UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Ferramenta para gerenciamento de projetos, visando a aplicação em Sistemas de Sistemas

Luis Eduardo Prado Santini

São Carlos – SP

Ferramenta para gerenciamento de projetos, visando a aplicação em Sistemas de Sistemas

Luis Eduardo Prado Santini

Orientadora: **Prof.ª Dr.ª Rosana T. Vaccare Braga**

Coorientador: **Msc. Iohan G. Vargas**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.
Área de Concentração: Engenharia de Software

USP – São Carlos
Novembro de 2022

Santini, Luis Eduardo Prado
Ferramenta para gerenciamento de projetos, visando a aplicação em Sistemas de Sistemas / Luis Eduardo Prado Santini. - São Carlos - SP, 2022.
63 p.; 29,7 cm.

Orientadora: Rosana T. Vaccare Braga.
Coorientador: Iohan G. Vargas.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2022.

1. Sistema de Sistemas. 2. gerenciamento de projetos. 3. aplicação Web. 4. *framework* de organização. 5. desenvolvimento *front-end*. 6. desenvolvimento *back-end*. I. Braga, Rosana T. Vaccare. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

Este trabalho é dedicado aos meus pais,
Rosane e Eduardo, que me proporcionaram tudo de mais precioso na vida, à meu irmão, Luis Henrique, pela amizade incondicional desde sempre e a minha companheira Renata, por todas alegrias e apoio nessa jornada.

AGRADECIMENTOS

Agradeço aos meus pais, Rosane e Eduardo, pelos valores que me ensinaram desde criança, como a importância da educação na vida de uma pessoa, e por terem me proporcionado tudo de melhor desde sempre.

Agradeço à meu irmão, Luis Henrique, pela amizade incondicional.

Agradeço à minha companheira, Renata, por sempre me apoiar e incentivar.

Agradeço aos meus familiares, por sempre me apoiarem nas minhas decisões.

Agradeço aos meus amigos, por estarem juntos comigo nos melhores e piores momentos.

Por fim, agradeço também à professora Rosana Teresinha Vaccare Braga, por ter aceitado me orientar neste trabalho e por toda atenção e suporte prestados. Bem como agradeço ao Iohan Gonçalves Vargas, por ter aceitado ser coorientador nesse trabalho, agregando com sua atenção e suporte.

“Quando vires um homem bom, tenta imitá-lo;
quando vires um homem mau, examina-te a ti mesmo. ”
(Confúcio)

RESUMO

SANTINI, L. E. P.. **Ferramenta para gerenciamento de projetos, visando a aplicação em Sistemas de Sistemas**. 2022. 63 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

O crescimento da presença de projetos de larga escala, sob a denominação de Sistema de Sistemas, trouxe consigo novas dificuldades, especialmente no âmbito do gerenciamento desses projetos. Suas características diferem de sistemas tradicionais, requerendo novas formas de organização, assim como novas ferramentas auxiliares. Diante disso, está sendo desenvolvido (em um projeto maior do qual esse faz parte) um *framework* para gerenciamento de Sistemas de Sistemas. Aplicando diversos processos de desenvolvimento de projeto de software e metodologias ágeis, desenvolveu-se, neste trabalho, uma ferramenta no formato de aplicação Web, cuja função principal é auxiliar gerentes de SoS nas tarefas de organizar e definir os papeis existentes no SoS e sistemas constituintes, com suas respectivas responsabilidades, permitindo sua alocação em diferentes equipes do projeto. Além disso, também foi desenvolvida a API responsável pela comunicação com o banco de dados e fornecimento de recursos e serviços à aplicação Web. Vale destacar que a ferramenta desenvolvida neste trabalho teve acompanhamento por especialistas da área a fim de adequá-la às necessidades do domínio da aplicação.

Palavras-chave: Sistema de Sistemas, gerenciamento de projetos, aplicação Web,*framework* de organização, desenvolvimento *front-end*, desenvolvimento *back-end*.

ABSTRACT

SANTINI, L. E. P.. **Ferramenta para gerenciamento de projetos, visando a aplicação em Sistemas de Sistemas**. 2022. 63 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

The increasing number of large scale projects, called System of Systems (SoS), brought with it new challenges, concerning specially the project management aspect. Its characteristics differ from traditional systems, requiring new methods of organization, as well as new auxiliary tools. In this context, a framework to manage Sos is being developed, in a larger project in which this one is a part of. In this project, a tool was developed in the form of a Web application, using a well-defined software development process based on agile methods. Its main function is to assist SoS managers in the task of organizing and defining the existing roles in the SoS and constituent systems, with their respective responsibilities, allowing their allocation to different project teams. The tool presents a Web application to interface with the end users and an API, which is responsible for the database communication while providing resources and services for the Web application. The tool developed in this project had the supervision of field specialists to make sure it is adequate to the application domain.

Key-words: System of Systems, project management, Web application, organization framework, front-end development, back-end development.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

Considerando a abundância de diferentes sistemas arraigados em nossa civilização, tornou-se necessário ser capaz de utilizar sistemas novos e já existentes de maneira integrada e eficiente. Assim, emergiu-se a abordagem arquitetural de multi-sistemas, chamada Sistemas de Sistemas (ou SoS, do inglês *Systems of Systems*). Os SoS estão por toda parte, desde segurança domiciliar, integrando diversos dispositivos, até empresariais, que integram folhas de pagamento e contabilidade, bem como sistemas de gerenciamento de crises, como por exemplo incêndios em larga escala ou até mesmo pandemias (LANE, 2013; MAIER, 1998).

SoSs tem sido desenvolvidos e disponibilizados há algumas décadas e conforme foram se tornando mais comuns, engenheiros e pesquisadores da área começaram a diferenciá-los como uma classe específica de sistema, resultando nessa classificação relativamente recente. Observando a literatura relacionada, percebe-se que o termo possui diversos significados, dependendo do interlocutor, variando desde sistemas de integração multi-empresarial, arquitetura multissistêmica planejada, até sistemas cuja arquitetura evolui ao longo do tempo de acordo com recursos e necessidades. Também podem ser definidos como sistemas com comportamento emergente que são capazes de proporcionar funcionalidades que vão além das capacidades de seus constituintes. Em razão dessa natureza dinâmica, é difícil encontrar uma descrição clara, porém, as características comuns e singulares desses sistemas são a independência operacional, que consiste na capacidade dos sistemas constituintes realizarem tarefas que sejam importantes tanto no contexto do SoS quanto fora dele e, a independência gerencial, que faz alusão à capacidade de gerenciamento e manutenção da maioria dos sistemas constituintes de acordo com seus próprios interesses (LANE, 2013).

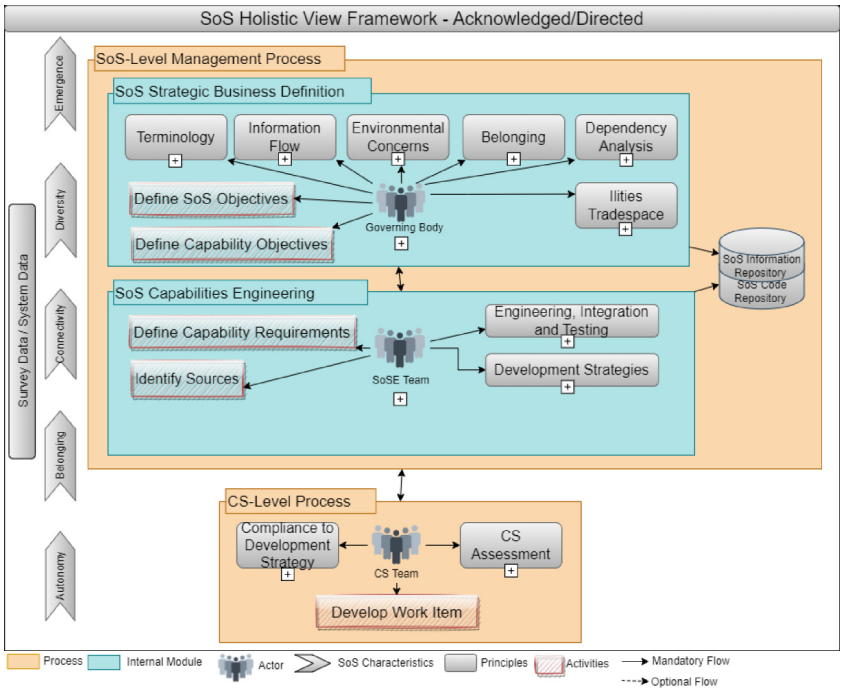
Da mesma forma que a complexidade dos problemas resolvidos por esses sistemas aumentou, o corpo de conhecimento necessário para seu desenvolvimento, integração e gerenciamento não conseguiu acompanhar tamanha ascensão, requerendo que times multidisciplinares de engenharia e gerência sejam capazes de elaborar padrões para que as diferentes áreas possam integrar suas contribuições. Tal situação implica em diversas dificuldades advindas de tamanho, escopo e variedade de pessoas envolvidas, bem como níveis inadequados de comunicação e suporte, falta de visão holística, visões conflitantes e o impacto de micro-comportamentos dos constituintes no desempenho do projeto. Métodos e ferramentas que sejam capazes de auxiliar

nesse processo são de suma importância para que o projeto e desenvolvimento de um SoS prossigam de maneira bem sucedida (LANE, 2010).

1.2 Motivação

A área de gerenciamento de sistemas tradicionais é bem estabelecida, há muito conhecimento sedimentado em guias como o PMBOK (PMI, 2017), além de muitas ferramentas disponíveis que dão suporte aos gerentes de projeto. Por outro lado, as boas práticas e definições sobre como gerenciar um SoS são um tema de pesquisa em aberto, especialmente na definição das responsabilidades e níveis de autoridade para executar tarefas gerenciais. Apesar de poucas, existem abordagens e frameworks que buscam descrever as práticas de gerenciamento de SoSs. Dentre os frameworks propostos há um feito em um trabalho de doutorado em andamento, realizado pelo (VARGAS, 2022), que foi a base para a estruturação deste projeto, apresentando duas entidades de suma importância, o *Governing Body* e o SoSE (*System of Systems Engineering*). A primeira é responsável principalmente pela área de organização do SoS e é composta por stakeholders que sejam capazes de compreender e modelar as dependências entre sistemas, de forma a determinar as capacidades do SoS bem como sedimentar padrões, gerenciar recursos e mudanças. A segunda é mais focada em engenheria de sitemas, aspectos de integração, e desenvolvimento das funcionalidades do SoS. O diagrama desse *framework* pode ser visto na Figura 1.

Figura 1 – Diagrama do *framework* de organização de um SoS



Fonte: (VARGAS, 2022)

As ferramentas existentes hoje, como por exemplo o Jira[1], Notion[2], ClickUp[3] e Trello[4], são focadas em projetos de sistemas tradicionais (VARGAS; GOTTARDI; BRAGA, 2016.). Não há evidencias de ferramentas que apoiem a gerência de equipes e responsabilidades, ou seja, ferramentas que fornecem uma visão holística do projeto e permitem que sejam vistos com clareza os papeis envolvidos, pessoas, responsabilidades e que possibilite uma interação.

1.3 Objetivos

Este trabalho tem por objetivo auxiliar gerentes de SoS em seu trabalho, facilitando as tarefas de organizar e definir os papeis existentes e suas respectivas responsabilidades, bem como sua alocação em diferentes equipes do projeto. Para isso, foi desenvolvida uma aplicação *Web* e um sistema de *back-end*, que trabalham em conjunto para compor a ferramenta de gestão.

1.4 Método e Resultados Esperados

De forma a atingir os objetivos citados, realizou-se um estudo sobre a área de SoS, para entender o estado da arte do gerenciamento desses projetos, com foco em suas maiores dificuldades. Periodicamente, foram realizadas reuniões com a orientadora e o doutorando que realiza pesquisa nesta área, de forma a analisar dentre as ferramentas de gerência de projeto disponíveis hoje, quais eram seus benefícios e funcionalidades úteis no contexto deste projeto de formatura.

Também foram realizadas reuniões para determinar as metas da aplicação a ser desenvolvida, de maneira a se adequar às necessidades da gerência de SoS, resultando em um documento de requisitos, documentos de projeto de desenvolvimento de software, como casos de uso e modelagem conceitual.

O desenvolvimento da ferramenta foi acompanhado de perto, com discussão do andamento, aprimoramento e refino do projeto, de maneira inspirada em metodologias ágeis como o SCRUM(DYBA; DINGSOYR, 2008).

A partir de então, foi desenvolvida uma API (*Application Programming Interface*), que é responsável por gerenciar as lógicas de serviço e expor as funcionalidades que são consumidas pela aplicação Web, bem como comunicar com o banco de dados.

A aplicação Web que foi desenvolvida fornece a visualização dos projetos e suas informações, registradas pelos usuários, permitindo a organização interna de cada projeto em diferentes times, com cargos específicos cujas responsabilidades são atribuídas. Além disso,

[1] https://www.atlassian.com/software/jira
[2] https://www.notion.so
[3] https://clickup.com
[4] https://trello.com

foram definidas as atividades no contexto de SoS, que podem ser atribuídas a *stakeholders* envolvidos no gerenciamento do SoS.

É esperado que a aplicação possa ser utilizada por um gerente de SoS, como ferramenta auxiliar para a organização e gerenciamento de projetos de SoS, fornecendo portanto um *framework* adequado à esse contexto.

1.5 Organização

Os métodos, tecnologias, padrões e técnicas utilizados no desenvolvimento deste trabalho, assim como os trabalhos da literatura relacionada são apresentados no Capítulo 2. Em sequência, no Capítulo 3, há a descrição do projeto desenvolvido, com implementações. Os resultados obtidos se encontram no Capítulo 4. Finalmente, as contribuições, melhorias e trabalhos futuros desse projeto, bem como seu relacionamento com o curso de graduação são encontrados no Capítulo 5.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Neste capítulo serão apresentados os principais conceitos utilizados durante o desenvolvimento deste trabalho, bem como os trabalhos relacionados.

2.2 Conceitos e Técnicas

2.2.1 Sistema de Sistemas

SoS pode ser definido como a integração de múltiplos sistemas cuja finalidade é a resolução de problemas em larga escala. A discussão sobre essa classe de sistemas existe há muito tempo, e suas principais características são (MAIER, 1998):

- Independência operacional: os sistemas que constituem o SoS (do inglês *Constituent Systems* ou CS) devem operar de maneira independente a cumprir seu objetivo, mesmo fora do SoS.
- Independência gerencial: cada sistema constituinte deve ser gerenciado de maneira independente aos outros que compõe o SoS;
- Distribuição geográfica: os sistemas constituintes devem operar mesmo que estejam em locais distintos.
- Comportamento emergente: a interação dos CSs resulta em um comportamento inalcançável por eles independentemente; e
- Desenvolvimento evolucionário: o SoS nunca está completo, seus objetivos e funcionalidades estão em constante mudança.

Além de sua caracterização principal, os SoS são comumente segmentados de acordo com a presença ou não de uma autoridade central, nível de controle gerencial e colaboração dos sistemas constituintes. Os quatro tipos de mais comuns são (MAIER, 1998):

- Reconhecido: os objetivos do SoS são reconhecidos, com gerente e recursos designados, mantendo a independência dos CS's;
- Direcionado: Os objetivos para os quais o SoS é desenvolvido são específicos, os CS's são operados de forma independente mas controlados por uma autoridade global, tendo evolução controlada;
- Colaborativo: Existe uma autoridade central, porém sem poder coercitivo de executar o sistema. A colaboração pelos CS's é voluntária, não havendo equipe de engenharia de SoS para orientar ou gerenciar atividades do SoS ou CS's; e
- Virtual: Não há autoridade central, nem propósito definido, os sistemas constituintes podem não ser reconhecidos.

2.2.2 Metodologias Ágeis

Os métodos para desenvolvimento ágil de software consistem em um conjunto de práticas para o desenvolvimento de software criadas por praticantes experientes dessa área. Podem ser considerados uma alternativa aos métodos tradicionais que admitem que os problemas podem ser completamente especificados e daí obtidas soluções previsíveis, resultando em planejamento extensivo (DYBA; DINGSOYR, 2008). Diferente da abordagem tradicional, os processos ágeis tem em seu cerne a consideração da imprevisibilidade do mundo, cujos quatro valores principais são (DYBA; DINGSOYR, 2008):

- Priorizar indivíduos e interações ao invés de processos e ferramentas;
- Software funcional ao invés de documentação compreensível;
- Colaboração do cliente ao invés de negociações contratuais; e
- Responder as mudanças ao invés de seguir um planejamento.

2.2.3 Desenvolvimento front-end e back-end

O desenvolvimento de aplicações é em geral dividido em dois componentes principais, o *front-end* e o *back-end*. O primeiro engloba o componente responsável pela interação com o usuário final, sendo composto pela interface de usuário, com menus, botões, fluxos de navegação, formulários, entre outros (ABDULLAH; ZEKI, 2014). A abordagem utilizada para seu desenvolvimento neste trabalho consiste numa tríade de tecnologias, o HTML (*HyperText Markup Language*), que é uma linguagem de marcação, ou seja, estruturação dos elementos da página, o CSS (*Cascading Style Sheets*) responsável pela maneira que tais elementos são renderizados, lidando com a estética da página, e o *JavaScript*, que consiste numa linguagem de programação

que nesse contexto é empregada para execução de lógicas e comportamentos da página, como por exemplo a comunicação com recursos do *back-end*.

Tais tecnologias são disponíveis em diversos *frameworks* que se propõe a resolver esse mesmo problema, como por exemplo *Angular*, *React*, *Flutter*, entre outros. Para este trabalho foi utilizado o *framework* Angular devido a experiência prévia de desenvolvimento, além de boas funcionalidades que auxiliam o desenvolvedor, como por exemplo a utilização de *TypeScript* ao invés de *JavaScript*, com uma estrutura modular em sua abordagem de desenvolvimento e possui suporte de longo prazo por uma empresa consolidada, o Google[1], fornecendo uma interface de usuário declarativa e consistência de código.

O *back-end*, por outro lado, consiste em um componente transparente ao usuário, uma característica vital de sistemas distribuídos, sendo totalmente automatizado para lidar com as requisições recebidas do *front-end*. Ele é constituído, geralmente, por um banco de dados e um servidor de aplicações (ABDULLAH; ZEKI, 2014). Esse componente pode ser construído com diversas linguagens, como *PHP*, *Ruby*, *Java*, *Python* e até mesmo *JavaScript*, possuindo diferentes *frameworks* para auxiliar em seu desenvolvimento, como *Spring* ou *Ruby on Rails*.

Escolheu-se utilizar o *Spring framework*, em *Java*, devido a experiência prévia de desenvolvimento com a tecnologia, sem a necessidade do aprendizado de uma nova tecnologia e por ser um sistema feito com desenvolvimento de APIs REST.[2]

2.2.4 Spring Framework e Angular

O *Spring* é um *framework* que provê um modelo de programação e configuração para aplicações modernas feitas em Java, cujo apelo principal é sua robustez de infra-estrutura, para que as equipes de desenvolvimento sejam capazes de focar no nível de lógica de serviço, reduzindo o tempo gasto com configurações e acoplamento a determinados ambientes de execução [3]. No contexto deste trabalho, foi implementada uma API REST, ou seja, uma API que segue os padrões arquiteturais de acordo com as diretrizes REST (FIELDING, 2000), baseada na comunicação cliente-servidor sem armazenamento do estados, com mensagens auto-descritivas que formam uma interface de comunicação padronizada e uniforme.

O *Angular* é um *framework* e ambiente de desenvolvimento focado na construção de aplicações Web de página única, se beneficiando do fato de se basear em *TypeScript*, um *superset* de *JavaScript* fortemente tipado, de maneira a detectar diversos erros durante o desenvolvimento que outrora seriam somente percebidos em produção. É focada na organização da aplicação em componentes que interagem, com o intuito de modularizar a estrutura do código e trazer uma experiência mais amigável ao desenvolvedor [4].

[1] https://about.google
[2] https://spring.io/web-applications
[3] https://spring.io/projects/spring-framework
[4] https://angular.io/guide/what-is-angular

2.2.5 Object–relational mapping (ORM)

A técnica de programação chamada de *Object-relational mapping* (ORM) (em português mapeamento objeto-relacional) consiste na conversão, usando linguagens de programação orientadas a objetos, dos dados de sistemas distintos, que são a priori incompatíveis, devida a discrepâncias em seus paradigmas, técnicas, conceitos e culturas (TORRES *et al.*, 2017). Tais diferenças são inerentes ao processo de integração desses sistemas, no contexto da integração entre paradigmas orientados a objetos e bancos de dados relacionais. Essa discrepância é conhecida como IMP (*impedance mismatch problem*)(IRELAND *et al.*, 2009).

O ORM é uma solução para o problema de IMP, fornecendo uma abstração do processo de mapeamento dos objetos no contexto da programação orientada a objetos para seus correspondentes registros em bases de dados relacionais. Dessa forma, é possível realizar o acesso às informações persistidas em bancos de dados sem a necessidade da construção de consultas SQL, não se atentando aos detalhes do dialeto utilizado em determinado sistema gerenciador de bancos de dados, tornando a implementação de certa forma agnóstica ao dialeto SQL. Assim, essa solução reduz a quantidade de código necessário, reduzindo custos de desenvolvimento, bem como prazos de entrega(CHEN *et al.*, 2014)

Os conceitos de ORM tem sido amplamente aplicados na prática, por diferentes *frameworks* de orientação a objetos, dado que a redução da complexidade da implementação provida é um diferencial muito significativo (JOHNSON, 2005). Alguns frameworks que implementam ORMs são o *TypeORM*[5], *Prisma*[6], *Sequelize*[7] e *Hibernate*[8], que está presente no *Spring framework* e foi utilizado neste projeto.

2.2.6 Padrão arquitetural Model-View-Controller (MVC)

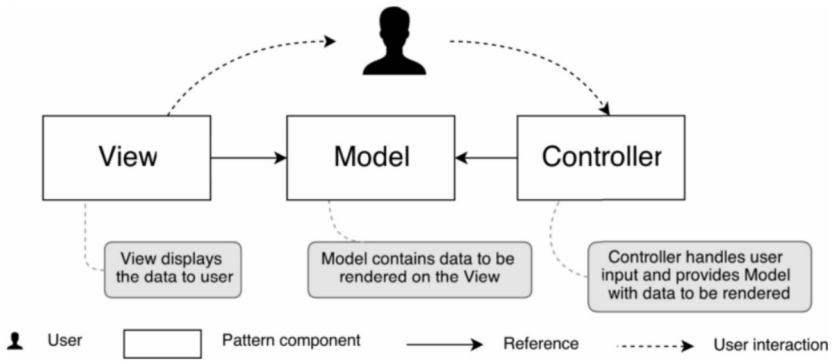
O problema de como integrar a interface de usuários com o domínio da aplicação se tornou particularmente relevante quando as interfaces gráficas surgiram nos anos 80. Juntamente com a evolução das linguagens de programação, surgiram novos padrões arquiteturais, entre eles o MVC. Sua principal ideia é a separação de preocupações, que no caso do design de interfaces significa separar o domínio da aplicação da interface propriamente dita, de forma que o domínio não depende, nem tem consciência da existência da interface do usuário. Tal abordagem levou a uma distribuição eficaz do trabalho em diferentes camadas entre os desenvolvedores(SYROMIATNIKOV; WEYNS, 2014).

O MVC separa a lógica de apresentação dos dados da lógica de domínio das aplicações, por meio de três componentes com responsabilidades distintas: o Modelo (*Model*) armazena os dados e é responsável pelas lógicas de serviço, o *Controller* é responsável pela gerência das

[5] https://typeorm.io/
[6] https://www.prisma.io/
[7] https://sequelize.org/
[8] https://hibernate.org/

ações dos usuários, e a *View* é responsável por gerar o layout da página. Tal modelo arquitetural se encaixa inerentemente em aplicações Web, dado que as páginas são visualizadas no lado do cliente, enquanto a lógica que gerencia as ações dos usuários ocorre no lado do servidor. No caso do padrão Web MVC, a diferença é que as lógicas de manipulação dos dados ocorrem no âmbito do Controller e o Model está somente preocupado com o armazenamento dos dados (SYROMIATNIKOV; WEYNS, 2014).

Figura 2 – Diagrama do padrão arquitetural Web MVC



Fonte: (SYROMIATNIKOV; WEYNS, 2014)

Este trabalho se encaixa no padrão arquitetural Web MVC, abstraindo os componentes *Model* e *Controller* no âmbito da API REST construída por meio do framework Spring, enquanto o componente *View* é delimitado pela aplicação Angular.

2.3 Trabalhos Relacionados

Sob a óptica do gerenciamento de SoS, algumas ferramentas de gerenciamento de projetos disponíveis no mercado foram avaliadas, de forma a se determinar como elas atendiam às necessidades dessa aplicação. Quatro ferramentas distintas foram analisadas, o Trello, Notion, ClickUp e Jira. A primeira é focada no framework de gerenciamento conhecido como Kanban, geralmente utilizado em desenvolvimento ágil de software, cuja visualização do estado do projeto foca em determinar pontos de gargalo e a partir daí remanejar os esforços da equipe envolvida de maneira adequada (TREGUBOV; LANE, 2015).

Tal sistema possui diversas funcionalidades úteis para monitoramento das atividades em andamento e de quais indivíduos são responsáveis por elas, porém, é mais adequada na gerência de projetos com escopos menores, não possuindo uma maneira adequada de lidar com diversas equipes intercomunicantes.

O Notion[9], assim como o Trello[10], são focados em uma estrutura de Kanban, possuindo

[9] https://www.notion.so
[10] https://trello.com

maior flexibilidade, permitindo a criação de *templates* que podem ser customizados de maneira a se adequar melhor ao projeto, porém sofre das mesmas dificuldades já apresentadas, em que a gerência de diferentes equipes não é bem apoiada, nem a integração com outras ferramentas de desenvolvimento de projetos.

Também analisou-se a ferramenta chamada ClickUp[11], cujo foco é em fornecer um conjunto de funcionalidades mais genéricas e customizáveis, com intuito substituir as diferentes ferramentas de gerenciamento em uma única. No caso, há o fornecimento de funcionalidades compartimentalizadas, possuindo suporte à criação e gerência de diversas equipes cujas responsabilidades são distintas, de acordo com suas necessidades. Além disso, a ferramenta proporciona um conjunto de visões variado, permitindo a visualização do projeto como um todo por diferentes pontos de vista, desde mapas mentais, gráficos de Gantt, até o próprio Kanban.

Assim como o ClickUp, o Jira[12] é outra ferramenta robusta, que possui diversas visões e também é capaz de gerar documentações e relatórios sob demanda, porém, é mais focada em times de desenvolvimento ágil e no processo SCRUM, não possuindo a generalidade necessária para se adequar a um SoS, além de ser a mais custosa financeiramente.

Em síntese, as ferramentas consideradas são voltadas para projetos de sistemas únicos, não permitindo organização tal qual *frameworks* de SoS, definindo equipes que possuem papeis específicos que cumprem determinadas responsabilidades, assim como falta a visão holística das atividades de um SoS.

2.4 Considerações Finais

Neste capítulo, foi apresentada a fundamentação teórica dos conceitos utilizados no projeto e desenvolvimento do trabalho, bem como expostos os trabalhos relacionados às ferramentas existentes no mercado bem como os pontos positivos e limitações de cada ferramenta.

[11] https://clickup.com
[12] https://www.atlassian.com/software/jira

Capítulo 3

DESENVOLVIMENTO

3.1 Considerações Iniciais

Neste capítulo são detalhadas todas as etapas de desenvolvimento do projeto da ferramenta de software proposta, desde a sua modelagem inicial até a sua implementação utilizando as tecnologias já apresentadas. Também são descritas as dificuldades encontradas.

3.2 Visão Geral do Projeto

O contexto no qual se insere este projeto de formatura é o desenvolvimento de um sistema de apoio ao gerenciamento de projetos focado no âmbito de SoS, de maneira a complementar um trabalho de doutorado em andamento e contribuir com uma ferramenta que empregue um framework de gerenciamento de SoS.

O sistema, chamado *System of Systems Tool*, ou *SoS Tool* para fácil referência, consiste em uma aplicação Web, cuja intenção é ser acessada pelos gerentes de projetos de SoS, permitindo a definição de projetos em que integram diferentes equipes, cada uma com seus respectivos papéis associados. Esses que por sua vez possuem responsabilidades determinadas e são associados a determinados membros do projeto. Além disso, dentro de determinado projeto pode-se definir atividades a serem cumpridas, cada uma com seus colaboradores, e seu atual status de progresso.

O *SoS Tool* teve sua modelagem feita a partir da arquitetura *Web-Model-View-Controller*, uma variação do MVC tradicional apresentado na Seção 2.2.6. Ele é composto por um *back-end* (REST API) cujos recursos e funcionalidade são acessados por uma aplicação *front-end* (Web-app). O *back-end* foi desenvolvido por meio do *Spring framework*, enquanto o *front-end* foi feito em *Angular*.

A responsabilidade da API é de fornecer recursos e serviços, além de se comunicar com o banco de dados. Por exemplo, para realizar a listagem dos membros participantes de um projeto, a aplicação *front-end* realiza requisições HTTP para a API e recebe na resposta as informações desejadas, permitindo que aplicações distintas acessem esses recursos simultaneamente, como uma aplicação Web e um aplicativo para celulares.

Outra característica importante é a funcionalidade Spring Data JDBC, que é uma solução alinhada à filosofia *Domain-Driven Design*. Essa filosofia ajusta o foco do projeto ao domínio

da aplicação, baseado no *feedback* de especialistas no assunto para que as regras de negócio sejam claras. Fornece assim uma maneira intuitiva e com alto-nível de abstração para o design de aplicações. Por exemplo, ao carregar uma entidade do domínio da aplicação em memória, como as informações de um usuário, o código SQL responsável por tal ação é definido pelo framework automaticamente, reduzindo o fardo sob o programador.

A linguagem Java foi escolhida pois é uma linguagem fortemente tipada, orientada a objetos e que proporciona uma boa experiência de desenvolvimento, entregando código de fácil leitura e compreensão sem abrir mão de um desempenho satisfatório e independente de plataforma, além de ser uma linguagem já validada no mercado há décadas e amplamente adotada (PARVEEN; FATIMA, 2016).

O *front-end*, por outro lado, foi implementado utilizando o framework Angular, um framework desenvolvido pelo Google, baseado em componentes que possui ampla adoção no mercado. Por trás do Angular existe a tríade de tecnologias usuais do desenvolvimento Web, no caso HTML, CSS e TypeScript, fornecendo uma programação fortemente tipada, roteamento de páginas intuitivo e modularização simplificada. Ambas as aplicações foram desenvolvidas integralmente neste projeto de formatura.

3.3 Atividades Realizadas

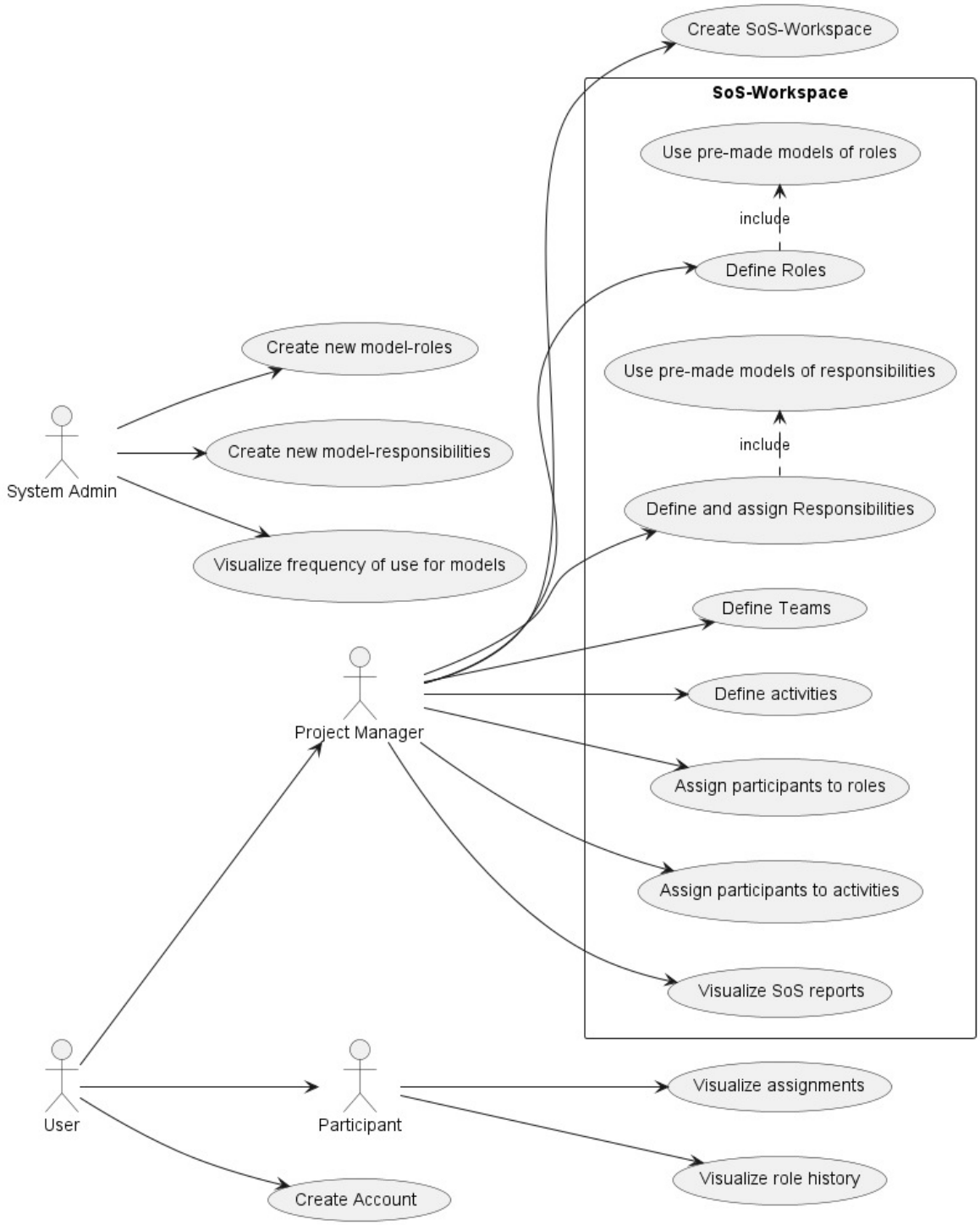
3.3.1 Modelagem do problema

Foi realizada uma pesquisa inicial para melhor compreensão do domínio da aplicação, bem como reuniões com pesquisadores do projeto, de maneira a traçar os objetivos e determinar os requisitos do software, a partir das dificuldades existentes e se elas eram amenizadas pelas ferramentas atuais. Dessa forma, foi elaborado um documento de requisitos (mostrado no Apêndice A) que foi validado com os interessados e serviu como base para o desenvolvimento do diagrama de casos de uso UML (Unified Modeling Language)[1], apresentado na Figura 3.

Considerando os casos de uso do sistema, bem como a natureza de sua utilização, foi definido que seria interessante uma solução em que o *back-end* fosse implementado por meio de uma API. Isso possibilita que se, futuramente, houver interesse em desenvolver outras aplicações que consumirão tais dados, como por exemplo um aplicativo para celular, essas funcionalidades já estarão disponíveis. Enquanto isso, o acesso ao sistema se dá pela aplicação Web (*front-end*). Na Figura 4 apresenta-se um diagrama arquitetural do sistema. O modelo conceitual do sistema também foi produzido, utilizando a ferramenta Astah[2], conforme a Figura 5.

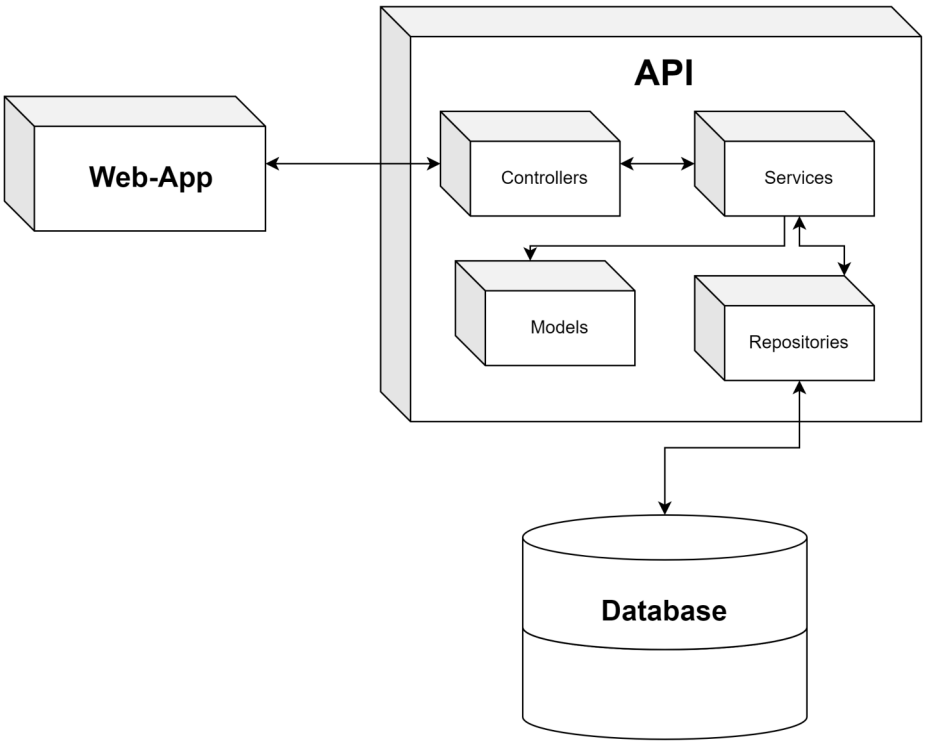
[1] https://www.uml.org/
[2] https://astah.net

Figura 3 – Casos de Uso do Sistema



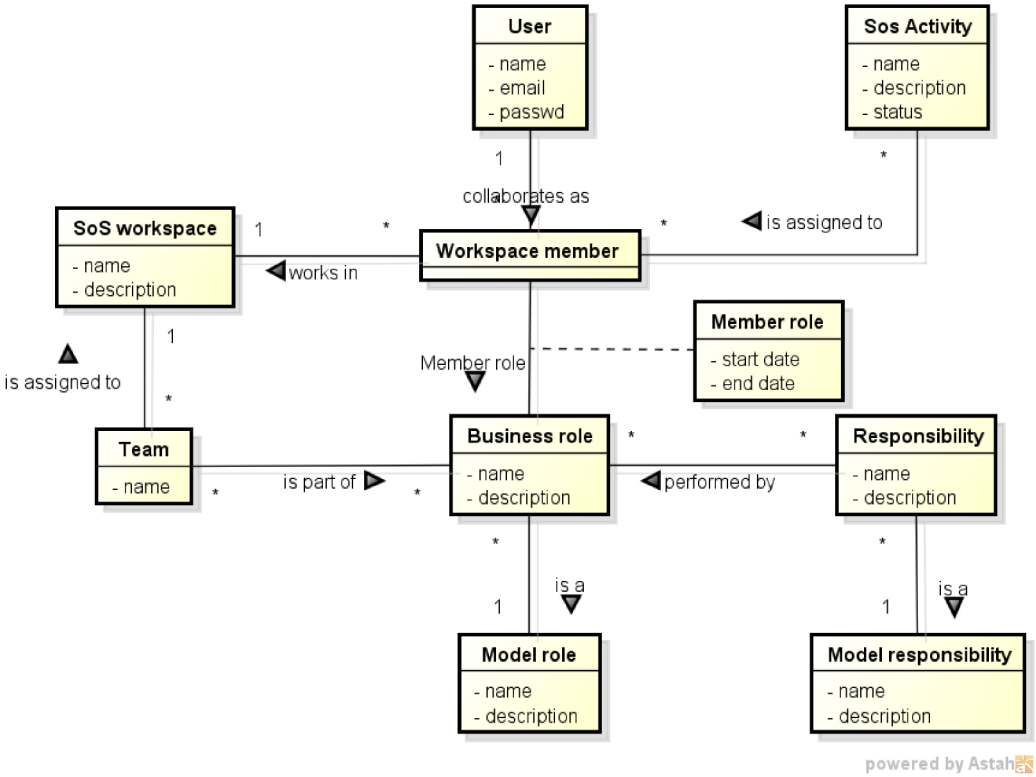
Fonte: Autoria própria

Figura 4 – Arquitetura do Sistema



Fonte: Autoria própria

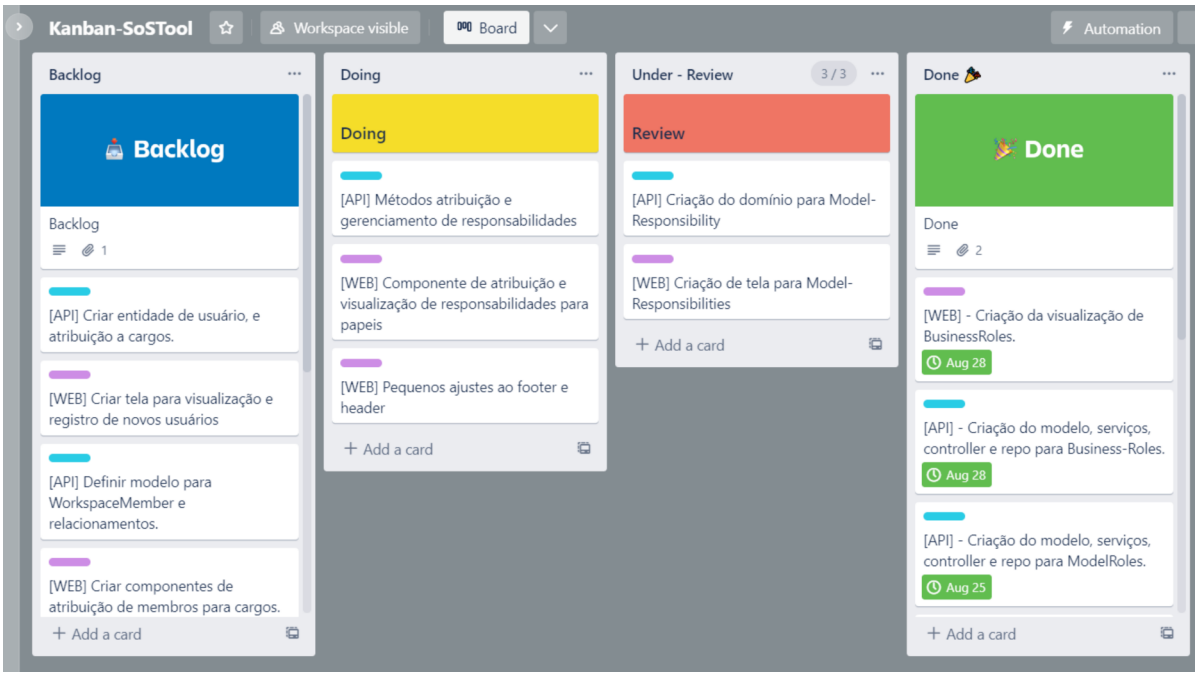
Figura 5 – Modelagem Conceitual do Sistema



Fonte: Autoria própria

Uma vez definidos os requisitos, a arquitetura e as modelagens do software, o desenvolvimento iniciou-se de maneira incremental, por meio de módulos, ou partes, concomitantemente para a API e a aplicação Web. Dadas essas características, o processo de desenvolvimento de software foi pautado nas metodologias ágeis, com *sprints* que duravam uma semana, que era justamente o intervalo entre as reuniões, tendo seu progresso rastreado por meio de um *Kanban board*, como na Figura 6. As reuniões tinham como função o refinamento dos requisitos, bem como revisão e validação das funções implementadas, avaliando principalmente a usabilidade do sistema do ponto de vista do *stakeholder* principal, o gerente de projeto de SoS.

Figura 6 – Kanban do projeto



Fonte: Autoria própria

Essencialmente, o sistema permite o acompanhamento do estado do projeto de um SoS, no âmbito gerencial, sendo possível o mapeamento dos membros em seus respectivos cargos, esses que podem ter suas responsabilidades definidas e também fazer parte de equipes distintas, de maneira a trazer uma visão panorâmica do projeto. Além disso, é possível a definição das atividades realizadas no contexto do SoS, bem como quais membros participaram em sua elaboração. As seções seguintes explicam mais detalhadamente o fluxo de desenvolvimento das funcionalidades apresentadas pelo sistema.

3.3.2 Desenvolvimento da API

A construção da API iniciou com o *Spring Initializr*[3], uma ferramenta de geração e configuração inicial de projetos Spring, permitindo a seleção das dependências pertinentes,

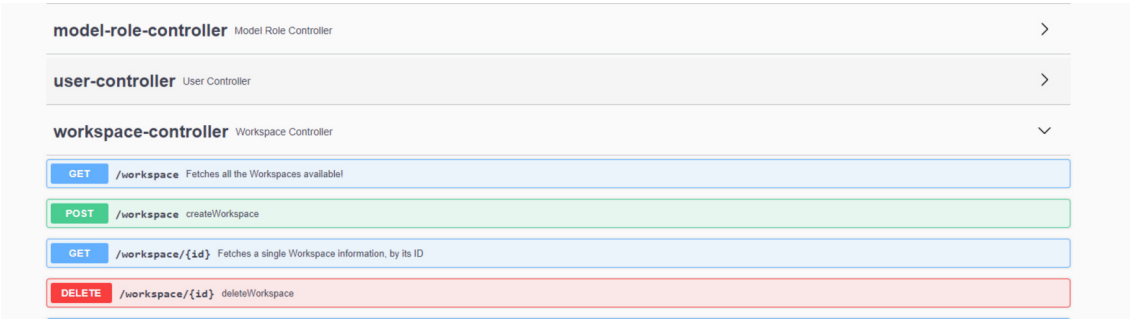
[3] https://start.spring.io

bem como versões utilizadas tanto do framework, quanto da linguagem. No caso deste projeto, foi utilizada a versão 11 do Java, com a versão 2.7.2 do Spring framework e o Maven como gerenciador de dependências. Tal estado inicial já é capaz de executar um servidor localmente que fica disponível em uma porta determinada pelo desenvolvedor. A porta escolhida foi a 8081, porém ainda não há recursos disponíveis.

Conforme a modelagem do domínio de aplicação, as principais classes de domínio são os *Workspaces*, que correspondem aos projetos de SoS, os times que compõe o projeto na classe *Teams*, tem-se também os papéis ou cargos relacionados a cada um desses times (*Business Roles*), as responsabilidades atribuídas a cada papel (*Responsibilities)* e os usuários que possuem esses papéis (*Users*), assim como as atividades desempenhadas no SoS (*Activities*). Tendo esse panorama definido, a implementação teve início.

A partir de então, foi realizada a modelagem da primeira classe de domínio, ou primeira entidade de persistência, chamada *Workspace*, com seus recursos iniciais de criação e remoção. Além disso, configurou-se um conjunto de ferramentas de documentação chamado *Swagger*, que consiste em uma maneira programática e integrada para documentar APIs[4], de maneira análoga ao Javadoc, gerando uma documentação HTML padronizada e interativa para seus recursos e funcionalidades. Na Figura 7 tem-se a visualização do *front-end* de documentação.

Figura 7 – Documentação pelo *Swagger*



Fonte: Autoria própria

A disponibilização de recursos, conforme a arquitetura Web-MVC, ocorre na camada de *Controllers*, esses que no âmbito da aplicação são um conjunto de classes cuja função é mapear URIs (*Unique Resource Identifiers*) lidando com as requisições que chegam à API e repassando para aos seus respectivos métodos na camada de *Models*. Suas implementações, portanto, devem possuir o mínimo possível de lógica de negócios, abstraindo tal função para a camada de modelos. No trecho de código Código-fonte 1 tem-se o exemplo do controlador REST implementado para lidar com requisições relacionadas aos projetos gerenciados pela aplicação (chamados de Workspaces).

Código-fonte 1: Exemplo de controlador implementado

[4] https://swagger.io/docs/specification/2-0/what-is-swagger/

```
1 @CrossOrigin("*")
2 @RestController
3 @RequestMapping(value = "/workspace")
4 public class WorkspaceController {
5
6     private final WorkspaceServiceInterface workspaceService;
7
8     @Autowired
9     public WorkspaceController(WorkspaceServiceInterface
      workspaceService) {
10        this.workspaceService = workspaceService;
11    }
12
13    @PostMapping
14    public ResponseEntity<Workspace> createWorkspace(
15    @RequestBody Workspace obj){
16        Workspace newWorkspace = workspaceService.create(obj);
17        return ResponseEntity.ok().body(newWorkspace);
18    }
19
20    @DeleteMapping(value = "/{id}")
21    public ResponseEntity<Void> deleteWorkspace(@PathVariable Integer
      id){
22        workspaceService.delete(id);
23        return ResponseEntity.noContent().build();
24    }
25 }
```

Esse trecho de código ilustra a injeção de dependências feitas pelo Spring Framework por meio do construtor dessa classe (linhas 6 a 11), no caso o controlador repassa as requisições (linhas 14 a 18 e 20 a 24) para uma classe no componente *Model* do Web-MVC, chamada *workspaceService*.

Essa classe por sua vez é um *Spring Bean*, que consiste em um objeto instanciado e com ciclo de vida gerenciado pelo próprio framework, uma aplicação do princípio de IoC (*Inversion of Contro*l), em que é o objeto quem define suas dependências, sem criá-las. Dessa forma, o próprio framework determina quando é necessário criar um novo serviço para atender a esse controlador, e gerencia a associação desses objetos. O *Spring Bean* de serviço é invocado explicitamente nas linhas 16 e 22.

Também é importante ressaltar como os métodos são enxutos, apenas repassando o que foi recebido nas requisições para suas devidas lógicas de serviço, de forma que requisições do tipo POST para o endereço *http://localhost:8081/workspace*, executam o método *createWorkspace* (linha 14) que repassa o objeto recebido no corpo da requisição para a lógica de serviço

responsável por sua criação, e então retorna, no corpo da resposta, o objeto que foi criado e persistido no banco de dados. Vale ressaltar também, no caso do método DELETE (linha 20 e 21), que é extraído o ID pelo caminho do recurso, ou seja, por um dado contido na URI correspondente a qual recurso será removido. Esse é um parâmetro dinâmico chamado de *path variable*.

É necessário, então, a definição do componente de *Models* respectivo a essa entidade Workspace, contendo as definições necessárias para modelagem desse objeto, seus relacionamentos, lógicas de serviço necessárias e persistência. Para isso, divide-se o componente em três sub-componentes distintos, a classe de modelo, serviço e repositório. O modelo é responsável pela definição de quais campos existirão e seus tipos, bem como relacionamentos que essa classe pode possuir. É o mais próximo do chamado POJO (*Plain Old Java Object*), com seus campos, *Getters*, *Setters* e construtores. No Código-fonte 2 tem-se um exemplo de tal definição (linha 6 a 31).

Código-fonte 2: Exemplo de classe de modelo implementada

```
1 @Entity
2 @Getter
3 @Setter
4 @NoArgsConstructor
5 @AllArgsConstructor
6 public class Workspace {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Integer id;
11
12    @Column(unique = true)
13    @NotEmpty(message = "This field is required")
14    @Length(min = 3, max = 100, message = "Workspace has to have
      between 3 and 100 characters")
15    private String name;
16
17    @Length(max=2000, message = "Workspace description can have at
      most 2000 characters")
18    private String description;
19
20    @OneToMany(cascade = CascadeType.ALL, mappedBy = "workspace")
21    @JsonManagedReference(value = "workspace-team")
22    private Set<Team> teams;
23
24    @Override
25    public boolean equals(Object o) {
26        if (this == o) return true;
27        if (!(o instanceof Workspace)) return false;
```

```
28          Workspace workspace = (Workspace) o;
29          return getId().equals(workspace.getId()) && getName().equals(
        workspace.getName());
30      }
31  }
```

Nesse trecho, ressalta-se a presença de diversos decoradores, precedidos por "@", que são fornecidos pelo Spring Framework a fim de facilitar a modelagem. Nesse ponto, há a definição do método de geração do identificador dos objetos dessa classe, ou seja, sua chave primária no banco, que no caso possui valores gerenciados pelo Spring (linhas 8 a 10).

Existem também restrições de unicidade dos valores de certos campos, bem como validações (linhas 12 a 14). É também possível observar o decorador que modela relacionamentos no Spring framework, onde se tem o *@OneToMany* (linha 20), a partir do qual é definida a estrutura do banco, no caso, a própria tabela que registra os objetos de tipo *Team* contém um campo que é chave estrangeira para qual *Workspace* aquele time pertence. Também pode-se ver a definição do método *equals()* para comparação de objetos (linha 25 a 30).

Outro detalhe interessante é a utilização do projeto Lombok, uma biblioteca Java feita para reduzir a necessidade de código *boilerplate*, que são estruturas repetitivas de código associadas ao padrão de sintaxe da linguagem. Nesse âmbito, a biblioteca Lombok fornece os métodos *getters* e *setters*, bem como construtores sem argumento e com todos argumentos (linhas 2 a 5), reduzindo consideravelmente a quantidade de linhas de código, sem perda de funcionalidade.

Em seguida, é feita a definição do sub-componente de repositório, cuja função é utilizar um ORM, como já citado, o Hibernate, para implementar o mapeamento dessas classes de modelos para suas tabelas persistidas em um banco de dados. Dessa forma, todo tipo de acesso ao banco de dado é feito por uma classe de repositório, no Código-fonte 3 segue o exemplo de implementação dessa classe.

Código-fonte 3: Exemplo de classe de repositório implementada

```
1  @Repository
2  public interface WorkspaceRepository extends JpaRepository<Workspace,
       Integer> {
3
4      @Query("SELECT t FROM Team t " +
5              "WHERE t.workspace.id =:workspace_id")
6      Set<Team> findTeamsOfWorkspace(@Param("workspace_id") Integer
       workspace_id);
7
8      Set<Workspace> findByNameContaining(String name);
9
10 }
```

Nessa classe, é notável a presença de um código SQL, o Spring Framework, que permite a confecção de consultas customizadas, utilizando seu dialeto SQL adaptado pelo ORM ao dialeto do banco de dados utilizado. Essa consulta é associada ao método que a sucede imediatamente, podendo ser invocado pelos serviços que precisarem realizá-la.

Além disso, estão presentes as funcionalidades do Spring Data, que realiza o mapeamento por meio de um ORM da classe Java para uma tabela no banco (linha 2), bem como provê diversas consultas usuais sob métodos com nomes padronizados para facilitar a implementação, como por exemplo *save()* ou *findById()*, que podem ser invocados por qualquer serviço que queira realizar um registro ou atualização no banco, ou consulta por Id, diminuindo a quantidade de linhas de código escritas.

Também está presente outra funcionalidade do Spring Data, o mapeamento direto de um método em uma consulta, apenas pelo seu nome. O método *findByNameContaining* (linha 8) tem assinatura padronizada e o próprio framework implementa a consulta para o programador a partir dela. Nesse caso, é uma busca no banco pelos Workspaces que contenham a *string: name* recebida como argumento, sem a necessidade de escrita de código SQL.

Com modelo e repositório definidos, implementa-se a última parte do componente *Model* do Web-MVC, a classe de serviço que é responsável pelas lógicas de negócio da aplicação, tendo seus métodos invocados pelos controladores e orquestrando a manipulação das classes de domínio e repositórios. É interessante definir uma interface para os serviços, a fim de permitir diversas implementações quando necessário, bem como explicitar a intenção do desacoplamento dos métodos que serão expostos para chamada por outros objetos, como por exemplo controladores de métodos internos, em conformidade com o princípio de injeção de dependências.

O trecho de Código-fonte 4 contém um exemplo dessa interface e o Código-fonte 5 da classe que a implementa.

Código-fonte 4: Exemplo de interface de serviço

```
1 public interface WorkspaceService {
2
3     Workspace findById(Integer id);
4
5     List<Workspace> findAll();
6
7     Set<Team> findTeams(Integer id);
8
9     Workspace create(Workspace obj);
10
11    void delete(Integer id);
12 }
```

Código-fonte 5: Exemplo de classe de implementação da interface

```
1 @Service
2 public class WorkspaceServiceImpl implements WorkspaceService{
3     private final WorkspaceRepository workspaceRepository;
4
5     @Autowired
6     public WorkspaceServiceImpl(WorkspaceRepository
      workspaceRepository) {
7         this.workspaceRepository = workspaceRepository;
8     }
9
10    @Override
11    public Workspace findById(Integer id) {
12        Optional<Workspace> obj = workspaceRepository.findById(id);
13        return obj.orElseThrow(() -> new ObjectNotFoundException("
      Workspace not found!!"));
14    }
15
16    @Override
17    public Workspace create(Workspace obj) {
18        obj.setId(null);
19        obj.setTeams(null);
20        return workspaceRepository.save(obj);
21    }
22
23    @Override
24    public List<Workspace> findAll() {
25        return workspaceRepository.findAll();
26    }
27
28    @Override
29    public void delete(Integer id) {
30        findById(id);
31        workspaceRepository.deleteById(id);
32    }
33
34    @Override
35    public Set<Team> findTeams(Integer id) {
36        return workspaceRepository.findTeamsOfWorkspace(id);
37    }
38 }
```

Vê-se que no código há novamente a injeção de dependências feita no construtor da classe de implementação (linhas 3 a 8), determinando que em tempo de execução o Spring framework proverá um objeto do tipo adequado para ser utilizado. Também há a presença das

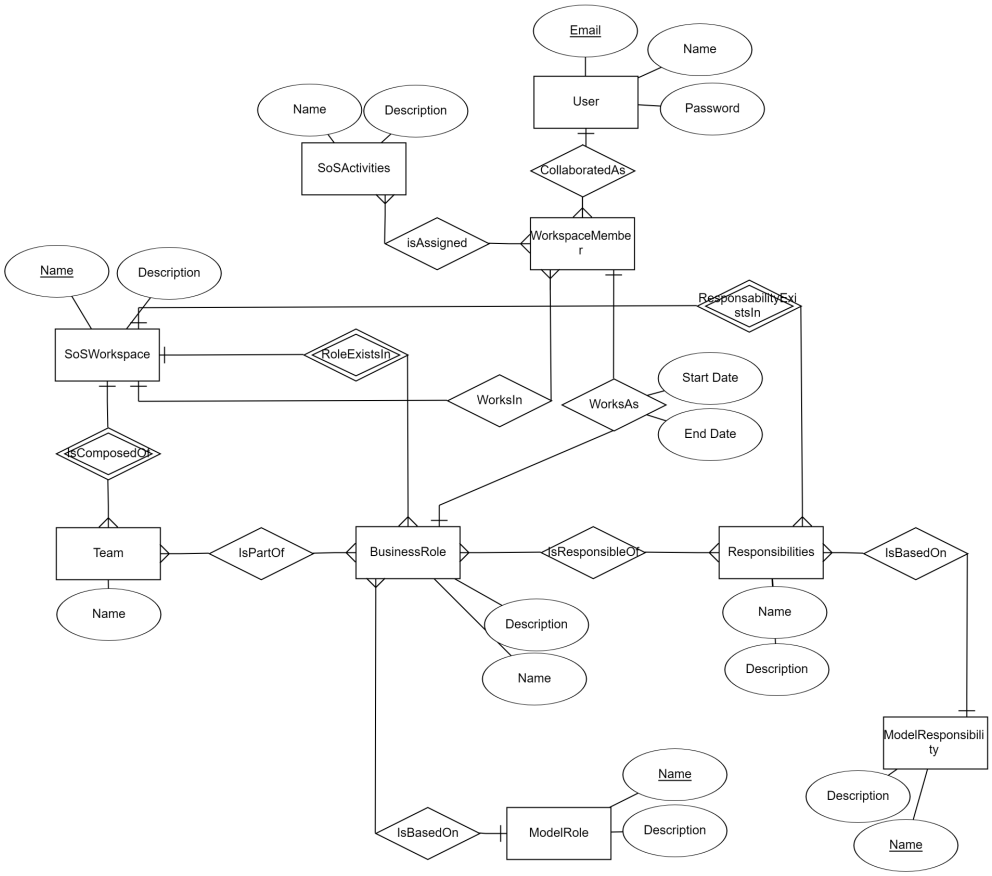
lógicas de serviço de cada método, e respectivas chamadas para o banco de dados por meio de um repositório.

O prosseguimento e a evolução da API ocorreu conforme esse mesmo processo. Para cada nova funcionalidade, foram mapeadas as devidas entidades do domínio da aplicação, com seus serviços e repositórios, e expondo os novos recursos por meio de controladores, de acordo com as necessidades identificadas.

3.3.3 Visão geral das tabelas e estrutura do banco de dados.

Durante a fase de desenvolvimento do projeto de software, foi construído um modelo entidade relacionamento para guiar a implementação da estrutura de persistência de dados do sistema, o qual pode ser visto na Figura 8.

Figura 8 – Modelo Entidade Relacionamento projetado.

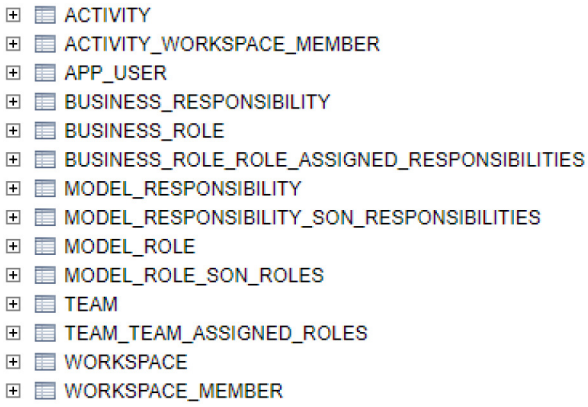


Fonte: Autoria própria

Na Figura 9, pode-se ter uma visão geral das tabelas que existem no banco de dados. Elas representam as entidades de persistência implementadas, que podem ser visualizadas no diagrama entidade-relacionamento resultante da implementação, apresentado na Figura 10. As entidades de persistência são uma abstração de nível mais alto do que de fato existe no banco, já que são diretamente relacionadas às classes de Modelo definidas no desenvolvimento da API. A

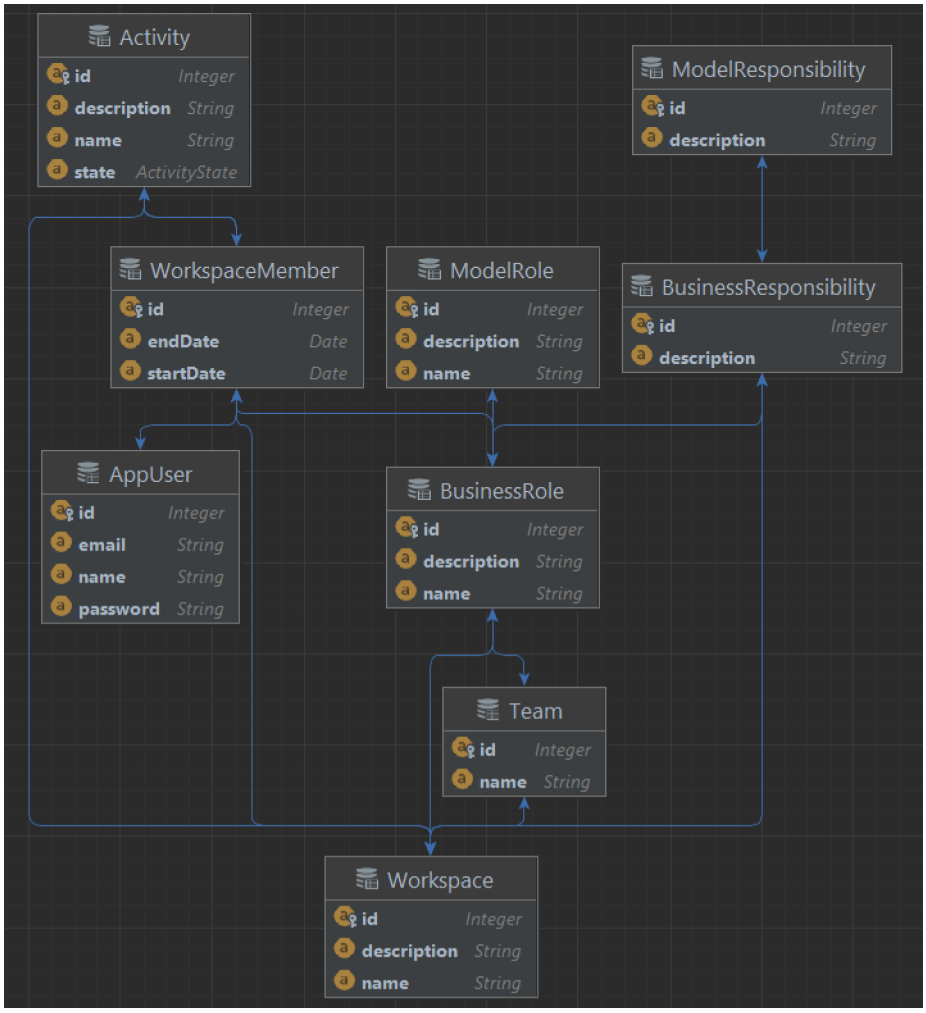
partir delas e seus relacionamentos a estrutura do banco é definida, dessa forma, existem tabelas que modelam exclusivamente os relacionamentos do tipo muitos para muitos.

Figura 9 – Tabelas existentes no banco de dados



Fonte: Autoria própria

Figura 10 – Diagrama Entidade Relacionamento do SoS Tool



Fonte: Autoria própria

3.3.4 Desenvolvimento da aplicação Web

No desenvolvimento da API, foi feito o desenvolvimento da aplicação Web que consumirá seus recursos e será de fato acessada pelos usuários. Por meio da análise do documento de requisitos, das reuniões semanais com os *stakeholders* e das funcionalidades disponíveis na API, foi modelada a aplicação Web.

Na aplicação desenvolvida, os usuários possuem capacidade de criar novos projetos de SoS, que possuem um conjunto de atividades padrão consideradas no projeto de doutorado do qual este projeto de formatura colabora (VARGAS, 2022). Durante a elaboração do projeto pode-se definir as equipes envolvidas, alocar cargos a essas equipes, que podem ser baseados em modelos que são comuns em SoSs, com responsabilidades específicas que também podem ser baseadas em modelos comuns pré-definidos. Tais cargos podem ser atribuídos a pessoas que são registradas na equipe pelo responsável pela gerência do projeto. As atividades do SoS podem ser associadas aos colaboradores que participaram em seu desenvolvimento.

A implementação foi feita utilizando o *framework Angular*, juntamente do *framework* de CSS chamado *Bulma*[5] para facilitar a criação dos *layouts* das páginas e, dessa forma, acelerar o desenvolvimento, permitindo um foco maior nas funcionalidades, sem o emprego de muito tempo na estilização da interface.

O desenvolvimento em *Angular* é pautado na arquitetura de seus componentes e no princípio de fluxo unidirecional de dados. Cada componente é composto de um conjunto de arquivos HTML, CSS e *TypeScript*, que são responsáveis tanto pela estruturação e renderização, quanto lógicas de apresentação. Existem dois conjuntos básicos de componentes, os chamados, componentes inteligentes, que são responsáveis por mudanças de estados da aplicação e passagem de informação via serviços, se comunicando com o *back-end* e os componentes de apresentação que apenas recebem e expõem informações.

Geralmente, as páginas roteadas são os componentes inteligentes, adotando o padrão de projeto *Observer* para lidar com as chamadas para a API, que são por natureza assíncronas. Já os componentes de apresentação são os elementos mais básicos da página, como por exemplo um rodapé. No trecho de código Código-fonte 6 tem-se um exemplo dos serviços implementados em *Angular* que se utilizam desse padrão de projeto.

Código-fonte 6: Exemplo de serviço Angular

```
1 import { HttpClient } from '@angular/common/http';
2 import { Observable } from 'rxjs';
3 import { environment } from 'src/environments/environment';
4
5 @Injectable({
6   providedIn: 'root'
```

[5] https://bulma.io/documentation/

```
})
export class WorkspaceService {

  baseUrl: String = environment.baseUrl;

  constructor(private http: HttpClient) { }

  findWorkspaces(): Observable<Workspace[]>{
    const url = `${this.baseUrl}/workspace`;
    return this.http.get<Workspace[]>(url);
  }

  createWorkspace(workspace: Workspace): Observable<Workspace> {
    const url = `${this.baseUrl}/workspace`;
    return this.http.post<Workspace>(url, workspace);
  }
}
```

Assim como no desenvolvimento da API, o *framework Angular* utiliza a injeção de dependências para realizar o gerenciamento dos objetos que utilizam métodos externos, no caso, esse serviço exemplificado é chamado pelos componentes que necessitam de seus métodos, tendo sua dependência explícita no construtor da classe, conforme o Código-fonte 7.

Código-fonte 7: Exemplo de componente Angular

```
import ... ;

@Component({
  selector: 'app-workspace-read',
  templateUrl: './workspace-read.component.html',
  styleUrls: ['./workspace-read.component.sass']
})
export class WorkspaceReadComponent implements OnInit {

  workspaceArray: Workspace[] = [];

  constructor(private service: WorkspaceService, private router:
   Router) { }

  ngOnInit(): void {
    this.getWorkspaces();
  }

  getWorkspaces(): void { ... }
}
```

Como o *TypeScript* é uma linguagem fortemente tipada, a definição dos tipos dos objetos é feita de forma a haver um casamento com o que é fornecido pela API, evitando erros de serialização em tempo de execução. No trecho Código-fonte 8, tem-se um exemplo da definição de um tipo utilizado (linha 1) com seus campos (linha 2 a 6).

Código-fonte 8: Exemplo de modelo TypeScript

```
1 export interface Workspace{
2   id: number;
3   name: string;
4   description: string;
5   teams: Teams[];
6   activities: Activity[];
7 }
```

Portanto, fez-se incrementalmente a construção das interfaces, dando vida e personalidade para o sistema. As telas das principais funcionalidades da aplicação Web são apresentadas na seção a seguir. Na tabela Tabela 1 mostra-se o histórico de implementação, com as principais tarefas concluídas em cada *Sprint* de desenvolvimento.

Tabela 1 – Tabela de histórico de implementação

Sprint	Tarefas Concluídas
1	Criação da entidade de Workspace, homepage e visualização de workspaces
2	Criação da entidade Teams e tela de visualização de um workspace, com seus times.
3	Criação das entidades Model-Roles e Business-Roles e suas respectivas páginas e interações no front-end.
4	Criação da entidade Model-Responsibilities, sua página na aplicação Web e componentes de atribuição de responsabilidades aos papeis.
5	Criação da entidade de usuários, e tela para sua listagem. Implementação da atribuição de usuários em cargos dos workspaces.
6	Adição de funcionalidades de relatórios sobre os usuários e sobre a estrutura do workspace.
7	Criação da entidade de Activities, assim como as funcionalidades de atribuição à workspaces.
8	Adição das funcionalidades internas às atividades, atribuição de usuários colaboradores e estados.

3.4 Considerações Finais

Neste capítulo, foi apresentado o processo completo de desenvolvimento da aplicação, desde o aspecto da modelagem, projeto e metodologia de desenvolvimento de software, até os detalhes de implementação considerando as tecnologias que foram escolhidas para este projeto.

Capítulo 4

RESULTADOS OBTIDOS

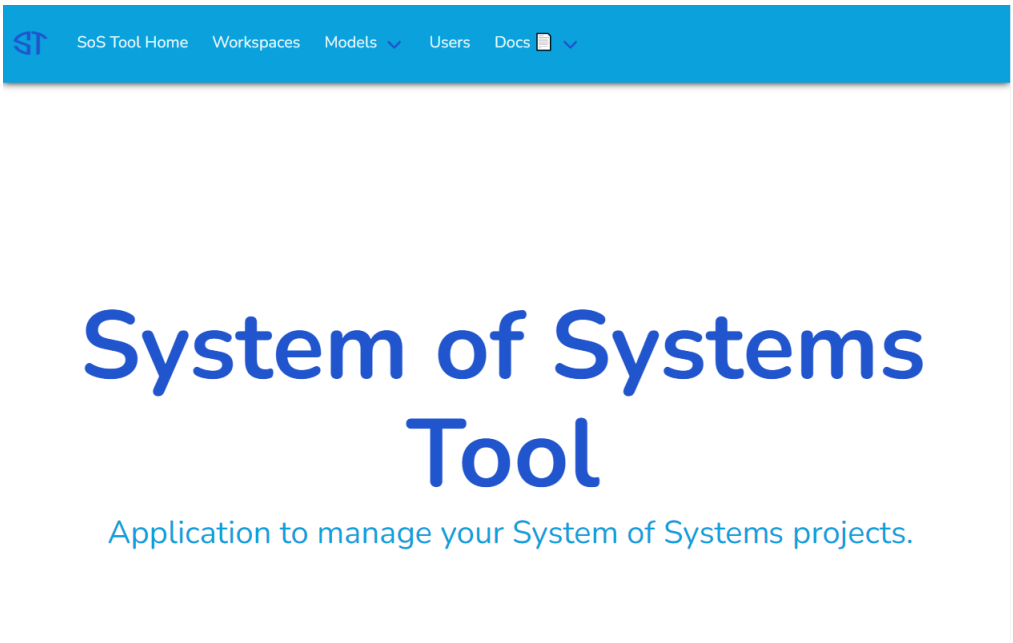
4.1 Considerações Iniciais

Neste capítulo apresentam-se os resultados obtidos ao final do desenvolvimento da aplicação, com as funcionalidades e telas a disposição do usuário do sistema.

4.2 Resultados Obtidos

Neste trabalho foi elaborado a aplicação *SoS Tool*, uma ferramenta de auxílio ao gerenciamento de projetos de SoS. Foram desenvolvidos, integralmente, tanto a API, que fornece recursos e serviços para o *front-end*, quanto a aplicação Web que pode ser utilizada por profissionais de administração de SoS, permitindo que acompanhem o desenvolvimento de seus projetos. A seguir, são mostradas algumas das principais telas da aplicação Web desenvolvida, começando pela Figura 11 que consiste na tela inicial da aplicação.

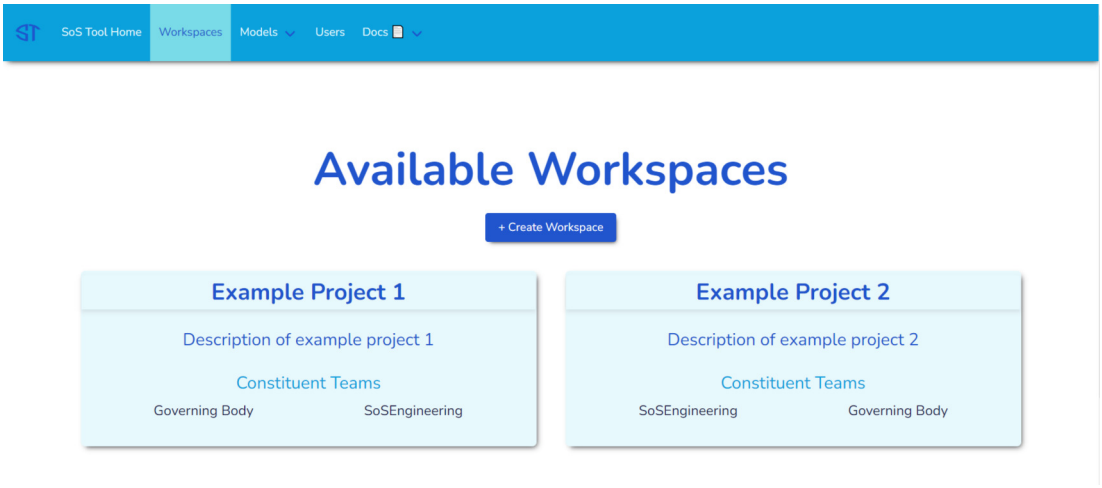
Figura 11 – Página Inicial do SoS Tool



Fonte: Autoria própria

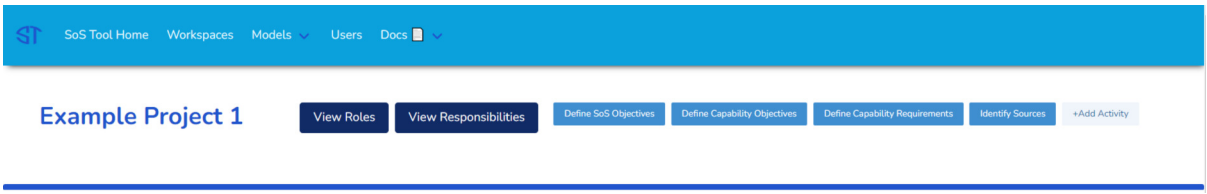
A partir dessa tela inicial, o usuário pode navegar, por meio da barra superior de navegação, até a funcionalidade que deseja utilizar. Na Figura 12 tem-se a tela de visualização dos projetos existentes, chamados Workspaces, permitindo que ele crie novos projetos de SoS de acordo com sua necessidade, visualize os já existentes e também selecione um projeto que deseja acessar e levando-o para a página específica desse projeto. Essa pagina possui duas visualizações principais, uma para o projeto como um todo com acesso a suas atividades e informações rastreadas (Figura 13) e outra com as informações dos times existentes, listando os papeis presentes e permitindo troca do time visualizado por abas (Figura 16).

Figura 12 – Página de Workspaces



Fonte: Autoria própria

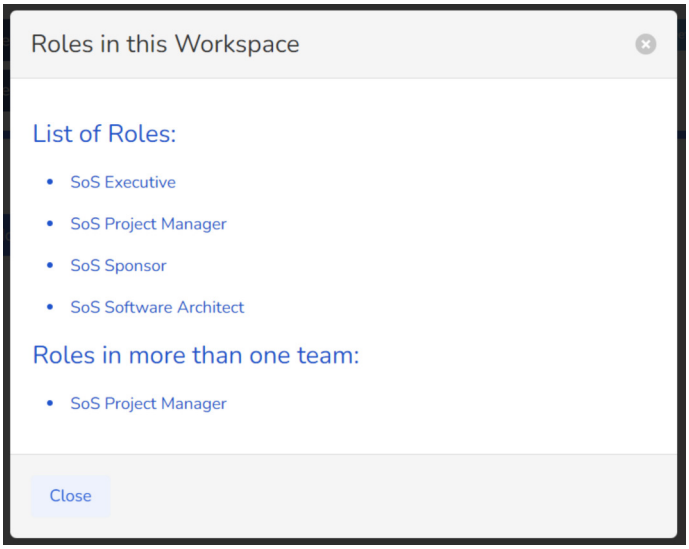
Figura 13 – Visualização das atividades e informações de um Workspace



Fonte: Autoria própria

No trecho da Figura 13 é possível acessar informações rastreadas sobre o *Workspace*, como por exemplo a visualização de papéis existentes e quais papeis desempenham funções em mais de uma equipe (Figura 14).

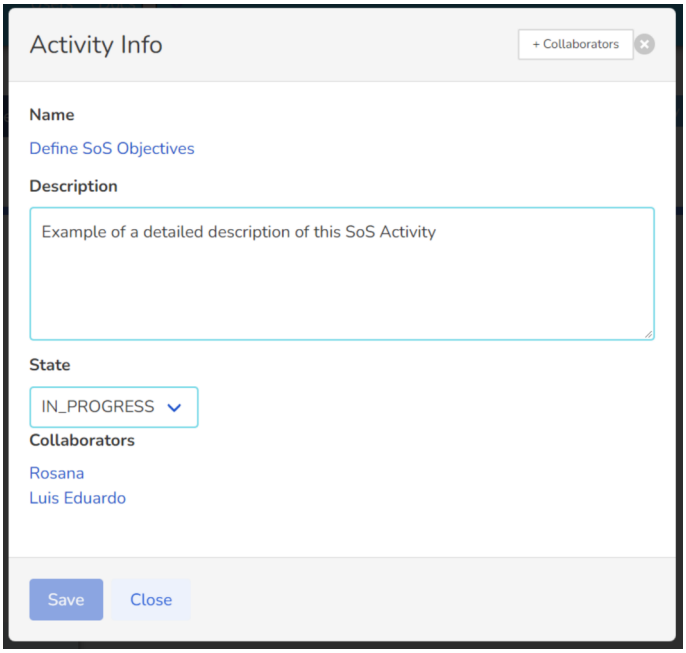
Figura 14 – Relatório sobre os papeis desempenhados no SoS



Fonte: Autoria própria

Permite-se também a adição de novas atividades, assim como a visualização e edição de atividades existentes, adicionando-se membros colaboradores ou refinando sua descrição Figura 15.

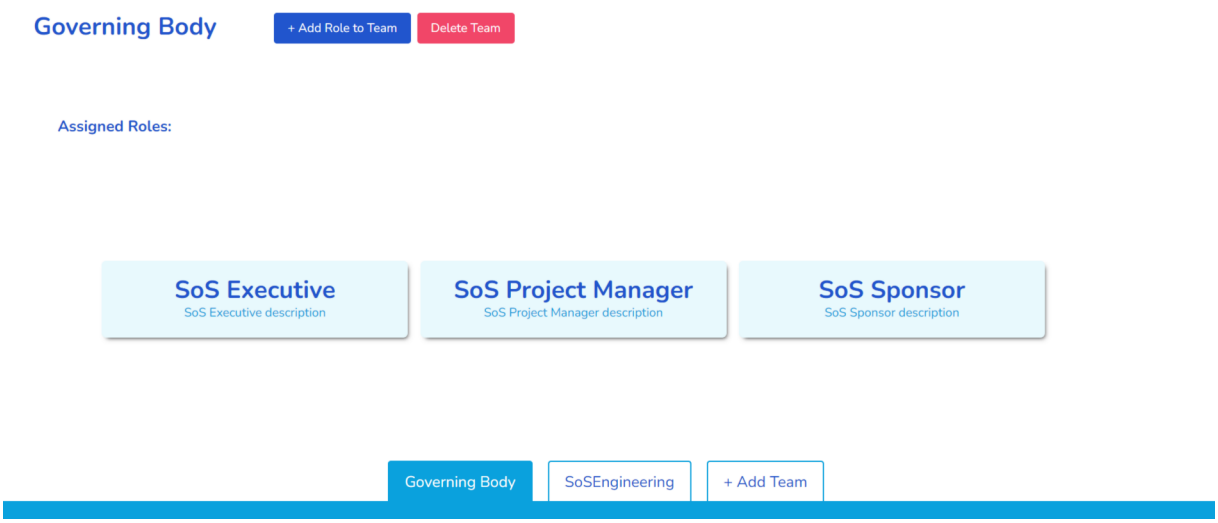
Figura 15 – Modal de Atividades



Fonte: Autoria própria

Na mesma página tem-se a visualização de times (Figura 16) que permite a gerência de quais papeis compõe cada equipe.

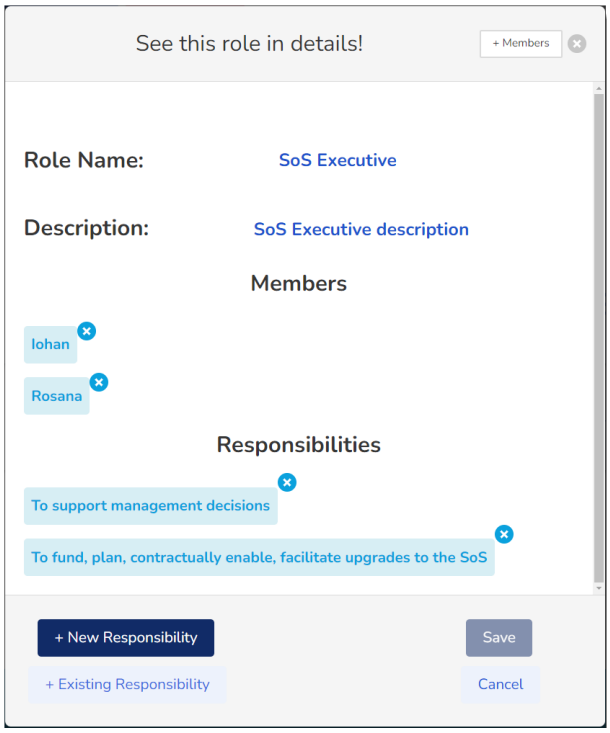
Figura 16 – Visualização das equipes do workspace



Fonte: Autoria própria

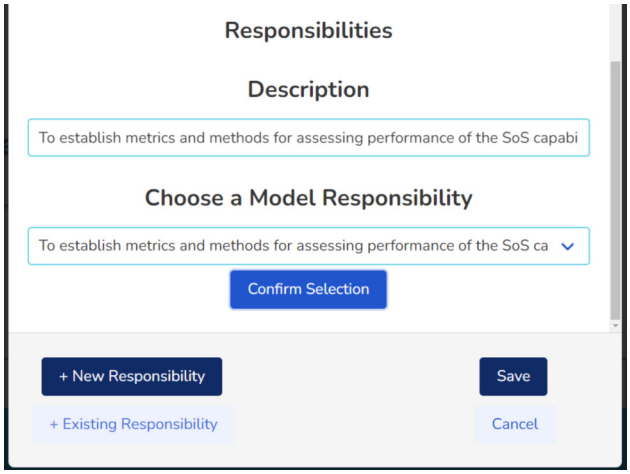
É possível selecionar um desses papéis para alterá-lo ou conferi-lo de maneira direcionada (Figura 17), permitindo atribuição de responsabilidades (Figura 18) e analogamente de pessoas.

Figura 17 – Visualização de um papel selecionado



Fonte: Autoria própria

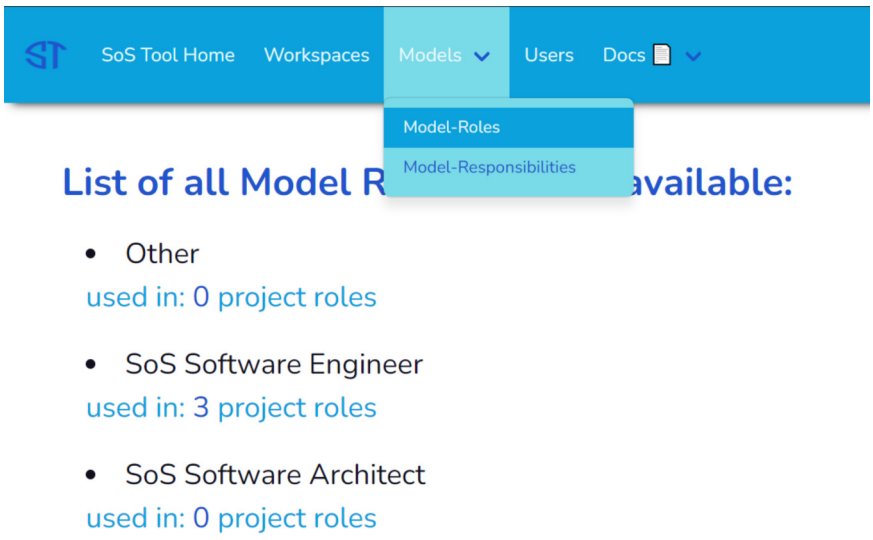
Figura 18 – Adição de responsabilidades



Fonte: Autoria própria

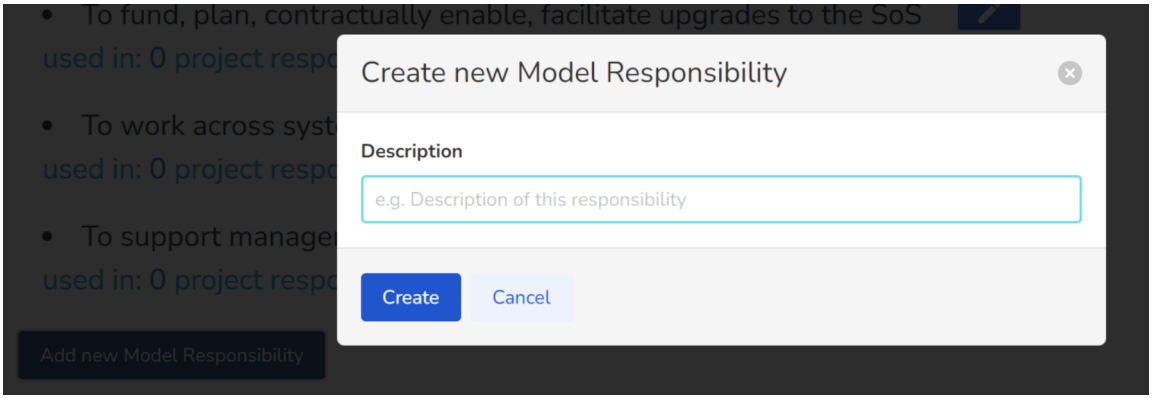
A seção de modelos é voltada para um administrador do sistema, permitindo a visualização dos modelos existentes, tanto de papéis quanto responsabilidades, e quantas vezes foram utilizados dentro dos projetos no sistema, denotando quais são mais comuns, além de poder criar novos modelos e alterar os já existentes, para que se adéque melhor ao contexto de SoS. Na Figura 19 é demonstrado a tela de modelos de papéis enquanto na Figura 20 tem-se o exemplo de criação de uma nova responsabilidade modelo. Vale ressaltar que, a tela de papéis-modelo também possui seu respectivo modal para criação de novos registros.

Figura 19 – Tela de Papéis-Modelo



Fonte: Autoria própria

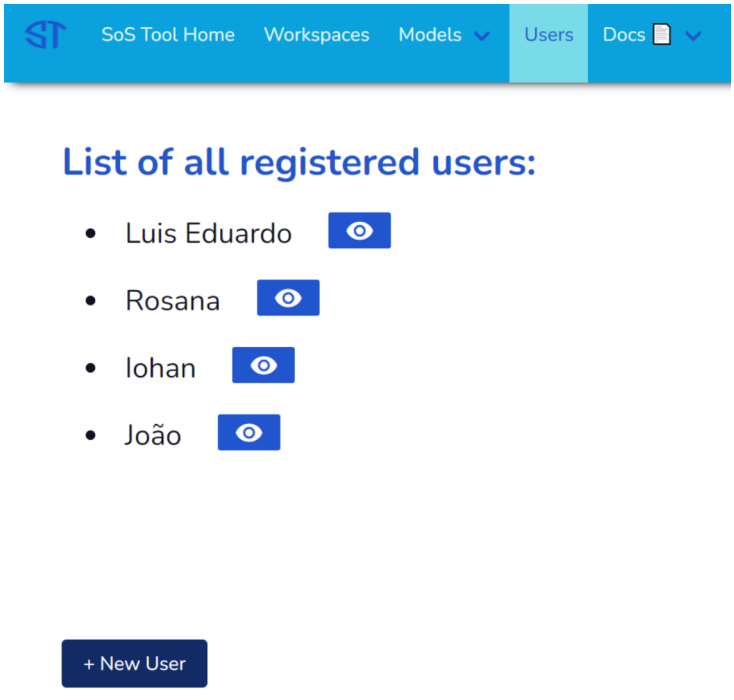
Figura 20 – Criação de Responsabilidades-Modelo



Fonte: Autoria própria

Outra seção desenvolvida foi a de gerenciamento de usuários, permitindo o registro de novos usuários, com modal análogo ao dos modelos, assim como visualização de seus dados (Figura 21) e histórico de cargos (Figura 22).

Figura 21 – Visualização de Usuários



Fonte: Autoria própria

Figura 22 – Resumo de um usuário

Fonte: Autoria própria

Além das funcionalidades obtidas, também é possível navegar para páginas de documentação, como o *front-end* do *Swagger*, repositório do projeto no *Git-Hub*[1].

Agregando os resultados obtidos e realizando uma análise das principais funcionalidades no escopo da aplicação *SoS Tool* as quais estão relacionadas ao *SoS Holistic View Framework*, pode-se comparar as ferramentas consideradas previamente neste trabalho (ver Seção 2.3), conforme a Tabela 2.

Pode-se observar na tabela que a *SoS Tool* procura suprir as funcionalidades relacionadas ao gerenciamento de equipes e responsabilidades que não estão presentes nas demais ferramentas consideradas na comparação.

Tabela 2 – Comparação entre as ferramentas consideradas

	Ferramentas				
	Trello	Notion	Jira	ClickUp	SoS Tool
Criar projeto	OK	OK	OK	OK	OK
Definir equipes	X	X	OK	OK	OK
Definir papeis	X	X	OK	X	OK
Associar responsabilidades	X	X	X	X	OK
Definir atividades	OK	OK	OK	OK	OK
Associar participantes	OK	OK	OK	OK	OK
Utilizar papeis-modelo	X	X	X	X	OK
Utilizar responsabilidades-modelo	X	X	X	X	OK

[1] https://github.com/stars/dupradosantini/lists/sos-tool

4.3 Considerações Finais

Neste capítulo foram apresentados os resultados obtidos, com detalhes, ressaltando as principais propostas da aplicação, no auxílio da gerência de projetos para SoS. No próximo capítulo, apresenta-se a conclusão do trabalho, assim como considerações do autor acerca do curso de graduação.

Capítulo 5

CONCLUSÃO

5.1 Contribuições

O objetivo deste trabalho foi desenvolver uma ferramenta de gerenciamento de projetos, focada na aplicação em SoS, de maneira a auxiliar, a maneira com que projetos de SoS são gerenciados, se adequando a ao *framework* visto anteriormente. Vale destacar que os SoS possuem características peculiares que não são adequadamente contempladas em ferramentas tradicionais. Assim sendo, elenca-se as seguintes contribuições científicas e tecnológicas deste projeto:

1. Proposta de uma solução para auxiliar no gerenciamento de projetos no âmbito de SoS.
2. Criação de uma aplicação Web desenvolvida integralmente neste trabalho, permitindo o mapeamento do estado de projetos SoS, do ponto de vista gerencial, para contribuir com seus respectivos desenvolvimentos.
3. Criação de uma API, também construída integralmente neste trabalho, para fornecimento de recursos e serviços para a aplicação Web desenvolvida.

5.2 Considerações sobre o Curso de Graduação

O curso de Engenharia de Computação da USP de São Carlos é notável por possuir uma das maiores cargas horárias, no âmbito de horas-aula, dentre os demais cursos de engenharia do campus. Seu caráter generalista, abrangendo majoritariamente tanto as áreas de Engenharia Elétrica, quanto Ciências da Computação, dentro de um período de somente 5 anos, reflete em uma sobrecarga do aluno de graduação, reduzindo seu tempo livre, que é de suma importância para o aluno fixar os conceitos introduzidos em sala de aula, além de tornar-se um detrimento à capacidade do aluno usufruir dos outros dois pilares que constituem uma universidade, a pesquisa e extensão.

Acerca da parte relacionada a computação do curso, os conceitos complexos que são estudados requerem também bastante prática extra-sala, pois as técnicas e habilidades aprendidas exigem esse esforço extra para que o aluno seja capaz de elaborar e desenvolver códigos de maneira eficiente. Sugere-se a redução da carga horária de aulas obrigatórias, permitindo um

foco maior nas atividades dentro e fora da sala associadas a uma maior liberdade de escolha do aluno do foco de sua graduação, seja no âmbito de eletrônica ou computação.

Durante a realização deste trabalho, foram fundamentais diversos conceitos aprendidos durante a graduação, destacando-se disciplinas como, Programação Orientada a Objetos, Bases de Dados, Análise e Projeto Orientado a Objetos, Engenharia de Software, Sistemas Distribuídos, Reuso de Software e Arquitetura de Software. Vale ressaltar que apesar da grade ser bastante completa no âmbito conceitual, não é muito contemporânea, em relação à tecnologias utilizadas no mercado, requerendo um esforço extra-sala dos alunos para aprender tais habilidades. Dessa forma, sugere-se também, por meio de disciplinas optativas, uma introdução dos alunos as tecnologias mais proeminentes para desenvolvimento de softwares hoje.

5.3 Trabalhos Futuros

Como sugestões de trabalhos futuros, pode-se destacar a implementação de demais aspectos relacionados ao gerenciamento de SoS, como *Objectives*, *Capabilities*, *Requirements* e *Constituent Systems*, a expansão das funcionalidades disponíveis para membros dos projetos, assim como a implementação de autenticação e autorização por meio do *Spring Security*, utilizando, por exemplo, o *JSON Web Token*, para gerenciamento de sessões *stateless*. Por fim, é importante lembrar que a aplicação Web, desenvolvida neste trabalho, ainda requer a validação por usuários, podendo sofrer alterações para incorporar suas possíveis sugestões.

REFERÊNCIAS

ABDULLAH, H. M.; ZEKI, A. M. Frontend and backend web technologies in social networking sites: Facebook as an example. In: IEEE. **2014 3rd international conference on advanced computer science applications and technologies**. [S.l.], 2014. p. 85–89. Citado 2 vezes nas páginas 26 e 27.

CHEN, T.-H.; SHANG, W.; JIANG, Z. M.; HASSAN, A. E.; NASSER, M.; FLORA, P. Detecting performance anti-patterns for applications developed using object-relational mapping. In: **Proceedings of the 36th International Conference on Software Engineering**. [S.l.: s.n.], 2014. p. 1001–1012. Citado na página 28.

DYBA, T.; DINGSOYR, T. Empirical studies of agile software development: A systematic review. **Information and Software Technology**, v. 50, n. 9-10, p. 833–859, 2008. Citado 2 vezes nas páginas 23 e 26.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Citado na página 27.

IRELAND, C.; BOWERS, D.; NEWTON, M.; WAUGH, K. A classification of object-relational impedance mismatch. In: IEEE. **2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications**. [S.l.], 2009. p. 36–43. Citado na página 28.

JOHNSON, R. J2ee development frameworks. **Computer**, IEEE, v. 38, n. 1, p. 107–110, 2005. Citado na página 28.

LANE, J. A. Sos management strategy impacts on sos engineering effort. In: MÜNCH, J.; YANG, Y.; SCHÄFER, W. (Ed.). **New Modeling Concepts for Today's Software Processes**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 74–87. ISBN 978-3-642-14347-2. Citado na página 22.

LANE, J. A. **What is a System of Systems and Why Should I Care?** 2013. Tecnical Report, University of Sourthern California. Citado na página 21.

MAIER, M. W. Architecting principles for systems-of-systems. In: **INCOSE International Symposium**. [S.l.]: John Wiley & Sons, Inc., 1998. v. 1, p. 565–573. ISSN 1520-6858. Citado 2 vezes nas páginas 21 e 25.

PARVEEN, Z.; FATIMA, N. Performance comparison of most common high level programming languages. **International Journal of Computing Sciences Research**, v. 5, p. 246–258, 10 2016. Citado na página 32.

PMI (Ed.). **A Guide to the Project Management Body of Knowledge (PMBOK Guide)**. 6. ed. Newtown Square, PA: Project Management Institute, 2017. Citado na página 22.

SYROMIATNIKOV, A.; WEYNS, D. A journey through the land of model-view-design patterns. In: **2014 IEEE/IFIP Conference on Software Architecture**. [S.l.: s.n.], 2014. p. 21–30. Citado 2 vezes nas páginas 28 e 29.

TORRES, A.; GALANTE, R.; PIMENTA, M. S.; MARTINS, A. J. B. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. **information and software technology**, Elsevier, v. 82, p. 1–18, 2017. Citado na página 28.

TREGUBOV, A.; LANE, J. A. Simulation of kanban-based scheduling for systems of systems: Initial results. **Procedia Computer Science**, v. 44, p. 224 – 233, 2015. ISSN 1877-0509. Conference on Systems Engineering Research. Citado na página 29.

VARGAS, I. G. **Towards a Framework to Support the Management of System of Systems**. 2022. Doutorado em andamento, PPG-CCMC-ICMC-Universidade de São Paulo. Citado 2 vezes nas páginas 22 e 44.

VARGAS, I. G.; GOTTARDI, T.; BRAGA, R. T. V. Approaches for integration in system of systems: A systematic review. In: **Proceedings of the 4th International Workshop on Software Engineering for Systems-of-Systems**. New York, NY, USA: ACM, 2016. (SESoS '16), p. 32–38. ISBN 978-1-4503-4172-1. Citado na página 23.

APÊNDICE A

DOCUMENTO DE REQUISITOS DO SOS TOOL

A.1 Escopo, Público Alvo e Uso

Este projeto tem o propósito de auxiliar no processo de gerenciamento de Sistemas de Sistemas, fornecendo uma ferramenta de organização para essas aplicações e suas necessidades específicas.

Os gerentes de projeto, no contexto de Sistemas de Sistemas, devem ser capazes de delegar responsabilidades para papéis específicos, e também alocá-los em seus respectivos times, de acordo com a estrutura do SoS. Além disso deve ser possível atribuir tais papéis para as pessoas que os desempenharão.

Além disso a ferramenta deve prover a estrutura para definir as Atividades do SoS, atribuindo-as às pessoas que as desempenharão e rastrear seu progresso por meio de um estado de andamento.

Dessa maneira este sistema é primariamente voltado para pessoas que queiram organizar projetos de SoS, mas também para participantes que gostariam de ter uma perspectiva da estrutura do projeto que se encontram e suas obrigações.

A.2 Requisitos funcionais do SoS Tool

A.2.1 Requisitos de Gerente de Projetos

Os requisitos a seguir referem-se a funcionalidades voltadas para os gerentes de projeto de SoS.

1. O sistema deve possibilitar a criação de projetos com um ou mais times.
2. O sistema deve permitir a criação de papéis de projeto, que podem ser atribuídos a um ou mais times.
3. O sistema deve oferecer a possibilidade de criação de responsabilidades, que compreendem uma obrigação, podendo ser atribuídas a um ou mais cargos.

4. O sistema deve permitir a criação de papéis e responsabilidades a partir de modelos já existentes, facilitando o trabalho do gerente de projetos.

5. O sistema deve permitir o registro de novos participantes.

6. O sistema deve permitir a elaboração de atividades.

7. O sistema deve permitir a atribuição de participantes às atividades.

A.2.2 Requisitos de Participante

Os requisitos a seguir referem-se a funcionalidades voltadas para os usuários participantes do projeto, que não são gerentes.

8. O sistema deve permitir a visualização dos projetos em que o participante foi inserido.

9. O sistema deve permitir que o participante visualize quais seus papéis e responsabilidades nos projetos, bem como quais atividades ele participa.

10. O sistema deve fornecer uma visualização do histórico de papéis do participante.

A.2.3 Requisitos do Administrador do Sistema

11. O sistema deve permitir a criação de novos modelos de papeis.

12. O sistema deve fornecer uma visualização da quantidade de vezes que um modelo de papel foi utilizado em um projeto.

13. O sistema deve permitir a criação de novos modelos de responsabilidades.

14. O sistema deve fornecer uma visualização da quantidade de vezes que um modelo de responsabilidade foi utilizado em um projeto.

A.3 Requisitos de Interface Externa

A.3.1 Requisitos de interface de software

O software consiste na integração de dois componentes principais, o *front-end*, que se trata de uma aplicação Web Angular, e o *back-end*, que é uma aplicação Spring em Java.

A.3.2 Requisitos de interface de hardware

A aplicação *front-end* deve ser acessível por meio de qualquer navegador web moderno, como Chrome, Edge, Firefox ou similares. Enquanto o servidor front-end requer uma máquina com Angular CLI e Node.js instalados. O *back-end* requer uma máquina com JRE instalado (Java Runtime Environment).