegar entre controladores a lógica de código foi sempre a mesma (exemplo no MainMenuController):



```
@FXML
public void listarProjetos() throws Exception {
    BorderPane root = new BorderPane();
    FXMLLoader loader = new FXMLLoader(getClass().getResource("../fxml/projetos_list.fxml"));
    root.setCenter(loader.load());
    ProjetosController projetosController = loader.getController();
    primaryStage.getScene().setRoot(root);
    DataModel model = new DataModel();
    projetosController.initModel(model, primaryStage, userID);

}
```

Importante referir ainda que todos os controllers apresentam uma funcionalidade goBack() que permite voltar atrás na aplicação e que todas as funcionalidades estão associadas a botões.

Também salientar que as chamadas de endpoints da API, realizadas no login, na lista de projetos, na lista de votações e nos detalhes dos projetos e votações, são sempre feitas desta forma (exemplo no LoginController):

```
private Optional<CidadaoDTO> getCidadaoByID(Long userID) {
    try {
        // Make an HTTP GET request to the API endpoint
        URL url = new URL("http://localhost:8080/api/cidadao/" + userID);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        // Check the response code
        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            // Read the response from the API
            BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line);
            }
            reader.close();

            // Parse the JSON response into CidadaoDTO object
            ObjectMapper objectMapper = new ObjectMapper();
            CidadaoDTO cidadaoDTO = objectMapper.readValue(response.toString(), CidadaoDTO.class);

            return Optional.ofNullable(cidadaoDTO);
        } else {
            return Optional.empty();
        }
    } catch (Exception e) {
        e.printStackTrace();
        return Optional.empty();
    }
}
```

Assim, de um modo geral:

A classe principal, "Democracia2", configura e exibe a janela principal da aplicação, carregando o layout da interface de user e iniciando o controlador do login. O "LoginController" é responsável pela lógica de autenticação e login, validando as informações fornecidas pelo user. Uma vez autenticado, o user é direcionado para o "MainMenuController", que gere a interface do menu principal, permitindo a navegação entre telas e a passagem de dados entre controladores. Nesta tela, é possível fazer logout, retornando ao login. O "ProjetosController" e o "ProjetoDetailsController" tratam da exibição e interação com projetos específicos. O "ProjetosController" recupera os dados dos projetos de um endpoint da API, preenche uma tabela e permite que o user clique em um projeto para ver seus detalhes no "ProjetoDetailsController". Este último define os valores dos campos com base nos dados do projeto e permite apoiar um projeto. Da mesma forma, o "VotacoesController" e o "VotacoesDetailsController" lidam com dados de votação. O "VotacoesController" recupera os dados das votações da API, preenche uma tabela e permite que o user clique numa votação para ver os seus detalhes no "VotacoesDetailsController". Este controlador define os valores dos campos com base nos dados da votação e permite votar a favor ou contra uma proposta.