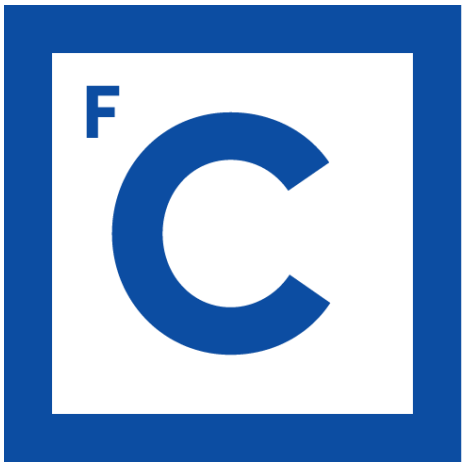

Ciências ULisboa

Construção de Sistemas de Software

Democracia 2.0 Fase 2

2022/2023

Relatório

Miguel Agostinho fc53475

Guilherme Garrilha fc53838

João Pena fc54477

Relativamente a esta fase o grupo não realizou a tarefa da implementação de JavaFX. Foram-nos dados vários erros de compatibilidade que não conseguimos resolver tais como "Javafx runtime components are missing and are required to run this application" ou criar module-info que não invalidasse as classes do projeto todas, mesmo em duas pastas root em separados. Com isto, também o client.sh não foi feito visto que o JavaFX também não foi realizado. Fora isso, foram todos os requisitos cumpridos.

Relativamente à arquitetura dos componentes rest e web, foi criada uma package presentation, onde devem estar os componentes view, e foram criadas outras duas subpackages de Restcontrollers e webcontrollers.
Na package REST controllers, foram criadas duas classes RestPollController e RestProjectController, para representar REST API Controllers.

Na package Web Controllers, foram criadas três classes, WebDelegateController, WebPollController e WebProjectController, para representar os Controllers utilizados em web.

Relativamente aos controllers REST, tanto no RestPollController como no RestProjectController temos que usamos a REST API para trocar informação entre sistemas de maneira segura para várias tarefas. Foram também desenhados DTOs para as classes Delegate, Poll, Project e Theme, para que exista melhor transferência de dados entre processos, e é extremamente indicado para o REST API.

RestPollController:

```
@RestController()
@RequestMapping("api")
public class RestPollController {

  @Autowired private OngoingPollsService ongoingPollsService;

  @Autowired private VoteProposalService voteProposalService;
```

@RestController() para definir um rest api controller, e fazer com que o endereço tenha (/api) com @RequestMapping("api"). É-se utilizado os Services de ongoingPolls e voteProposal para utilizar os seus métodos.

```
@GetMapping("/polls")
ResponseEntity<?> getPolls() {

  try {

    return ResponseEntity.ok(ongoingPollsService.getOngoingPolls());

  } catch (PollNotFoundException e) {

    return ResponseEntity.notFound().build();
  }
```

Temos o @GetMapping("/polls") que se executar com sucesso irá nos dar um Get com endereço /api/polls. São usadas Response Entities para termos respostas de sucesso (ou não) para a execução dos métodos.

```java
@GetMapping("/polls/{id}")
ResponseEntity<?> getDelegateVote(@PathVariable Long id, @RequestParam Long citizenId) {

  try {

    return ResponseEntity.ok(voteProposalService.pickPolls(id, citizenId));

  } catch (NoDelegateAssignedException e) {

    return ResponseEntity.status(HttpStatus.BAD_REQUEST)
        .body("Nao existe delegado associado ao tema do projeto desta votacao");

  } catch (PollNotFoundException e) {

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Votacao nao encontrada");

  } catch (CitizenNotFoundException e) {

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Cidadao nao existe");
  }
}
```

Temos @GetMapping("/polls/{id}"), tendo que @PathVariable id fará com que o id seja necessário para o url, e o @RequestParam CitizenId também fique bound ao pedido de URL. Para além disso, acontece o mesmo que no getPolls(), temos a ResponseEntity que nos dá a resposta da execução do método com sucesso, caso contrário, em falta de algum componente, levanta exceções de cada componente.

```java
@PutMapping("/polls/{id}")
ResponseEntity<?> voteOnPoll(
    @PathVariable Long id,
    @RequestParam Long citizenId,
    @RequestParam Boolean DelegateVotes,
    @RequestParam Boolean choice) {

  try {

    voteProposalService.voterChoice(citizenId, id, DelegateVotes, choice);

    return ResponseEntity.ok("Voto recebido!");

  } catch (AssignedDelegateNullVoteException e) {

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Delegado associado não votou");

  } catch (HasAlreadyVotedException e) {

    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Já votou neste poll");

  } catch (DelegateNotFoundException e) {

    return ResponseEntity.status(HttpStatus.BAD_REQUEST)
        .body("Delegado associado não encontrado");

  } catch (PollNotFoundException e) {

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Votacao nao encontrada");

  } catch (CitizenNotFoundException e) {

    return ResponseEntity.status(HttpStatus.NOT_FOUND)
        .body("ID do cidadao nao existe no sistema");
  }
 }
}
```

Temos @PutMapping(“polls/{id}”) para atualizar o voto, sendo que executamos o método e a ResponseEntity envia uma resposta de sucesso se for sucedido, caso não existam componentes levanta exceções para os componentes em falta.

RestProjectController:

```
@RestController
@RequestMapping("api")
public class RestProjectController {

  @Autowired private CheckProjectService checkProjectService;

  @Autowired private SupportProjectService supportProjectService;
```

@RestController() para definir um rest api controller, e fazer com que o endereço tenha (/api) com @RequestMapping(“api”). É-se utilizado os Services de checkProject e supportProject para utilizar os seus métodos.

```
@GetMapping("/projects")
ResponseEntity<?> getProjects() {
  try {

    return ResponseEntity.ok().body(checkProjectService.CheckNonExpiredProjects());

  } catch (ProjectNotFoundException e) {

    return ResponseEntity.notFound().build();
  }
}
```

Temos o @GetMapping(“/projects”) que se executar com sucesso irá nos dar um Get com endereço /api/projects. São usadas Response Entities para termos respostas de sucesso (ou não) para a execução dos métodos.

```
@GetMapping("/projects/{id}")
ResponseEntity<?> getProject(@PathVariable Long id) {

  try {

    return ResponseEntity.ok(checkProjectService.getProject(id));

  } catch (ProjectNotFoundException e) {

    return ResponseEntity.notFound().build();
  }
}
```

Temos @GetMapping(“/polls/{id}”), tendo que @PathVariable id fará com que o id seja necessário para o url. Para além disso, acontece o mesmo que no getProjects(), temos a ResponseEntity que nos dá a resposta da execução do método com sucesso, caso contrário, em falta de algum componente, levanta exceções de cada componente.

```java
  @PutMapping("/projects/{id}")
  ResponseEntity<?> supportProject(@RequestParam Long citizenId, @PathVariable Long id) {

    try {

      supportProjectService.supportProject(citizenId, id);

      return ResponseEntity.ok().body("Project Apoiado!");

    } catch (CitizenNotFoundException e) {

      return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Cidadao nao encontrado");

    } catch (ProjectNotFoundException e) {

      return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Projeto nao encontrado");

    } catch (AlreadyVotedException e) {

      return ResponseEntity.status(HttpStatus.BAD_REQUEST)
          .body("Este cidadao ja apoiou este projeto");

    } catch (ProjectNotOnGoingException e) {

      return ResponseEntity.status(HttpStatus.BAD_REQUEST)
          .body("O projeto ja nao se encontra disponivel para apoiar");
    }
  }
}
```

Temos @PutMapping(“projects/{id}”) para apoiar o projeto, sendo que executamos o método e a ResponseEntity envia uma resposta de sucesso se for sucedido, caso não existam componentes levanta exceções para os componentes em falta.

WebDelegateController:

```java
@Controller
public class WebDelegateController {

  Logger logger = LoggerFactory.getLogger(WebPollController.class);

  public WebDelegateController() {
    super();
  }
```

Relativamente ao WebDelegateController, @Controller para definir um Controller. Temos super() no construtor para chamar o controller.

```java
@Autowired ChooseDelegateService chooseDelegateService;
@Autowired DelegateService delegateService;
@Autowired ThemeService themeService;
@Autowired CitizenService citizenService;

@GetMapping("/delegates/assign")
String getDelegates(final Model model) {

  model.addAttribute("users", citizenService.findAll());

  model.addAttribute("themes", themeService.getThemes());

  model.addAttribute("delegates", delegateService.getDelegates());

  return "assigndelegate";
}

@PostMapping("/delegates/assign")
String assignDelegates(
    @RequestParam("userid") Long userID,
    @RequestParam("themeid") Long themeID,
    @RequestParam("delegateid") Long delegateID) {

  try {

    chooseDelegateService.chooseDelegate(userID, delegateID, themeID);
    return "success";

  } catch (CitizenNotFoundException e) {

    return "error/citizen404";

  } catch (DelegateNotFoundException e) {

    return "error/citizen404";

  } catch (ThemeNotFoundException e) {

    return "error/theme404";

  } catch (SameIdException e) {

    return "error/sameid";
  }
 }
}
```

Temos @Autowired nos Services para fazer injeção de dependências nos serviços. No getDelegates temos um @GetMapping(“/delegates/assign”) para termos um URL com este endereço. Temos também um Model para atribuirmos utilizadores, temas e delegados.

Temos @PostMapping(“/delegates/assign/”) no método assignDelegates com @RequestParam de (“userid”), (“themeid”) e (“delegateid”) para cada parâmetro de id. Se for feito com sucesso, executa o método e retorna uma mensagem de sucesso, caso contrário levanta exceções e levanta uma mensagem de erro.

WebPollController:

```java
@Controller
public class WebPollController {

  @Autowired OngoingPollsService ongoingPollsService;
  @Autowired VoteProposalService voteProposalService;
  @Autowired PollService pollService;
  @Autowired CitizenService citizenService;

  Logger logger = LoggerFactory.getLogger(WebPollController.class);

  public WebPollController() {
    super();
  }

  @GetMapping("/polls")
  String getPolls(final Model model) {
    try {

      model.addAttribute("polls", ongoingPollsService.getOngoingPolls());

    } catch (PollNotFoundException e) {

      model.addAttribute("error", "No polls where found");
    }
    return "polls";
  }

  @GetMapping("/polls/{id}")
  String getPoll(final Model model, @PathVariable Long id) {

    model.addAttribute("users", citizenService.findAll());
    model.addAttribute("poll", pollService.findById(id).get());
    return "poll";
  }
```

Relativamente ao WebPollController, @Controller para definir um Controller. Temos super() no construtor para chamar o controller.

Temos @Autowired nos Services para fazer injeção de dependências nos serviços. No getPolls temos um @GetMapping(“/polls”) para termos um URL com este endereço. Se se tiver polls, adiciona-se os polls ao model, caso contrário levantava uma exceção e adicionava ao model uma mensagem de erro.

Temos @GetMapping(“/polls/{id}/”) no método getPoll com @PathVariable de id, para cada parâmetro de id. Se for feito com sucesso, executa o método e retorna uma mensagem de “poll” e adiciona utilizadores e polls ao model.

```java
@PostMapping("/polls/{id}")
String voteOnPoll(final Model model, @PathVariable Long id, @RequestParam("userid") Long userID) {

  try {

    // voteProposalService.voterChoice(citizenID, id, DelegateVotes, choice);
    model.addAttribute("poll", pollService.findById(id).get());
    model.addAttribute("citizen", citizenService.getCitizen(userID).get());
    model.addAttribute("delegateVote", voteProposalService.pickPolls(id, userID));
    return "vote";

  } catch (PollNotFoundException e) {

    return "error/poll404";

  } catch (CitizenNotFoundException e) {

    return "error/citizen404";

  } catch (NoDelegateAssignedException e) {

    model.addAttribute("poll", pollService.findById(id).get());
    model.addAttribute("citizen", citizenService.getCitizen(userID).get());
    model.addAttribute("delegateVote", null);
    return "vote";
  }
}
```

Neste método temos um modelo, @PathVariable id e @RequestParam(“userid”). Adicionamos polls, um citizen e votos de delegados ao model. Caso não existisse algum atributo levanta uma exceção e uma mensagem de erro, em exceção ao não existir voto de delegado, nesse caso carrega os dados para a página menos o voto do delegado, que é atribuído valor de null.

```java
@PostMapping("/polls/{id}/vote")
String voted(
    @PathVariable Long id,
    @RequestParam("citizenid") Long citizenID,
    @RequestParam("delegatevotes") boolean delegateVotes,
    @RequestParam("choice") boolean choice) {

  try {

    voteProposalService.voterChoice(citizenID, id, delegateVotes, choice);
    return "redirect:/polls/" + id;

  } catch (AssignedDelegateNullVoteException e) {

    return "error/badrequest";

  } catch (HasAlreadyVotedException e) {

    return "error/voted";

  } catch (DelegateNotFoundException e) {

    return "error/citizen404";

  } catch (PollNotFoundException e) {

    return "error/poll404";

  } catch (CitizenNotFoundException e) {

    return "error/citizen404";
  }
 }
}
```

Neste método temos @PostMapping para atualizar o poll, que @PathVariable id fará com que o id seja necessário para o url, e o @RequestParam CitizenId, delegateVotes e choice também fique bound ao pedido de URL.

Caso falte algum componente, levanta exceção a cada caso e retorna uma mensagem de erro, caso contrário, executa o método voterChoice e retorna um endereço de retorno com o id.

WebProjectController:

```
@Controller
public class WebProjectController {

  Logger logger = LoggerFactory.getLogger(WebPollController.class);

  @Autowired CheckProjectService checkProjectService;
  @Autowired SupportProjectService supportProjectService;
  @Autowired PresentProjectService presentProjectService;
  @Autowired ThemeService themeService;
  @Autowired CitizenService citizenService;
  @Autowired DelegateService delegateService;

  public WebProjectController() {
    super();
  }

  @GetMapping("/projects")
  String getProjects(final Model model) {
    try {

      model.addAttribute("projects", checkProjectService.CheckNonExpiredProjects());

    } catch (ProjectNotFoundException e) {

      model.addAttribute("error", "Projects Not Found");
    }

    return "projects";
  }

  @GetMapping("/projects/{id}")
  String getProject(final Model model, @PathVariable Long id) {

    try {

      model.addAttribute("users", citizenService.findAll());

      model.addAttribute("project", checkProjectService.getProject(id));

    } catch (ProjectNotFoundException e) {

      return "error/project404";
    }

    return "project";
  }
```

Relativamente ao WebProjectController, @Controller para definir um Controller. Temos super() no construtor para chamar o controller. Temos @Autowired nos Services para fazer injeção de dependências nos serviços. No getProjects temos um @GetMapping(“/projects”) para termos um URL com este endereço. Temos também um Model para atribuirmos projetos, ou uma mensagem de erro caso não exista projeto, é também levantada uma exceção.

Temos @GetMapping(“/projects/{id}”), tendo que @PathVariable id fará com que o id seja necessário para o url. Para além disso, acontece o mesmo que no getProjects, atribuímos utilizadores e projeto ao modelo da página, caso contrário caso falte um Projeto, levanta uma exceção e retorna mensagem de erro.

```
@PostMapping("/projects/{id}/support")
String supportProject(
    final Model model, @PathVariable Long id, @RequestParam("userid") Long userID) {

  try {

    supportProjectService.supportProject(userID, id);
    model.addAttribute("project", checkProjectService.getProject(id));
    model.addAttribute("mensagem_de_sucesso", "projeto apoiado com sucesso");
    return "redirect:/projects/" + id;

  } catch (CitizenNotFoundException e) {

    return "error/citizen404";

  } catch (ProjectNotFoundException e) {

    return "error/project404";

  } catch (AlreadyVotedException e) {

    return "error/voted";

  } catch (ProjectNotOnGoingException e) {

    return "error/notongoing";
  }
}

@GetMapping("/project/new")
public String createProject(final Model model) {

  model.addAttribute("users", delegateService.getDelegates());
  model.addAttribute("temas", themeService.getThemes());
  return "presentproject";
}
```

Neste método temos @PostMapping para atualizar o projeto, que @PathVariable id fará com que o id seja necessário para o url, e o @RequestParam userID também fique bound ao pedido de URL. Caso falte algum componente, levanta uma exceção e retorna um endereço de erro específico. Caso corra tudo bem, então adiciona ao modelo o projeto dado um id, uma mensagem de sucesso e retorna o endereço de retorno.

```java
@PostMapping("/project/new")
public String createProjectAction(
    @RequestParam("userid") Long userID,
    @RequestParam("titulo") String titulo,
    @RequestParam("descricao") String descricao,
    @RequestParam("data_validade") LocalDateTime dataValidade,
    @RequestParam("tema") String tema) {

  try {

    Long id = presentProjectService.presentProject(userID, titulo, descricao, dataValidade, tema);
    return "redirect:/projects/" + id;

  } catch (DelegateNotFoundException e) {

    return "error/citizen404";

  } catch (ThemeNotFoundException e) {

    return "error/theme404";

  } catch (InvalidTitleTextException e) {

    return "error/badrequest";

  } catch (IOException e) {

    return "error/badrequest";

  } catch (InvalidDateException e) {

    return "error/badrequest";
  }
}
```

Neste método, temos @PostMapping para atualizar e criar um projeto, @RequestParam nos vários parâmetros para ficarem bound ao url. Caso não existam estes parâmetros, levanta exceção referente ao parâmetro em falta e retorna o endereço de erro. Caso ocorra com sucesso, cria um id com o método presentProject e retorna o endereço do novo projeto criado.

```html
<!DOCTYPE html>
<html th:fragment="layout (content)" xmlns:th="http://www.thymeleaf.org">

<head>
    <title>Democracia2</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@picocss/pico@1/css/pico.min.css">
    <style>
        #container {
            margin: auto;
            max-width: 500px;
        }

        th {
            font-weight: bold;
        }
    </style>
</head>

<body>

    <div class="container">
        <!--/* Standard layout can be mixed with Layout Dialect */-->
        <div layout:fragment="header">
            <h1>Democracia 2.0</h1>

            <ol>
                <li><a th:href="@{/projects}">Projects</a></li>
                <li><a th:href="@{/polls}">Polls</a></li>
                <li><a th:href="@{/project/new}">Present Project</a></li>
                <li><a th:href="@{/delegates/assign}">Assign Delegate</a></li>
            </ol>
        </div>

        <div th:replace="${content}">
            <!-- ============================================================================ -->
            <!-- This content is only used for static prototyping purposes (natural templates)-->
            <!-- and is therefore entirely optional, as this markup fragment will be included -->
            <!-- from "fragments/header.html" at runtime.                                     -->
            <!-- ============================================================================ -->
            <h2>Static content for prototyping purposes only</h2>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                Praesent scelerisque neque neque, ac elementum quam dignissim interdum.
                Phasellus et placerat elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                Praesent scelerisque neque neque, ac elementum quam dignissim interdum.
                Phasellus et placerat elit.
            </p>
        </div>
    </div>
</body>

</html>
```

Relativamente ao layout, foi utilizado o layout disponibilizado pelos Professores na página na disciplina, e agradecemos a disponibilidade. Foi utilizado este layout para as páginas da web.

```html
<!DOCTYPE html>
<html th:replace="layout :: layout(~{::section})" xmlns:th="http://www.thymeleaf.org">

<body>
    <section>

        <h2>User and Delegate cannot be the same!<h2>

    </section>
</body>

</html>
```

Foram feitas várias templates de html de erro como a acima demonstrada (404, badrequest, citizen404, notongoing, poll404, project404, sameid, theme404 e vote). Nestas templates, simplesmente a mensagem de erro altera.

```
<!DOCTYPE html>
<html th:replace="~{layout :: layout(~{::section})}">

<body>
    <section>
        <div>
            <h2 th:text="${project.title}"></h2>
            <p th:text="'Project text: ' + ${project.text}"></p>
            <p th:text="'Signatures : ' + ${project.signatures.size()}"></p>
            <p th:text="'Is ongoing : ' + ${project.ongoing}"></p>
            <p th:text="'Assigned Delegate : ' + ${project.delegate.name}"></p>
            <p th:text="'Related theme : ' + ${project.theme.name}"></p>
            <p th:text="'End Date: ' + ${project.d}"></p>
        </div>
        <form th:action="@{/projects/{id}/support(id=${project.id})}" method="post">

            <label for="userid">Current User:</label><br>
                <select id="userid" name="userid" required>
                <option value="">Select a User</option>
                <option th:each="user : ${users}" th:value="${user.id}" th:text="${user.name}"></option>
            </select><br><br>

            <button type="submit" class="btn btn-primary">Support</button>
        </form>

        <div th:if="${mensagem_de_sucesso}">
            <p th:text="${mensagem_de_sucesso}"></p>
        </th:if>
    </section>
</body>

</html>
```

E o layout foi também utilizado para os vários componentes (project, projects, index, assigndelegate, poll, polls, presentproject, success, vote).