Pontifícia Universidade Católica
do Rio de Janeiro

Rodrigo Lucas Soares

Desenvolvimento de uma aplicação web para modelagem colaborativa

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Engenharia Civil do Departamento de Engenharia Civil e Ambiental da PUC-Rio.

Orientador: Prof. Luiz Fernando Martha

Rio de Janeiro

Março de 2022

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Lucas Soares

Desenvolvimento de uma aplicação web para modelagem colaborativa

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Engenharia Civil do Departamento de Engenharia Civil e Ambiental da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Luiz Fernando Martha
Orientador
Departamento de Engenharia Civil e Ambiental - PUC-Rio

Prof. Deane de Mesquita Roehl
Departamento de Engenharia Civil e Ambiental - PUC-Rio

Prof. André Maués Brabo Pereira
Departamento de Ciência da Computação – UFF

Dr. Márcio Rodrigues de Santi
Instituto Tecgraf

Rio de Janeiro, 25 de março de 2022

Rodrigo Lucas Soares

Graduou-se em Engenharia Civil na Universidade de Fortaleza em 2020. Iniciou o curso de mestrado em Engenharia Civil (área de concentração em estruturas) na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em 2020. Atua na linha de pesquisa de Computação Gráfica aplicada a Engenharia Civil e no desenvolvimento de ferramentas para modelagem geométrica de sólidos.

Ficha Catalográfica

Soares, Rodrigo Lucas

Desenvolvimento de uma aplicação web para modelagem colaborativa / Rodrigo Lucas Soares; orientador: Luiz Fernando Martha. – 2022.
76 f.: il. color.; 29,7 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Civil e Ambiental, 2022.
Inclui bibliografia

1. Engenharia Civil e Ambiental - Teses. 2. Colaboração. 3. Modelagem. 4. Aplicação como serviço. 5. Web. I. Martha, Luiz Fernando. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Civil e Ambiental. III. Título.

CDD: 624

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Agradeço ao CNPq e a PUC-Rio pelo suporte financeiro concedido durante o período do mestrado.

Ao meu orientador Prof. Luiz Fernando Martha do Departamento de Engenharia Civil e Ambiental da PUC-Rio, agradeço o repasse de conhecimento proporcionado pelas disciplinas do curso de mestrado, bem como durante a orientação deste trabalho. Agradeço, também, pelos conselhos concedidos que foram primordiais para o desenvolvimento e conclusão deste trabalho.

Agradeço à Prof. Deane Roehl, ao Prof. André Pereira e ao Dr. Márcio de Santi pela participação na banca e pelas sugestões e comentários a esta dissertação.

Agradeço aos meus pais, Angélica e Assis, por me incentivarem aos estudos desde o início, pelo amor e por ensinar aos filhos que com dedicação, humildade e honestidade podemos conquistar todos os nossos sonhos.

Agradeço aos meus irmãos, Bruno, Leonardo e Carol, por serem meus exemplos como pessoa e nos estudos.

Agradeço à minha namorada, Suzy, pelo amor, carinho e apoio. Agradeço a compreensão nos dias em que tinha que ficar até mais tarde estudando e escrevendo, e pelas conversas quando estava preocupado com as disciplinas e desenvolvimento desse trabalho.

Aos meus amigos de mestrado, por tornarem essa jornada mais fácil, apesar da distância física ocasionada pela pandemia. Em especial, ao Danilo Bonfim que fizemos disciplinas juntos e que participou e acompanhou esse trabalho desde o início.

Agradeço a Deus, por me acompanhar em todos os momentos.

Resumo

Soares, R. L.; Martha, L. F. **Desenvolvimento de uma aplicação web para modelagem colaborativa**. Rio de Janeiro, 2022. 76p. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho apresenta uma estratégia de modelagem colaborativa na web através do protocolo de comunicação em rede WebSocket. Para compreender a relevância e identificar possíveis lacunas sobre o tema, foi realizada uma investigação acerca do estado da arte de aplicações web colaborativas por meio de uma revisão sistemática da literatura. Para demonstrar a estratégia proposta, foi desenvolvida uma prova de conceito utilizando as linguagens de programação JavaScript e Python. A prova de conceito possui uma arquitetura cliente-servidor, na qual o cliente se comunica com o servidor, sendo este responsável por hospedar a estrutura de dados topológica da aplicação permitindo a criação de modelos bidimensionais de elementos finitos. Devido o canal de comunicação bilateral fornecido pelo WebSocket, é possível a colaboração entre diversos usuários em salas virtuais que compartilham o mesmo modelo em tempo real. Ao final, é realizada uma avaliação da usabilidade da aplicação para verificar a estratégia proposta e encontrar possíveis limitações.

Palavras Chaves

Colaboração, Modelagem, Aplicação como serviço, Web

Abstract

Soares, R. L.; Martha, L. F. **Development of a web application for collaborative modeling**. Rio de Janeiro, 2022. 76p. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

This work presents a collaborative modeling strategy on the web through the WebSocket network communication protocol. To understand the relevance and identify possible gaps on the subject, an investigation was carried out on the state of the art of collaborative web applications through a systematic review of the literature. To demonstrate the proposed strategy, a proof of concept was developed using the JavaScript and Python programming languages. The proof of concept has a client-server architecture, in which the client communicates with the server, which is responsible for hosting the application's topological data structure, allowing the creation of two-dimensional finite element models. Due to the two-way communication channel provided by WebSocket, the collaboration between several users in virtual rooms that share the same model in real-time is possible. In the end, an evaluation of the usability of the application is carried out to verify the proposed strategy and find possible limitations.

Keywords

Collaboration, Modeling, Software as a Service, Web

Sumário

Lista de Figuras

1
Introdução

O constante avanço na infraestrutura de redes de computadores tem proporcionado às novas aplicações o funcionamento em nuvem, atenuando as tarefas de processamento e armazenamento dos computadores pessoais e repassando-as para servidores que, em geral, são máquinas com maior poder computacional. Aliado a isso, devido à globalização de processos entre equipes multidisciplinares surge a necessidade de trabalho remoto e simultâneo. Com isso as aplicações em nuvem acrescentam a possibilidade de trabalho cooperativo em tempo real.

Por outro lado, as aplicações de modelagem geométrica tiveram grande avanço em suas estruturas de dados nos últimos anos, algumas dessas aplicações, devido às suas finalidades, abdicaram de menor armazenamento em memória para possuir ganho em tempo de processamento. Dessa forma, à medida que os modelos são aprimorados para representar de forma fiel a realidade, o armazenamento torna a aplicação menos eficiente para dispositivos com menor capacidade computacional, tendo como exemplo os computadores pessoais de entrada e dispositivos móveis.

A tarefa de modelagem, por sua vez, é realizada por equipes multidisciplinares que de acordo com sua experiência modificam a estrutura do objeto modelado, bem como seus atributos. Dessa maneira, é incomum que a tarefa de modelagem seja executada por apenas uma pessoa. No método tradicional é necessário o compartilhamento de arquivos entre as equipes para que as modificações sejam realizadas, e após isso é necessário realizar a junção de modificações feitas em paralelo.

Portanto, no contexto da tecnologia atual, é notável a importância de que as aplicações de modelagem geométrica possuam uma arquitetura eficiente que

possibilite ao utilizador final uma aplicação leve e funcional e com a capacidade de que múltiplos usuários possam cooperar em tempo real.

De acordo com ZISSIS et al. (2017) as aplicações CAD/CAE tornam-se mais flexíveis quando são inseridas funcionalidades de computação na nuvem. Alguns desses benefícios são:

- Não há necessidade de instalar a aplicação em cada máquina, pois o acesso se dá através do acesso da aplicação no *browser*, e as atualizações serão aplicadas assim que disponível em produção.
- O compartilhamento de arquivos é facilitado.
- Uso do processamento de *workstations* de alto poder computacional na nuvem.

Os serviços de computação em nuvem são oferecidos em três modelos de serviço (Zissis et al., 2017):

- Infraestrutura como Serviço (IaaS): É fornecido armazenamento, processamento e outros recursos computacionais. Nesse modelo o usuário pode realizar o envio de aplicações e possui controle sobre o armazenamento e a aplicação.
- Plataforma como Serviço (PaaS): O usuário possui controle sobre a aplicação enviada ao servidor, porém não possui controle das configurações de arquitetura da infraestrutura exceto algumas variáveis de ambiente.
- Aplicação como Serviço (SaaS): O usuário possui acesso a aplicação fornecida; a aplicação pode ser acessada por vários usuários utilizando diferentes interfaces. Nesse modelo o usuário apenas possui controle das configurações relativas ao próprio usuário.

Com base no exposto, esta dissertação procura estudar metodologias de modelagem geométrica compartilhadas pela *Web*. Para identificar o estado da arte nessa área, foi realizada uma revisão sistemática sobre aplicações de visualização e modelagem com a utilização de arquitetura em nuvem e sistemas *web*. Foi identificada uma escassez de publicações na área de modelagem geométrica e colaboração em tempo real, e neste trabalho é proposta uma estratégia, utilizando o modelo de serviço Aplicação como Serviço, para o desenvolvimento de uma aplicação *web* de modelagem geométrica colaborativa. Trata-se de uma estratégia que utiliza a arquitetura cliente-servidor, na qual a camada cliente é responsável por

possibilitar a interação com o usuário através de uma interface gráfica desenvolvida utilizando a linguagem *JavaScript* e a biblioteca *React*, enquanto a camada do servidor foi desenvolvida utilizando a linguagem Python e é responsável pelo armazenamento e processamento da estrutura de dados, e pelo cruzamento das múltiplas camadas clientes conectadas a uma sala virtual. Para a comunicação entre as camadas cliente e servidor é utilizado o protocolo[1] *WebSocket* através do protocolo TCP (Transmission Control Protocol), para isso são utilizadas as bibliotecas *Socket.io* do lado cliente e a biblioteca *Flask-SocketIO* do lado servidor.

1.1. Objetivo

Esse trabalho possui como objetivo principal o desenvolvimento de uma aplicação *web* colaborativa em tempo real de modelagem geométrica. Como forma de fundamentar e fornecer subsídio para o desenvolvimento da aplicação, este trabalho possui como objetivo secundário: a realização uma revisão sistemática acerca do estado da arte de aplicações colaborativas de modelagem e visualização geométrica.

1.2. Metodologia

Inicialmente é feita uma revisão bibliográfica com intuito de buscar dados e informações sobre os conceitos de redes de computadores. Após isso, é realizada uma revisão sistemática para entender o estado da arte do tema de pesquisa, e encontrar lacunas para conduzir o desenvolvimento da aplicação. Em seguida, são mapeadas as tecnologias para desenvolvimento da aplicação. E, por fim, é realizada o desenvolvimento de uma prova de conceito através de uma aplicação colaborativa de modelagem geométrica em tempo real.

[1] Um protocolo é um padrão de comunicação que possibilita a troca de informações entre aplicações através da internet.

1.3.
Organização da dissertação

Esse trabalho é estruturado em sete capítulos da seguinte forma. No primeiro capítulo apresenta-se o tema de pesquisa, as motivações e os objetivos da pesquisa.

Para ambientar o leitor com os conceitos de redes de computadores e facilitar o entendimento das seções seguintes, no segundo capítulo é realizada uma breve revisão bibliográfica desses conceitos.

O Capítulo 3 apresenta a revisão sistemática relacionada com o tema de pesquisa. Inicialmente são apresentados os critérios para condução da revisão sistemática e em seguida são apresentados os resultados de forma quantitativa e qualitativa.

O Capítulo 4 introduz as tecnologias escolhidas para o desenvolvimento da aplicação. São apresentadas as tecnologias mais importantes com casos simples de uso para demonstrar suas funcionalidades.

O Capítulo 5 descreve a aplicação desenvolvida. Inicialmente é apresentada a arquitetura que promove a funcionalidade de colaboração da aplicação. Em seguida, a interface gráfica da aplicação é detalhada. Por fim, o fluxo de utilização da aplicação é descrito por meio de um exemplo de caso de uso.

O Capítulo 6 apresenta os resultados de avaliação da aplicação que são obtidos através de um questionário respondido por um grupo submetido à sua utilização por meio de exemplos pré-definidos.

No Capítulo 7 são realizadas as considerações finais e propostas para trabalhos futuros.

2
Rede de computadores

Neste capítulo são discutidos os princípios básicos de redes de computadores para o melhor entendimento do restante deste trabalho pelo leitor. Na Seção 2.1 é apresentada a arquitetura de aplicações *web*, na Seção 2.2 é discutida a arquitetura em camadas do modelo TCP/IP que é composta das Seções 2.2.1 e 2.2.2 que discutem as camadas de transporte e de aplicação, respectivamente.

2.1.
Aplicações *Web*

Aplicações *Web* utilizam a arquitetura cliente-servidor, que são formadas por dois processos que se comunicam: o processo cliente funciona no navegador do usuário e o processo servidor reside em um servidor remoto na rede. Dessa forma, as aplicações não se comunicam diretamente entre si, elas se comunicam por meio de um servidor que possui um endereço IP (Internet Protocol) fixo, que possibilita ser encontrado pelas aplicações clientes, conforme ilustrado na Figura 2.1 (Kurose & Ross, 2013).

As mensagens trocadas por esses processos são enviadas passando por camadas, que serão discutidas na próxima seção. Um processo recebe e envia mensagens para a rede através de uma interface chamada de *socket*, que serve como uma interface de programação entre as camadas da aplicação e de transporte, conforme mostra o esquema da Figura 2.2. Dessa forma, as requisições e respostas passam da camada de transporte à camada de aplicação através dos *sockets*. Os desenvolvedores têm pouco acesso à programação da camada de transporte, porém possuem maior liberdade nas escolhas para a implementação na camada de aplicação (Kurose & Ross, 2013).

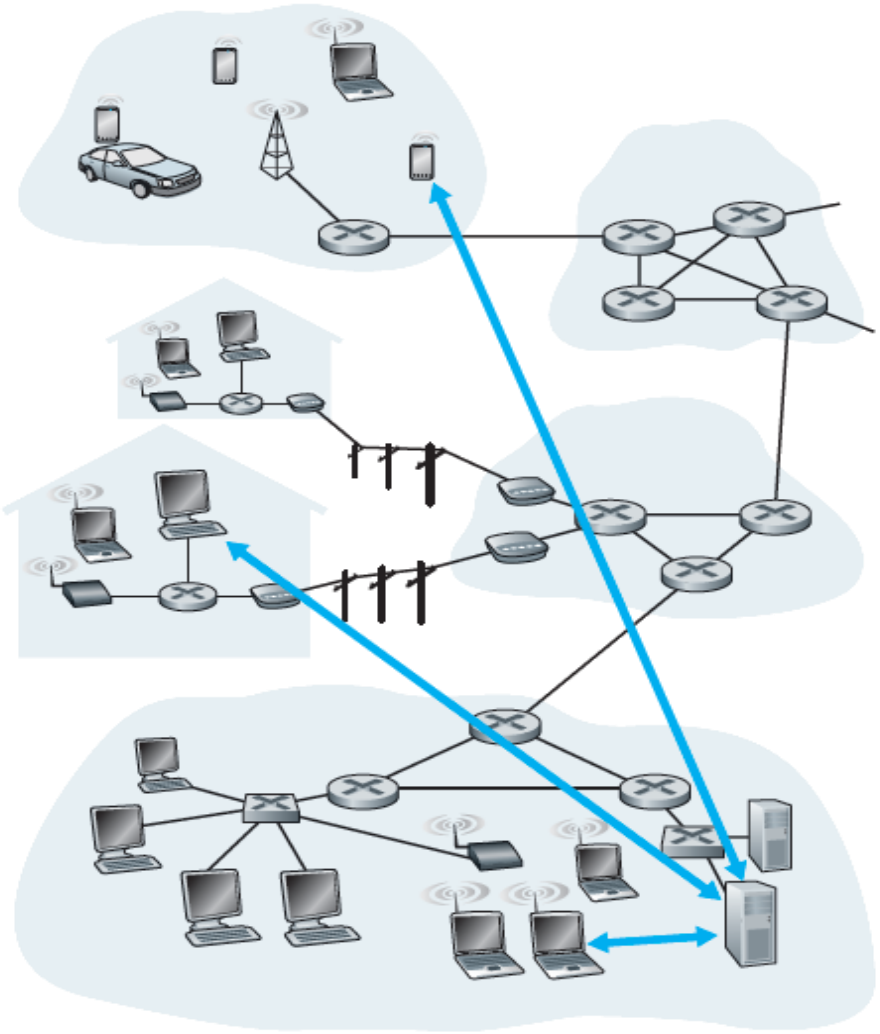


Figura 2.1 – Arquitetura cliente-servidor (Kurose & Ross, 2013).

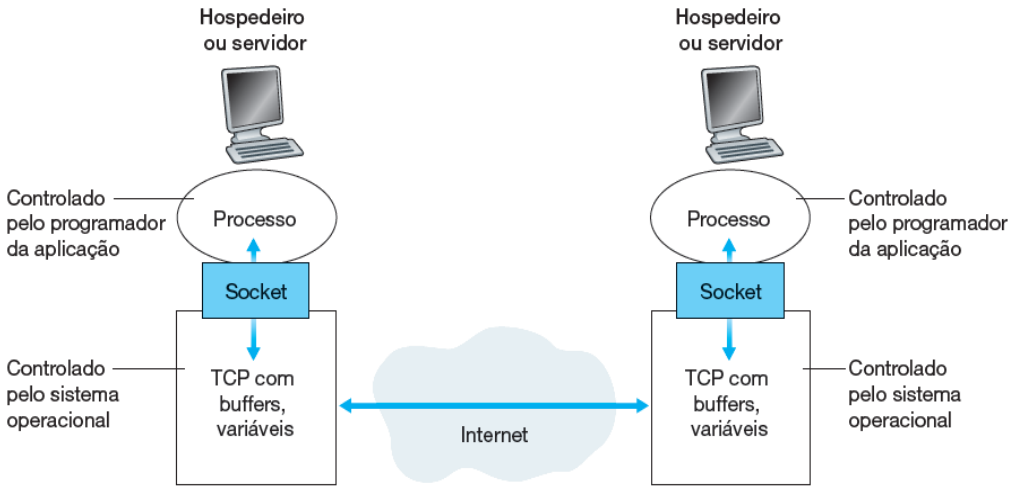


Figura 2.2 – Interface de programação da aplicação *socket* (Kurose & Ross, 2013).

2.2.
Arquitetura de camadas

Segundo Kurose & Ross (2013), os modelos de rede são organizados em arquitetura de camadas, e cada camada possui protocolos específicos que atuam sobre ela. Essas camadas são organizadas em forma de pilha de tal maneira que uma camada um nível acima utiliza serviços fornecidos pela camada de nível abaixo.

O modelo TCP/IP, também conhecido por pilha de protocolos, é dividido em quatro camadas (aplicação, transporte, internet e enlace), assim como mostrado na Figura 2.3. Na mesma imagem são ilustrados também alguns dos protocolos que podem ser utilizados em cada camada. A camada de aplicação é a camada de mais alto nível, ou seja, a camada mais próxima do usuário e a camada de enlace é a camada de mais baixo nível, ou seja, a camada mais distante do usuário.

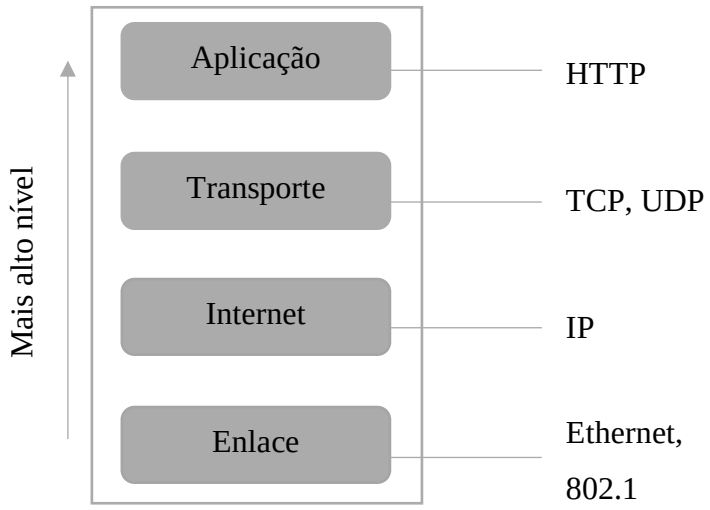


Figura 2.3 – Pilha de protocolos.

Este trabalho limita-se a discutir apenas as camadas de aplicação e de transporte por serem as mais necessárias ao entendimento da escolha das tecnologias utilizadas.

2.2.1.
Camada de transporte

Há dois processos que podem atuar na camada de transporte, o protocolo UDP (*User Defined Protocol)* e o protocolo TCP (*Transmission Control Protocol*). Esses protocolos definem padrões de comunicação entre duas aplicações cliente-servidor.

Segundo Kurose & Ross (2013), o protocolo UDP é um protocolo simplificado que não necessita de uma apresentação inicial (*hand-shake)*, com isso permite que as aplicações enviem dados sem a necessidade de estabelecer uma conexão. Dessa forma, o protocolo UDP é não orientado à conexão e não confiável pois não há garantia de que os dados serão entregues.

De forma oposta ao protocolo UDP, o protocolo TCP é um protocolo orientado à conexão e seguro para transferência de dados. É necessário realizar inicialmente o *hand-shake* entre as aplicações (cliente e servidor) o que irá fornecer um canal para a transferência de dados entre as aplicações. Esse canal é *full-duplex* visto que ambas as aplicações podem enviar dados entre si. Além disso, o protocolo TCP garante que os dados serão entregues da mesma forma em que foram enviados e na ordem correta – diferentemente do protocolo UDP (Kurose & Ross, 2013).

2.2.2.
Camada de aplicação

A seguir são apresentados os protocolos que definem os padrões de transporte de dados na camada de aplicação. Os protocolos estudados são os protocolos HTTP (*HyperText Transfer Protocol*) e *WebSocket*.

2.2.2.1.
Protocolo HTTP

O protocolo HTTP define a forma na qual as aplicações clientes enviam requisições aos servidores, bem como a forma na qual os servidores enviam respostas às aplicações clientes (Kurose & Ross, 2013). O HTTP utiliza o TCP como protocolo de transporte, dessa forma inicialmente o cliente realiza um *hand-*

shake com o servidor para após isso ser possível realizar as suas requisições e receber as respostas do servidor. O HTTP possui métodos padrões para as requisições, entre eles: *GET*, *POST, HEAD, PUT* e *DELETE*.

O método *GET* é utilizado quando o cliente pretende apenas requisitar um objeto. O método *POST*, por sua vez, é utilizado quando o cliente pretende requisitar um objeto que dependa de algumas informações, como por exemplo o preenchimento de um formulário, na qual o usuário preenche algumas informações e requisita um objeto que necessita dessas informações, que são enviadas na requisição. O método *HEAD* é semelhante ao *GET*, porém o servidor não envia o objeto requisitado, esse por sua vez é utilizado para depuração. O método *PUT* é utilizado quando o cliente pretende carregar um objeto no servidor e o método *DELETE* é utilizado quando o cliente pretende eliminar um objeto de um caminho específico.

As requisições e respostas possuem um formato específico onde são especificadas as informações necessárias para que as requisições sejam bem-sucedidas. Na Figura 2.4 é exemplificada uma mensagem de requisição.

```
GET: /diretório/página.html HTTP/1.1
Host: www.endereco.com
Connection: close
User-agent: navegador/versão_do_navegador
Accept-language: pt-br
```

Figura 2.4 – Exemplo de mensagem de requisição HTTP.

A primeira linha da requisição mostrada na Figura 2.4 é chamada de linha de requisição. Essa especifica o método utilizado, a URL na qual é especificada a requisição e a versão do protocolo. As linhas que seguem são denominadas linhas de cabeçalho – podem conter outros campos, e podem não conter alguns dos campos do exemplo, dependendo da requisição. O campo *host* indica o endereço do servidor (hospedeiro), o campo *connection* informa que o cliente deseja encerrar a conexão com o servidor quando a resposta for recebida, e os demais campos informam, respectivamente, o navegador do qual a requisição fora enviada e sua versão e a língua em que o cliente quer receber o objeto solicitado, caso exista.

A Figura 2.5 mostra um exemplo de uma requisição de um arquivo através do protocolo HTTP.

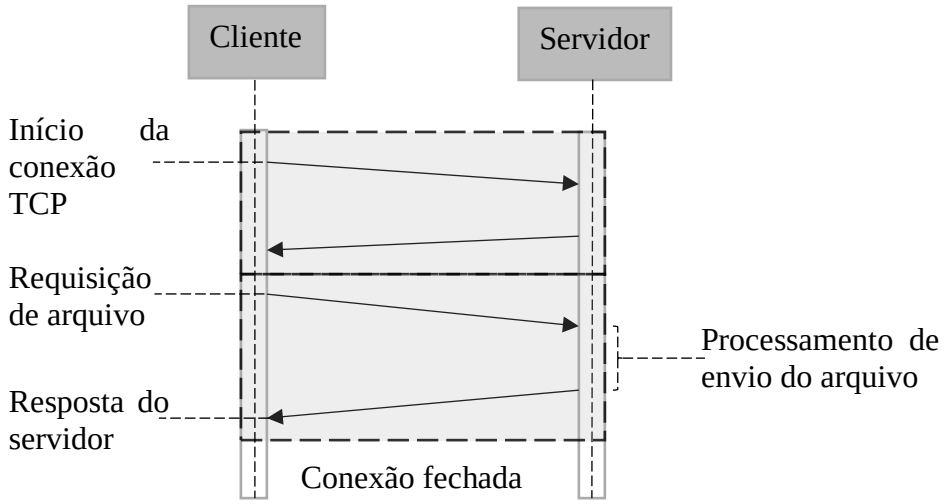


Figura 2.5 – Fluxograma requisição HTTP.

2.2.2.2. *WebSocket*

O *WebSocket*, por sua vez, provê um canal bidirecional *full-duplex* que opera sobre o HTTP através de um único *socket* (Lombardi, 2015).

Inicialmente a aplicação cliente realiza uma apresentação inicial, chamada *hand-shake*, com a aplicação servidor para iniciar a comunicação através do *WebSocket*. Esse primeiro contato é realizado através do método GET padrão do HTTP contendo no cabeçalho da requisição o pedido de *upgrade* da conexão para *WebSocket*. De forma a exemplificar, é mostrado na Figura 2.6 – utilizando como exemplo o caso da aplicação desenvolvida neste trabalho – o cabeçalho contendo os itens obrigatórios, para que a conexão através do *WebSocket* seja estabelecida.

```
GET: wss://half-edge-apy.herokuapp.com/socket.io/?EIO=4&transport=websocket
Origin: https://curve-collector.herokuapp.com
Sec-WebSocket-Key: tUZmdn6HzY1GKYV18nxe8A==
Sec-WebSocket-Version: 13
Upgrade: websocket
Connection: Upgrade
Host: half-edge-apy.herokuapp.com
```

Figura 2.6 – Requisição HTTP com *upgrade* para *WebSocket.*

Caso os itens acima não estejam presentes na requisição, o servidor irá retornar com o código HTTP 400 (*bad-request)*. De outra forma, caso os itens estejam conforme o esperado, o servidor irá retornar com o código HTTP 101 (*switching protocols)*. O cabeçalho da resposta para a aplicação discutida nesse trabalho é mostrada na Figura 2.7 como forma de exemplificação.

```
HTTP 101 Switching Protocols
Sec-WebSocket-Accept: tUZmdn6HzY1GKYV18nxe8A==
Upgrade: websocket
Connection: Upgrade
```

Figura 2.7 – Resposta da requisição HTTP para *WebSocket*.

A Figura 2.8 mostra o fluxograma para início da conexão através do protocolo *WebSocket*. Nesse caso, o exposto acima é realizado na fase de *hand-shake* e, após isso, o cliente e o servidor possuem um canal bidirecional para realizar o envio de dados.

É importante ressaltar ainda, que diferente do protocolo HTTP, o *WebSocket* só irá realizar o envio de cabeçalho na etapa de *hand-shake*, ou seja, as próximas requisições e respostas – no canal conectado - não irão necessitar de cabeçalho, o que significa uma diminuição no tamanho das requisições. Isso é um ponto importante para aplicações que necessitam de respostas mais ágeis, visto que com a diminuição do tamanho das requisições os dados podem ser enviados e recebidos de forma mais rápida. Esse canal fica aberto até que alguma das partes solicite que a conexão seja fechada.

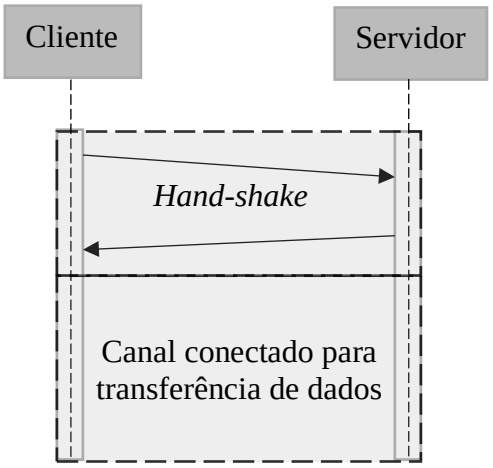


Figura 2.8 – Modelo de comunicação entre cliente e servidor através de *WebSocket*.

3
Revisão Bibliográfica

Este capítulo descreve a metodologia adotada para conduzir a revisão bibliográfica, possibilitando identificar estudos científicos relevantes para a condução deste trabalho. Através de uma revisão sistemática foi possível identificar lacunas existentes no estado da arte, direcionando a pesquisa com intuito de cobrir as lacunas existentes.

3.1.
Metodologia

De acordo com Gough et al. (2012), realizar uma revisão bibliográfica de forma sistemática requer três atividades chaves: identificar e descrever pesquisas relevantes; avaliar criticamente a pesquisa de forma sistemática; e realizar a junção das descobertas em uma síntese.

Como metodologia para realizar a revisão bibliográfica foi adotada a revisão sistemática de literatura. A revisão sistemática difere da revisão tradicional ao adotar um modelo imparcial, confiável e replicável; permitindo, também, identificar campos de estudo inexplorados (Tranfield et al., 2003)

Uma revisão sistemática é realizada em quatro passos: o primeiro passo é formular uma pergunta com o intuito de estabelecer o foco da pesquisa; o segundo passo é localizar pesquisas relevantes através de buscas em bancos de dados com uma frase de busca definida; o terceiro passo é a seleção e avaliação que sejam relevantes ao contexto das questões levantadas; e o quarto passo é realizar a análise e uma síntese dos trabalhos selecionados (Tranfield et al., 2003)

As próximas seções delimitam e descrevem os passos, conforme descrito anteriormente, de forma que a revisão seja imparcial e replicável. Após isso são

apresentados os dados obtidos da etapa de busca e seleção de trabalhos e uma síntese dos trabalhos selecionados.

3.1.1. Desenho do estudo

Para guiar este trabalho foram formuladas as seguintes questões acerca de softwares de modelagem geométrica e análises estruturais colaborativos e em ambiente *web*:

1. Softwares de modelagem geométrica e análise estrutural podem ser utilizados de forma eficaz em ambientes baseados na *web*?
2. Como pode ser inserida a funcionalidade de colaboração em tempo real nessas aplicações?
3. Quais os requisitos para que esses programas possam ser adotados por um público maior?

3.1.2. Estratégia de busca

A partir da delimitação do foco da revisão bibliográfica, foi elaborada a frase para realizar a busca nos bancos de dados. Para elaboração da frase de busca montou-se uma hierarquia com os temas relevantes conforme mostra a Figura 3.1. As palavras contidas no nível 1 são os temas principais e serão combinados com o operador booleano AND, já as palavras contidas no nível 2 serão combinadas com o operador OR.

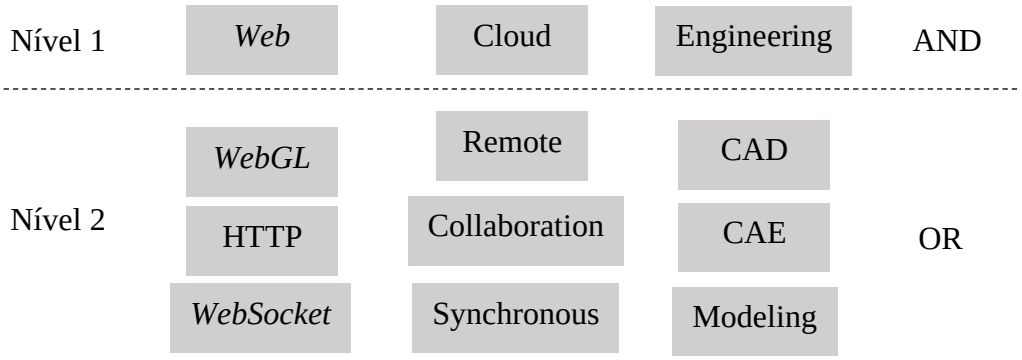


Figura 3.1 – Hierarquia de palavras adotadas.

Para englobar as possibilidades de aplicações *web* de engenharia sem colaboração e aplicações *web* de engenharia com colaboração foram elaboradas duas frases para realizar a busca nas bases de dados (Scopus, Engineering Village e *Web* of Science):

1. *Web* AND (*WebGL* OR HTTP OR *WebSocket*) AND Engineering AND (CAD OR CAE OR Modelling).
2. *Web* AND (*WebGL* OR HTTP OR *WebSocket*) AND Cloud AND (Collaboration OR Remote OR Synchronous) AND Engineering AND (CAD OR CAE OR Modelling).

3.1.3. Seleção de trabalhos e avaliação quantitativa

O próximo passo após a criação das expressões de busca consiste em aplicá-las nas bases de dados escolhidas e aplicar os filtros e condições de inclusão e exclusão.

Como critérios para inclusão, foram considerados trabalhos publicados em revistas no período de 2015 a 2021 e escritos em língua inglesa que tratavam sobre visualização e/ou modelagem utilizando sistemas voltados para *web*. Foram excluídos os trabalhos que não foram publicados em revistas (dissertações, teses, artigos de congressos, entre outros), que não abordavam aplicações *web*, e trabalhos que não tratavam sobre aplicações de modelagem ou visualização, independente da tecnologia de visualização utilizada. Além disso, vale ressaltar que as expressões

de busca foram pesquisadas nas bases de dados sobre o título, resumo e palavras-chave. A Tabela 1 apresenta um resumo dos critérios de inclusão e exclusão.

Tabela 1 – Critérios de inclusão e exclusão.

Inclusão	**Exclusão**
Artigos de revistas	Outros tipos de publicação
Publicados entre 2015 e 2021	Fora do período
Língua inglesa	Outras línguas
Visualização e/ou modelagem *web*	Artigos que não tratavam sobre modelagem ou visualização *web*

Com o intuito de mostrar a evolução da quantidade de artigos antes e após a aplicação dos filtros e condições de inclusão e exclusão, a Tabela 2 mostra os números de trabalhos encontrados nas bases de dados para cada expressão de busca com e sem os critérios de inclusão e exclusão.

Tabela 2 – Evolução da quantidade de trabalhos com aplicação dos critérios de exclusão.

Expressões:	**Expressão 1**		**Expressão 2**	
Base de dados	Sem critérios de inc./exc.	Com critérios de inc./exc.	Sem critérios de inc./exc.	Com critérios de inc./exc.
Scopus	197	33	3	1
Eng. Village	574	75	39	6
Web of Science	175	40	0	0

Dessa forma, já é possível ter informações acerca da baixa quantidade de trabalhos de *softwares* de modelagem e análise que utilizam sistemas *web* e funcionalidades colaborativas em nuvem.

Os artigos que resultaram da busca com os critérios de inclusão e exclusão, tanto para a expressão 1 quanto para a expressão 2, foram inicialmente ajustados para remover possíveis duplicações, e em seguida foi realizado o próximo passo, que consistiu na leitura dos títulos e resumos para enquadrar os trabalhos relevantes.

Consequentemente, esse passo, descrito na subseção a seguir, foi realizado de forma qualitativa.

3.1.4.
Avaliação qualitativa.

A pesquisa proposta neste trabalho limita-se a trabalhos de modelagem geométrica e análises de engenharia utilizando sistemas *web* com ou sem possibilidade de colaboração simultânea.

Conforme descrito anteriormente, os trabalhos selecionados na busca das bases de dados foram avaliados de forma qualitativa para remover os que não são relevantes à pesquisa proposta.

Devido a similaridades de alguns trabalhos de outras áreas do conhecimento optou-se por enquadrar alguns trabalhos da área médica por utilizarem arquiteturas e tecnologias similares, e desenvolverem aplicações semelhantes aos trabalhos buscados, apesar de alguns não serem aplicações de modelagem geométrica e análise, mas de visualização *web* e/ou colaborativa. Os resultados dessa etapa contemplando a exclusão de artigos duplicados é disposta na Tabela 3.

Tabela 3 – Remoção de trabalhos duplicados e análise qualitativa.

Base de dados	Expressão de busca	Após remoção de duplicação	Artigos escolhidos após leitura do resumo
Scopus	34	122	22
Eng. Village	81		
***Web* of Science**	40		

Após a análise qualitativa realizada através da análise do título e resumo dos artigos é realizada a leitura completa dos artigos selecionados. A Tabela 4 dispõe os artigos selecionados contendo as informações de título e autores.

Tabela 4 – Artigos selecionados.

Nº	Título	Autores
1	3D model management for e-commerce	(Diez et al., 2017)
2	3D visualization for building information models based upon IFC and *WebGL* integration	(Xu et al., 2016)
3	A *web*-based collaborative framework for facilitating decision making on a 3D design developing process	(Nyamsuren et al., 2015)
4	A *Web*-based system enabling the integration, analysis, and 3D sub-surface visualization of groundwater monitoring data and geological models	(Hunter et al., 2015)
5	Application of 3D Printing and *WebGL*-Based 3D Visualization Technology in Imaging Teaching of Ankle Joints	(Li et al., 2021)
6	Atomistic modelling of scattering data in the collaborative Computational Project for Small Angle Scattering (CCP-SAS)	(Perkins et al., 2016)
7	Bottled SAFT: A *Web* App Providing SAFT-γ Mie Force Field Parameters for Thousands of Molecular Fluids	(Ervik et al., 2016)
8	CardiacPBPK: A tool for the prediction and visualization of time-concentration profiles of drugs in heart tissue	(Tylutki et al., 2019)
9	CCBuilder 2.0: Powerful and accessible coiled-coil modelling	(Wood & Woolfson, 2018)
10	Collaborative CAD/CAE as a cloud service	(Zissis et al., 2017)
11	Enabling Collaborative Numerical Modelling in Earth Sciences using Knowledge Infrastructure	(Bandaragoda et al., 2019)
12	EzCADD: A Rapid 2D/3D Visualization-Enabled *Web* Modelling Environment for Democratizing Computer-Aided Drug Design	(Tao et al., 2019)
13	From video games to solar energy: 3D shading simulation for PV using GPU	(Robledo et al., 2019)
14	Implementation of BIM plus *Web*GIS Based on Extended IFC and Batched 3D Tiles Data: An Application in RCC Gravity Dam for Republication of Design Change Model	(Zhang & Jiang, 2021)
15	Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with *web*-based tools	(Hu et al., 2016)
16	Interactive *WebGL*-based 3D visualizations for EAST experiment	(Xia et al., 2016)

17	Non-linear least squares fit of specific heat data within the Schotte–Schotte model using *web* page	(Goraus, 2019)
18	Rendering Optimization for Mobile *Web* 3D Based on Animation Data Separation and On-Demand Loading	(Li et al., 2020)
19	TouchTerrain: A simple *web*-tool for creating 3D-printable topographic models	(Hasiuk et al., 2017)
20	Using artificial neural network and *WebGL* to algorithmically optimize window wall ratios of high-rise office buildings	(Zhao, 2021)
21	*Web*-Based Applications to Simulate Drinking Water Inorganic Chloramine Chemistry	(Wahman, 2018)
22	*Web*-based visualization of 3D factory layout from hybrid modelling of CAD and point cloud on virtual globe DTX solution	(Salehi & Wang, 2019)

Após a leitura completa dos artigos foi constatado que alguns artigos não se adequavam completamente ao escopo delimitado anteriormente, ou seja, não abordavam tecnologias com arquiteturas *web* e com visualização através do *WebGL* ou de outra tecnologia que permitisse a visualização através dos navegadores. Os títulos desses artigos são dispostos na Tabela 5.

Tabela 5 – Títulos excluídos após leitura completa.

Nº	**Título**	**Autores**
1	Atomistic modelling of scattering data in the collaborative Computational Project for Small Angle Scattering (CCP-SAS)	(Perkins et al., 2016)
2	Bottled SAFT: A *Web* App Providing SAFT-γ Mie Force Field Parameters for Thousands of Molecular Fluids	(Ervik et al., 2016)
3	CardiacPBPK: A tool for the prediction and visualization of time-concentration profiles of drugs in heart tissue	(Tylutki et al., 2019)
4	Non-linear least squares fit of specific heat data within the Schotte–Schotte model using *web* page	(Goraus, 2019)
5	*Web*-Based Applications to Simulate Drinking Water Inorganic Chloramine Chemistry	(Wahman, 2018)

3.2.
Síntese

Nessa seção os artigos selecionados são dispostos em forma de síntese conectando-os a pontos importantes do tema central desse trabalho quando necessário. Para o melhor entendimento da conexão dos artigos aos pontos relevantes com relação a esse trabalho, os artigos foram separados em três classes: aplicações *web* de visualização e/ou modelagem de outras áreas de conhecimento, aplicações *web* de visualização e/ou modelagem na área de engenharia, e aplicações *web* de visualização e/ou modelagem com sistema de colaboração remota em tempo real.

3.2.1.
Aplicações de outras áreas

Diversas áreas do conhecimento como medicina, química e biotecnologia possuem conceitos de difícil visualização utilizando metodologias tradicionais. Portanto, para o completo entendimento da disciplina em estudo, ou para que os resultados de pesquisas sejam analisados corretamente, são necessárias ferramentas adequadas para visualização e/ou modelagem. Diversos autores, realizaram pesquisas de forma a levantar requisitos para tais ferramentas e elaborando provas de conceito.

Com a intenção de criar uma aplicação acessível para usuários não-especialistas, Wood & Woolfson (2018) desenvolveram uma aplicação (CCBuilder) utilizando a arquitetura *browser*-servidor para desenvolvimento de estruturas superhélice. Estruturas superhélice são amontoados de feixes de uma cadeia de aminoácidos estabilizados por pontes hidrogênio chamados α-hélice (Lupas & Gruber, 2005). A camada de *front-end* foi desenvolvida utilizando Elm, *JavaScript*, HTML e CSS para permitir a visualização dos modelos. Através do protocolo HTTP a camada *front-end* realiza a comunicação com o servidor e banco de dados desenvolvidos utilizando a linguagem Python e o *framework Flask,* e MongoDB para o banco de dados.

Utilizando um visualizador de proteínas (NGL), desenvolvido utilizando a biblioteca Three.js que provê uma interface ao *WebGL*, Tao et al. (2019) desenvolveram uma aplicação *Computer-aided Drug Design (CADD)* com funcionalidade *Web* para auxiliar estudantes e cientistas na descoberta e desenvolvimento de novos medicamentos. A aplicação foi testada com 95 alunos do primeiro ano do curso de Farmácia através da aplicação de questionários e uma atividade prática. Após o fim da avaliação o nível de experiência em modelagem de medicamentos, dos estudantes que responderam os questionários, foi aperfeiçoado de zero ou pouca experiência (64% dos estudantes) para pouca ou boa experiência (79% dos estudantes), 84% dos estudantes conseguiram finalizar completamente a atividade proposta e 14% completaram parcialmente. Além disso, 88% dos estudantes consideraram o aplicativo de fácil utilização e amigável ao usuário.

Com intuito de facilitar e aprimorar o entendimento de estudantes de medicina com relação à estrutura anatômica de ligações do tornozelo, Li et al. (2021) desenvolveram uma metodologia de ensino utilizando modelos impressos 3D e um sistema *web* para visualização e iteração com modelos 2D e 3D. Os modelos são gerados a partir de imagens escaneadas por ressonância magnética, e para a visualização foi utilizada a ferramenta *WebGL* possibilitando ao estudante rotacionar, aplicar escalas, esconder e aplicar transparência ao modelo. Para avaliar a eficácia da metodologia utilizada foi realizado um estudo com grupos de estudantes. Os autores puderam concluir que a utilização dos modelos impressos e visualização 3D aumentaram o interesse e entusiasmo dos estudantes na disciplina, bem como uma significativa melhora na compreensão acerca da disciplina ministrada.

3.2.2. Aplicações de engenharia sem colaboração

Xia et al. (2016) aprimoraram um ambiente virtual para o Tokamak Supercondutor Experimental Avançado (EAST, da sigla em inglês), o qual permite que cientistas e engenheiros possam visualizar a estrutura interna, componentes de diagnóstico e meta dados durante os experimentos. O sistema antigo, baseado em VRML/Java 3D, foi aprimorado utilizando as tecnologias *web WebGL* e HTML5,

permitindo a utilização do sistema através de computadores, computadores portáteis e dispositivos móveis. A arquitetura do sistema consiste basicamente em um cliente, um servidor *web*, banco de dados e um módulo conversor. Os autores pretendem futuramente diminuir o tempo de carregamento e renderização.

Além de permitir a visualização, as placas gráficas, devido ao seu alto poder de paralelização, são ótimas para realizar cálculos matemáticos. Levando isso em consideração, Robledo et al. (2019) realizaram um estudo para a avaliação da perda de energia de placas solares devido ao sombreamento por objetos circundantes. Para isso, foi utilizado o processo de rasterização oferecido pelas placas gráficas como forma de simular e visualizar sombras que resultam do bloqueio da luz solar. Os autores desenvolveram um sistema baseado na *web* utilizando a tecnologia de *WebGL* como alternativa para realizar os cálculos de sombreamento e posteriormente realizar a conversão do sombreamento em perda de energia. Foram realizados três estudos de caso em edificações existentes. Através dessa metodologia os autores constataram com sucesso que a utilização do processo de rasterização das placas gráficas é possível. Além disso, a utilização de tecnologias *web* permite ao aplicativo que este funcione localmente no *browser* do usuário sem necessidade de instalação de *softwares* adicionais, bem como oferecer maior interatividade possibilitando a utilização de bibliotecas 3D disponibilizadas online.

Hasiuk et al. (2017), desenvolveram uma aplicação com arquitetura *browser*-servidor para a criação de modelos de terrenos para impressão 3D. Ao desenvolver a aplicação os autores possuíam a intenção de ultrapassar as dificuldades técnicas para que mais pessoas pudessem utilizar o sistema em salas de aulas. A arquitetura da aplicação é composta por uma camada servidor desenvolvido em Python para realizar processamentos pesados que não seriam eficientes na camada de *front-end*, e uma camada cliente desenvolvida em *JavaScript*. A arquitetura proposta é funcional, porém os autores propõem adicionar novas funções para melhorar performance.

Na medida em que os modelos se tornam mais complexos, buscando retratar de forma mais significativa a realidade, os arquivos desses modelos resultam em arquivos maiores e mais pesados para a comunicação *web* e renderização. Dessa forma, Diez et al. (2017) propuseram uma metodologia para aprimorar o gerenciamento multimídia de modelos 3D complexos, por exemplo os modelos de

CAD e BIM, em um ambiente *web* facilitando o desenvolvimento de serviços de e-commerce que contenham catálogos com conteúdo 3D. Os autores pontuaram diversas dificuldades para a implementação, entre elas os formatos CAD/BIM não estarem entre os formatos padrões para *web*, e os arquivos paramétricos ao serem transformados para malhas poligonais tornarem-se grandes demais para serem utilizados com eficiência em navegadores. Além disso, o maior empecilho, segundo os autores, é o tempo de transferência, e por isso foram utilizados algoritmos de redução de malha. A plataforma é baseada na arquitetura *browser*-servidor e utiliza o protocolo SOAP de mensagens. Os autores testaram a plataforma com modelos 3D gerados em diferentes localidades, bem como através de conexões a cabo e via tecnologia 3G de dispositivos móveis.

Devido às limitações de dispositivos com baixo poder computacional, como os dispositivos móveis, a metodologia tradicional de carregamento de modelos 3D pesados em plataformas *web* é lenta e podem ocasionar o travamento do *browser*. Dessa forma, em outro estudo, Li et al. (2020) propuseram uma metodologia de separação de modelo e animação, e carregamento sob demanda de modelos 3D. Os autores utilizaram o formato JSON 3D (*JavaScript* Object Notation 3D - JD), que se trata de um formato mais leve em comparação aos demais formatos padrões para modelos 3D, e, além disso, otimizaram o método de carregamento do *WebGL* para realizar o carregamento assíncrono do modelo 3D.

Outros autores desenvolveram pesquisas voltadas à indústria de arquitetura e construção civil (AEC). Utilizando redes neurais artificiais, Zhao (2021) propôs a otimização da razão entre janelas e paredes de edificações sem a necessidade de realizar simulações, que, de acordo com o próprio autor, consomem maior tempo para conclusão. Após a execução da rede neural ser concluída, os dados de saída precisam ser apresentados de forma fácil para que arquitetos possam dar continuidade à tomada de decisões, dessa forma foi desenvolvido um visualizador *web* utilizando a biblioteca Three.js.

Em outro estudo, Salehi & Wang (2019) propuseram um fluxo de trabalho para gerar e visualizar o *layout* 3D de fábricas no qual um modelo de nuvem de pontos é combinado com objetos CAD de novos equipamentos de manufatura. Para a geração do modelo CAD foi utilizado o formato glTF, que, segundo os autores, é

projetado para ser leve o suficiente para minimizar as demandas de processamento e renderização em um *browser*.

Quanto maior e mais complexos os projetos se tornam, maior é a necessidade de colaboradores especializados em áreas diferentes. À medida que cada empresa desenvolve seus *softwares* e formatos de arquivos compatíveis, torna difícil que todos os participantes do projeto utilizem os mesmos modelos sem que haja falha de compatibilidade. Dessa forma, diversos autores realizaram pesquisas acerca da utilização do formato IFC (Industry Foundation Classes) e de arquiteturas *browser*-servidor e cliente-servidor como forma de promover aplicações multiplataformas, compatíveis com diversos formatos de arquivos e sem necessidade de instalação de bibliotecas de terceiros.

Baseado nos padrões IFC e *WebGL*, Xu et al. (2016) desenvolveram uma metodologia para habilitar a visualização 3D de modelos BIM (Building Information Modeling) em um *browser web*. Inicialmente, o modelo IFC é convertido em um arquivo OBJ, posteriormente o arquivo OBJ é combinado com o código *WebGL* para facilitar a visualização no *browser*. A plataforma é dividida em três camadas: embasamento, apresentação, aplicação. Na camada de embasamento é aplicada um banco de dados para os modelos BIM. Na camada de apresentação, o *WebGL* é aplicado para extrair as informações dos modelos a partir dos arquivos IFC. E, por fim, a camada de aplicação é desenvolvida para aprimorar a interação e experiência de usuário. A aplicação desenvolvida é aplicável em planejamentos de fluxos de trabalho, particularmente em ambientes com múltiplos usuários, múltiplas aplicações BIM e ambientes em tempo real, que necessitam que os arquivos exportados sejam visualizados e compartilhados facilmente (Xu et al. 2016).

Em outro estudo, Hu et al. (2016) propuseram uma nova abordagem utilizando o padrão IFC e ambientes cliente-servidor e *browser*-servidor para melhorar a interoperabilidade e promover colaboração central e remota para usuários espalhados geograficamente. A abordagem proposta pelos autores é baseada na Informação Unificada do Modelo, que, segundo os autores, evita mudanças na estrutura de dados de ferramentas estruturais de *softwares* comerciais e habilita a conversão bidirecional entre diversos *softwares* comerciais. Para comunicação e visualização na arquitetura *browser*-servidor, que para modelos grandes e complexos pode acarretar perda de performance, são utilizados arquivos

em formato JSON e transmissão de dados *web* baseados em compressão. Os modelos são enviados pelos usuários para o servidor, onde serão convertidos utilizado a abordagem descrita anteriormente, após a conversão os dados são enviados novamente para o cliente em formato JSON. A aplicação foi avaliada em quatro projetos reais, demostrando performance satisfatória em termos de qualidade de conversão, acurácia e latência.

Zhang & Jiang (2021) desenvolveram em seu estudo uma plataforma para realizar exportação de arquivos IFC baseado no *software* Revit da Autodesk e intercâmbio entre IFC e o formato *batched 3D tiles*. Como o padrão IFC não possui suporte para elementos de hidroelétricas, os autores realizaram uma extensão do formato IFC. Para visualização é utilizada a biblioteca Celsium.js, uma biblioteca *JavaScript* baseada no *WebGL* para renderezar elementos GIS (Geographic Information System). A plataforma é dividida em três camadas: dados, lógica e apresentação. A camada de dados é baseada em banco de dados e repositório eletrônico de documentos. A camada de lógica é responsável por conectar a camada de apresentação à camada de dados, bem como hospedar as ferramentas de conversão. A camada de apresentação utiliza dois *frameworks web*, LayUI.js para criação da interface de usuário e Cesium.js para visualizar os modelos. Os autores comprovaram a eficiência do sistema proposto através de um estudo de caso real por meio de uma estação hidrelétrica real construída em concreto.

3.2.3. Aplicações com sistema de colaboração remota

As ferramentas CAD e de gestão de ciclo de vida de projetos tradicionais possuem várias limitações em fornecer serviços que sustentem as demandas imediatas do mercado e de ambientes colaborativos (Nyamsuren et al., 2015). Dessa forma, arquiteturas *web* são de extrema importância para fornecer plataformas multidisciplinares, com interfaces de fácil utilização e multiplataformas. Alguns autores realizaram pesquisas acerca da criação de ambientes colaborativos com usuários geograficamente dispersos para aprimorar o desenvolvimento de produtos.

Nyamsuren et al. (2015) desenvolveram um ambiente *web* colaborativo para facilitar a tomada de decisão em processos de desenvolvimento de produtos 3D. A

arquitetura consiste em três módulos responsáveis pela conversão de formatos, armazenamento de modelos e comunicação em tempo real (sistema de mensagens e vídeo conferência). A aplicação também possui integrado um sistema de anotação e de comparação de modelos realizada através de diferença de formas (*Constructive Solid Geometry*) para auxiliar no processo de decisão. Além disso, o protocolo *WebSocket* é utilizado para promover a comunicação bilateral em tempo real. Através da aplicação desenvolvida foi possível realizar a discussão de modelos simples, porém ainda devem ser realizadas melhorias para a utilização em modelos mais complexos (Nyamsuren et al., 2015).

Em outro estudo, Zissis et al. (2017) realizaram a implementação de uma arquitetura nomeada CAD/CAE as a Service (CaaS) com intuito de superar as limitações de performance, interoperabilidade, modularidade, escalabilidade e interatividade com a interface de usuário. A interface de usuário é desenvolvida utilizando a linguagem HTML5 e o *framework* Bootstrap, permitindo a criação de interface responsiva para diferentes tipos de dispositivos, inclusive dispositivos móveis. Para permitir a visualização dos modelos 3D é utilizada a biblioteca Three.js como interface ao *WebGL*. A comunicação em rede é construída por meio de diversos serviços promovidos por meio do protocolo HTTP. Como forma de validação, os autores submeteram profissionais do ramo de manufatura para utilizar o sistema desenvolvido e responder um questionário para avaliar a performance da aplicação em condições reais. Através da validação dentre outros pontos, é mencionada a limitação de o sistema não possuir suporte para visualização e anotação simultânea, ou seja, quando um usuário modificar o modelo os outros usuários não são notificados da mudança imediatamente.

Como conclusão, é possível notar o movimento de diversas áreas para integrar soluções *web* às suas aplicações. Com isso, diversos autores propuseram a utilização de sistemas de visualização baseados no *WebGL*, bem como formas de contornar problemas de atraso na comunicação com formatos de arquivos leves e compatíveis com os protocolos de rede.

Porém, apesar do exposto anteriormente, ainda há carência de pesquisas relacionadas a colaboração em tempo real na qual usuários distantes geograficamente possam trabalhar simultaneamente em um mesmo modelo. Como contribuição para a linha de pesquisa, neste trabalho é proposta uma estratégia para

permitir a colaboração em tempo real entre usuários de forma remota através de navegadores *web*.

4
Tecnologias utilizadas

Neste capítulo são discutidas as principais tecnologias utilizadas para desenvolvimento da aplicação final. Está dividido da seguinte forma: na Seção 4.1 é discutida a biblioteca *React* utilizada no desenvolvimento da interface de usuário, na Seção 4.3 são discutidas as bibliotecas *socket.io* e *flask-socket.io* que implementam o protocolo *WebSocket*.

4.1.
Biblioteca *React*

Criada por um engenheiro de *software* do Facebook, *React* é uma biblioteca para desenvolvimento de interfaces de usuário utilizando a linguagem *JavaScript*. É atualmente utilizada amplamente pela comunidade de desenvolvimento de *software* bem como por diversas empresas, como por exemplo o próprio Facebook e Netflix (Banks & Porcello, 2020).

A biblioteca utiliza uma sintaxe semelhante à do HTML e é dividida em duas bibliotecas principais: *React* e *React*DOM. A biblioteca *React* é responsável pela criação dos componentes visuais enquanto a biblioteca *ReactDOM* é responsável pela renderização da interface da aplicação no *browser* (Banks & Porcello, 2020).

4.1.1.
Componentes

Os componentes são códigos modularizados que possuem e gerenciam seu próprio estado, e são dispostos em hierarquia de árvore. Possuem um método principal chamado *render* o qual retorna um elemento *React*, que é uma descrição

do que irá ser renderizado[2] na tela. Nesse método é utilizada a sintaxe JSX, na qual é possível combinar código *JavaScript* e código em HTML para a escrita dos elementos *React* que irão compor o componente.

Conforme descrito anteriormente, os componentes gerenciam o próprio estado, sendo possível compartilhar o estado de um componente *pai* com seus componentes *filhos*. Não é possível, porém, sem o auxílio de bibliotecas externas de gerenciamento de estado, compartilhar o estado de componentes que não possuam algum parentesco, bem como de um componente *filho* para um componente *pai*. O descrito anteriormente é ilustrado na Figura 4.1.

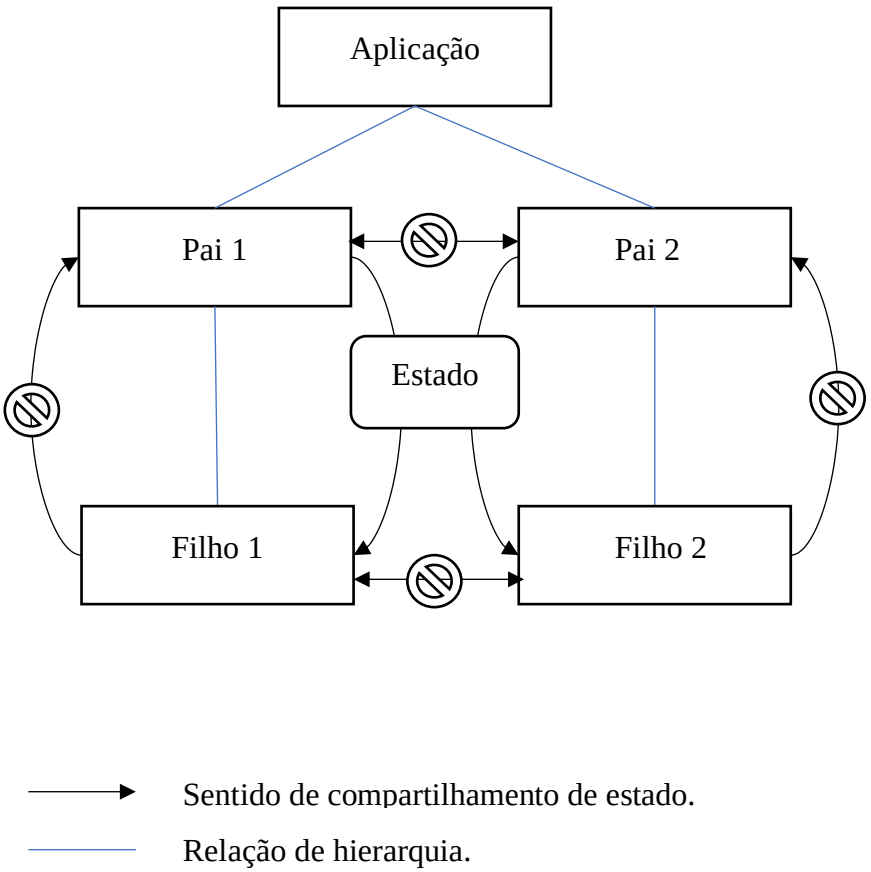


Figura 4.1 – Gerenciamento de estado na árvore de componentes.

À medida que o estado e as propriedades de um componente são modificados, os novos dados são repassados através da árvore hierárquica, que ocasiona a

[2] O verbo renderizar é um anglicismo comumente utilizado na área de ciência da computação e significa desenhar na tela do computador.

renderização dos componentes que compartilham o estado e propriedades, baseando-se nos novos dados (Banks & Porcello, 2020). Ou seja, conforme o componente sofre uma alteração em seu estado, o método *render* do componente é chamado ocasionando a sua renderização na tela.

Como forma de ilustrar o funcionamento dos componentes *React*, o código mostrado no Algoritmo 1 representa um componente de botão, que ao ser selecionado atualiza o seu estado mostrando quantas vezes foi clicado pelo usuário.

Os componentes são criados por meio de classes que herdam da classe *React.Component* através da palavra-chave *extends*. No método construtor é realizada uma chamada à função *super* que inicializa o construtor da classe herdada, e é definido o estado inicial do componente através do atributo *state* que é definido na sintaxe JSON. O método *atualizarContador* é criado nesse exemplo para atualizar o estado do componente, para isso deve-se passar o novo estado do componente como parâmetro do método *setState* que é herdado da classe *React.Component*. Após essa chamada ao método *setState*, o método definido como *render* é chamado automaticamente para renderizar o componente com o estado atualizado na tela.

Algoritmo 1 – Exemplo de componente *React*.

```
class Button extends React.Component {
      constructor() {
            super();
            this.state = {
                  contador: 0
            };
      }

      atualizarContador() {
            const novoEstado = {
                  contador: this.state.contador + 1
            }
            this.setState(novoEstado);
      }

      render() {
            return (<button
                        onClick={() => this.atualizarContador()}>
                              Clicked {this.state.contador} times
                        </button>);
      }
}
```

A Figura 4.2 ilustra o fluxo de estado do componente descrito pelo Algoritmo 1.

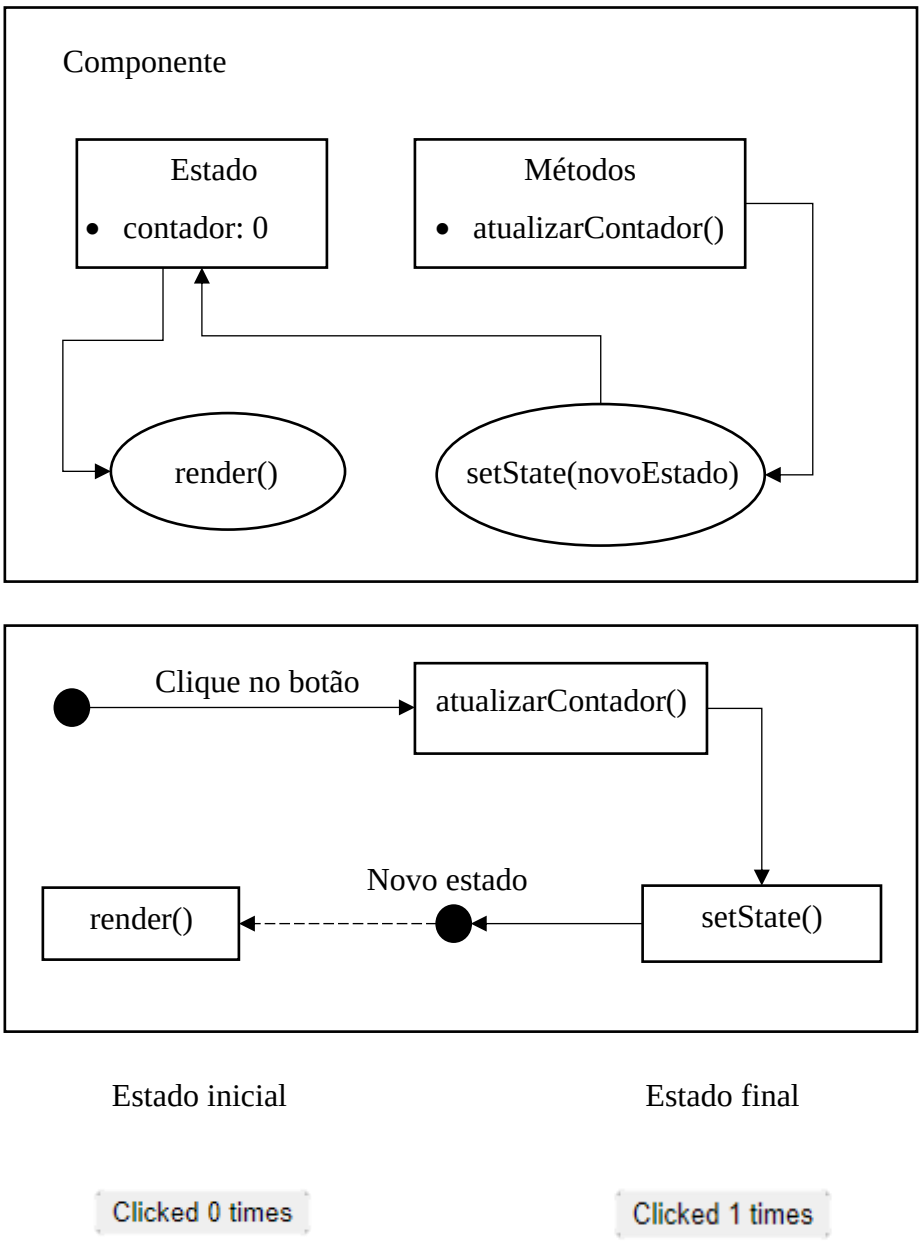


Figura 4.2 – Fluxo de estado de um componente *React*.

Com esse sistema, é possível desenvolver interfaces de usuário de forma modularizada e possibilitando a reutilização de componentes em diferentes locais da interface gráfica.

Para facilitar o processo de gerenciamento de estado dos componentes podem ser utilizadas bibliotecas de terceiros que retiram do componente essa função simplificando o compartilhamento de estado entre componentes, de forma que não seja necessária a passagem de estados pela árvore hierárquica. Essa abordagem também evita a renderização desnecessária de alguns componentes evitando problemas de performance. Devido a simplicidade da interface de usuário abordada nesse trabalho, não é utilizada nenhuma biblioteca de gerenciamento de estado.

4.2. *WebGL*

WebGL é uma tecnologia que possibilita a criação de cenas gráficas 3D através de *browsers.* Tradicionalmente, para a criação de cenas gráficas 3D era necessária a criação de aplicações *stand-alone* utilizando linguagens como C ou C++ e bibliotecas gráficas como o OpenGL ou Direct3D. Com o *WebGL* torna-se possível a criação de aplicações gráficas iterativas através de *browsers* utilizando as linguagens HTML e *JavaScript*. (Matsuda & Lea, 2013).

O *WebGL* é derivado do OpenGL, mais especificamente da versão do OpenGL para dispositivos embarcados, conhecida por OpenGL ES (Angel & Shreiner, 2014; Matsuda & Lea, 2013). A versão mais recente do *WebGL* é a versão 2.0, que de acordo com a documentação expõe a API do OpenGL ES 3.0.

As empresas desenvolvedoras dos *browsers* mais modernos são membros do grupo de trabalho do *WebGL*. Entre eles, Apple (Safari), Google (Chrome), Microsoft (Edge), e Mozilla (Firefox). Isso possibilita a utilização do *WebGL* diretamente através do *browser* sem a necessidade de instalação de plugins ou bibliotecas de terceiros.

4.2.1. *Shaders*

O *WebGL* depende de um eficaz mecanismo de desenho de objetos 2D e 3D chamado de shader (Matsuda & Lea, 2013). Shader são pequenos programas

escritos em linguagem GLSL (OpenGL Shader Language) semelhante à linguagem C responsável por desenhar os objetos em tela.

São necessários dois tipos de *Shaders*: *vertex shader* e *fragment shader*. O *vertex shader* é um programa que descreve características como posição e cor de um vértice. Já o *fragment shader* lida com processamentos como iluminação. A Figura 4.3 mostra o fluxo de um programa em *JavaScript* com o sistema *WebGL*.

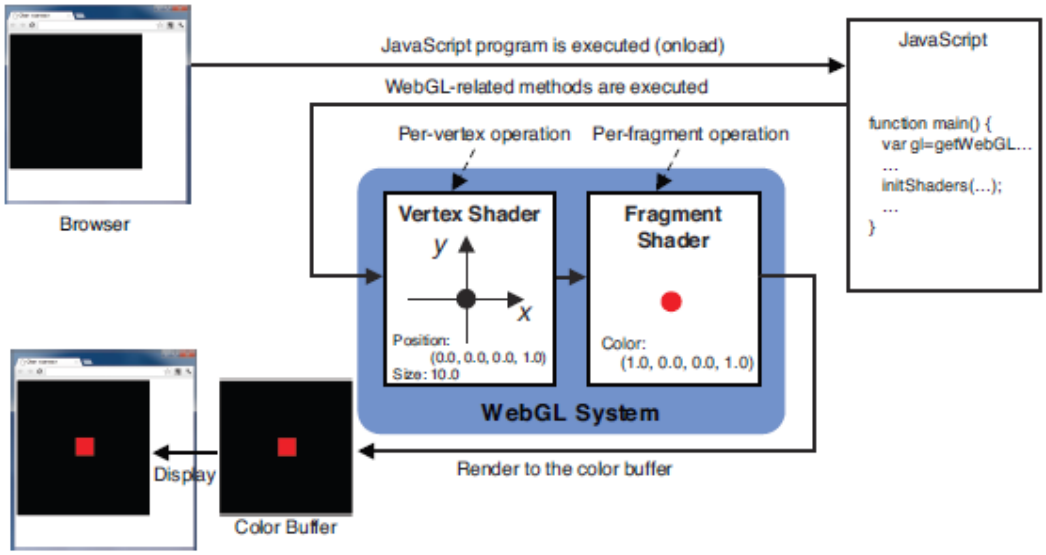


Figura 4.3 – Fluxo de processamento de um programa *JavaScript* e *WebGL* Matsuda & Lea, 2013).

Usualmente, como o sistema 2D pode ser tomado como um caso particular do sistema 3D (Angel & Shreiner, 2014), para aplicações de sistemas gráficos em duas dimensões é utilizada uma projeção ortográfica de pontos que se situam em um mesmo plano.

Os dados como vetores de posição de vértices, matrizes de projeção e outros, são passados do código *JavaScript* para os *Shaders* através de *attributes* ou *uniforms*. A posição das variáveis declaradas no código GLSL são encontradas no código *JavaScript* através das funções `gl.getAttribLocation` e `gl.getUniformLocation`, para que possam ser posteriormente repassados o valores correspondentes. Os trechos de código mostrados no Algoritmo 2 e no Algoritmo 3 mostram um exemplo de um conjunto *vertex shader* e *fragment shader*, respectivamente, para a utilização em geração de gráficos em 2D. Esses mesmos códigos são utilizados na aplicação desenvolvida neste trabalho.

No *vertex shader* mostrado no Algoritmo 2, inicialmente são definidos as variáveis do *shader*. O atributo *a_position* é o vetor da posição dos vértices que são repassados do código *JavaScript* para o shader por meio de um *buffer* de vértices de três dimensões. Esse vetor é aumentado com uma coordenada homogênea, e armazenado na variável *u_position*, para a correta multiplicação pela matriz de projeção (*u_projection*). A variável *gl_Position* é uma palavra-chave da linguagem *GLSL* que trata do valor de saída das posições para a renderização da imagem na tela.

A variável *u_color* é utilizada para repassar as cores das entidades geométricas ao *shader*. Ela é definida no *vertex shader* porém é utilizada apenas no *fragment shader* (Algoritmo 3), que retorna as cores através da palavra-chave *gl_FragColor*.

Algoritmo 2 – Exemplo de código GLSL para Vertex shader.

```
precision mediump float;
attribute vec3 a_position;
vec4 u_position;
uniform vec4 u_color;
uniform mat4 u_projection;
uniform float u_pointSize;

void main() {
      u_position = vec4(a_position.xyz, 1.0);
      gl_Position = u_projection*u_position;
      gl_PointSize = u_pointSize;
}
```

Algoritmo 3 – Exemplo de código GLSL para Fragment shader.

```
precision mediump float;
uniform vec4 u_color;

void main() {
      gl_FragColor = u_color;
}
```

4.3. Bibliotecas *Socket.io* e *Flask-SocketIO*

Conforme discutido na Seção 2.2.2 o protocolo *WebSocket* é um importante protocolo utilizado para o desenvolvimento de aplicações *Web* em tempo real. Para

potencializar o desenvolvimento dessas aplicações é interessante o uso de bibliotecas robustas que forneçam uma API de fácil utilização para o *WebSocket*. Com isso, o *Socket.io* é uma biblioteca que possibilita uma comunicação bidirecional em tempo real e baseada em eventos entre um *browser* e um servidor utilizando por padrão o protocolo *WebSocket*.

Como visto no Capítulo 2, nas aplicações *Web* são necessários dois processos que se comunicam entre si: os processos cliente e servidor. Em ambos os processos é necessário a codificação com a biblioteca *Socket.io*, que disponibiliza uma parte para a aplicação cliente e outra parte para o servidor. Os trechos de código do Algoritmo 4 e do Algoritmo 5 foram retirados e adaptados a partir da documentação da biblioteca *Socket.io* para introduzir o leitor acerca da utilização da biblioteca.

O Algoritmo 4 demonstra como é tratada a utilização da biblioteca para a aplicação cliente. Inicialmente a conexão é iniciada por meio do construtor da classe *io* que retorna um objeto no qual é possível definir eventos e as funções de *call-back* através do método *on*. No método *on* o primeiro parâmetro é o nome do evento e o segundo é a função de *call-back* que será disparada quando esse evento for acionado. Os eventos *connect* e *disconnect* são padrões e serão chamados sempre que houver uma nova conexão ou uma desconexão. Outros eventos podem ser criados conforme necessidade da aplicação.

Para acionar eventos é utilizado o método *emit* da classe *io* que pode ser acionado tanto na aplicação cliente quanto no servidor.

Algoritmo 4 – Exemplo de código *JavaScript* para o cliente utilizando *Socket.io*.

```
import {io} from 'socket.io-client';

const socket = io("ws://localhost:3000");

socket.on("connect", () => {
      socket.emit("salutations", "Client connected");
});

socket.on("message", data => {
      console.log(data);
});

socket.on("greetings", (elem1) => {
      console.log(elem1);
});
```

Já do lado do servidor (Algoritmo 5), para a inicialização é necessário passar como parâmetro para o construtor da classe *io* apenas a porta na qual a aplicação irá funcionar. O restante do código é semelhante ao código da aplicação cliente, entretanto os eventos são definidos na função de *call-back* do evento *connection* que por padrão recebe o parâmetro *socket* que define os atributos do cliente que realizou a conexão. Dessa forma, é possível identificar qual cliente está recebendo os eventos através do *WebSocket*.

Algoritmo 5 – Exemplo de código *JavaScript* para o servidor utilizado *Socket.io*.

```
const io = require("socket.io")(3000);

io.on("connection", socket => {

      socket.emit("greetings", "Hello user");

      socket.on("message", (data) => {
            console.log(data);
      });

      socket.on("salutations", (data) => {
            console.log(data);
      });
});
```

A biblioteca *Flask-SocketIO* é a implementação da biblioteca *Socket.io* para construção de servidores utilizando a linguagem Python e o *framework Flask*. O Algoritmo 5 representado anteriormente para o servidor escrito em *JavaScript* pode ser reescrito para Python conforme ilustrado abaixo no Algoritmo 6.

A maior diferença entre o Algoritmo 5 e o Algoritmo 6, além das sintaxes das linguagens, trata-se de que o objeto criado pela classe *Flask* é passado como parâmetro para o construtor da biblioteca *Socket.io,* a qual trabalha como uma camada para que o *Flask* possa ter as funcionalidades do *WebSocket*.

Como alternativa às funções de *call-back* no *JavaScript* em *Python* são utilizados *decorators*, que funcionam de forma similar às funções de *call-back*. Como exemplo, no Algoritmo 6, no método *on* do objeto *socketio* são utilizados *decorators*, que podem ser identificados pela utilização do caractere "@" antes do método, e logo em seguida é definida a função que irá ser utilizada pelo evento.

Algoritmo 6 – Exemplo de código Python para o servidor utilizando *Flask*-SocketIO.

```
from flask import Flask
from flask_socketio import SocketIO, emit

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins = "*")

@socketio.on("connect") #decorator
def handle_connection():

   emit("greetings", "Hello user")

@socketio.on("message") #decorator
def handle_connection(data):

   print(data)

@socketio.on("salutations") #decorator
def handle_connection(data):

   print(data)

if __name__ == '__main__':
socketio.run(app, port=8000, host='0.0.0.0')
```

5
Aplicação para prova de conceito

5.1.
Arquitetura da aplicação

Pelo exposto nos Capítulos 2 e 4 e devido às necessidades da aplicação de possuir colaboração em tempo real e múltiplas requisições serem feitas em intervalos curtos, preferiu-se a utilização do *WebSocket* como protocolo da camada de aplicação. Para isso, são utilizadas as bibliotecas *Socket.io* e *Flask-SocketIO* que implementam o protocolo de *WebSocket* para utilização nas linguagens *JavaScript* e Python, respectivamente.

A aplicação consiste em duas aplicações principais, que são chamadas neste trabalho de servidor e cliente. As aplicações realizam uma conexão por meio do *WebSocket* e participam do canal bidirecional para troca de mensagens entre a camada cliente e servidor.

A aplicação que executa no servidor foi desenvolvida utilizando a linguagem Python, e utiliza uma biblioteca (*HeTool*) resultante da pesquisa de um grupo de pesquisa liderado pelos Professores Luiz Fernando Martha e André Maues Brabo e descrita de forma aprofundada na Dissertação de Danilo Silva Bomfim (Bomfim, D. S. & Martha, L. F., 2021).

A biblioteca *HeTool* apresenta uma metodologia de uso genérico, é baseada na estrutura de dados *Half-Edge*, e considera os aspectos topológicos e geométricos, bem como permite a configuração de atributos de simulação pelo usuário sem a necessidade de alteração do código fonte (Bomfim, D. S. & Martha, L. F., 2021).

A aplicação cliente foi desenvolvida neste trabalho utilizando a linguagem *JavaScript* e possui como um dos objetivos principais funcionar no *browser* de diversos dispositivos e sistemas operacionais, inclusive dispositivos móveis.

O servidor possui uma interface de programação de aplicação (API) desenvolvida utilizando a biblioteca *Flask-SocketIO*, que recebe as requisições através do protocolo *WebSocket* e processa de acordo com os métodos disponíveis para manipular as entidades geométricas e topológicas na estrutura de dados.

As duas aplicações, cliente (*JavaScript*) e servidor (*Python*), são hospedadas na plataforma Heroku, que se trata de um modelo de serviço do tipo *Platform as a Service* (PaaS) escolhido devido a simplicidade fornecida pela plataforma em realizar a hospedagem de códigos em Python e *JavaScript*. A biblioteca *HeTool* possui diversas aplicações, de forma que uma camada visual mais específica a um determinado problema pode ser desenvolvida e pode-se utilizar a API que o servidor fornece. Levando em consideração esse fator, as aplicações encontram-se armazenadas em domínios diferentes na plataforma Heroku.

Na Figura 5.1 é possível observar a arquitetura da aplicação discutida nesse trabalho. Inicialmente o usuário insere o domínio da aplicação cliente solicitando o código *JavaScript* que será interpretado pelo *browser*. Após o recebimento desse arquivo o *browser* iniciará a interpretá-lo e logo em seguida solicitará a conexão com a aplicação servidora. Após esse processo, a aplicação cliente possuirá uma conexão com a aplicação servidora através do *WebSocket*.

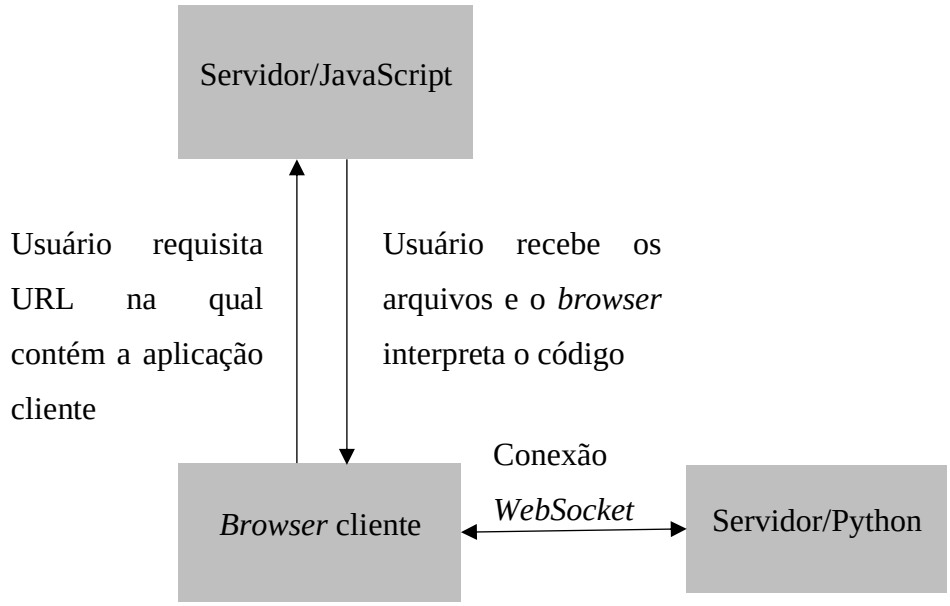


Figura 5.1 – Arquitetura da aplicação.

A Figura 5.2 mostra em detalhe os componentes principais da aplicação desenvolvida.

A aplicação cliente possui um *Canvas* para renderizar as informações geométricas na tela utilizando o *WebGL*, e receber as entradas do usuário. As entradas do usuário são processadas e quando finalizadas são enviadas ao componente *API*, que permite a comunicação através do *WebSocket*. A aplicação cliente armazena um modelo simplificado contendo as informações geométricas para renderização.

O servidor possui também um componente *API* responsável pela comunicação através do *WebSocket*. As rotas da *API* direcionam as chamadas recebidas para os métodos disponíveis na biblioteca *Hetool*, que armazena a estrutura de dados. Por fim, há o componente para geração de malha de elementos finitos bidimensionais, que utiliza a biblioteca *Mesh2d.*

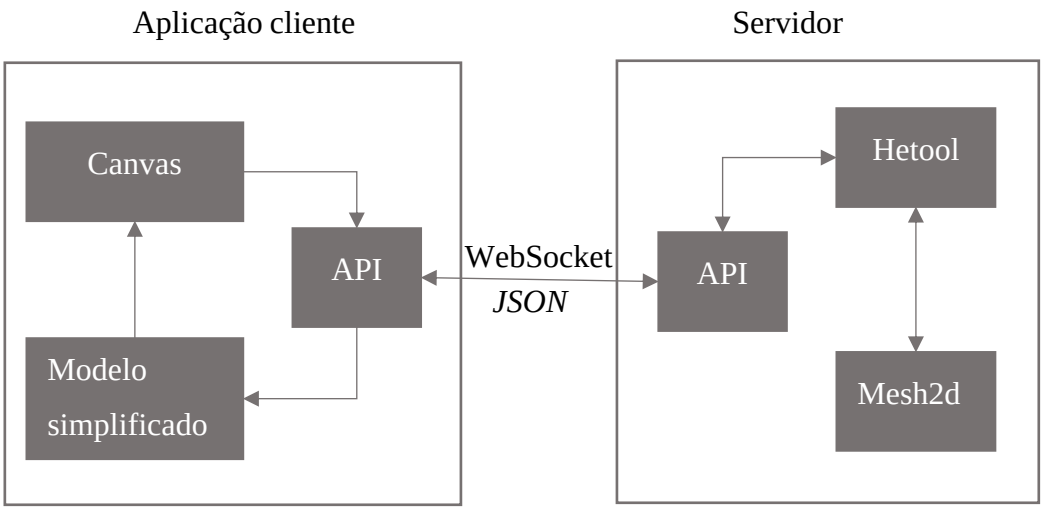


Figura 5.2 – Componentes da arquitetura da aplicação.

5.2. Interface de usuário

A Figura 5.3 apresenta a interface de usuário que consiste basicamente em: um menu lateral de opções na qual o usuário poderá selecionar uma ação (desenhar curvas, ativar/desativar o *grid*, selecionar e deletar curvas) na aba *Modeling*, selecionar as opções de criar ou entrar em uma sala na aba *Collaboration* e também

o gerenciador de atributos na aba *Attributes*; uma faixa de opções de câmera no canto superior direito na qual é possível movimentar a câmera, aumentar e diminuir o zoom e ajustar o desenho à tela, essas ações com exceção da última podem ser realizados com o botão central do mouse; linha de comando; e, opções de *snap* (atração) e subdivisão do *grid* (grade de ponto).

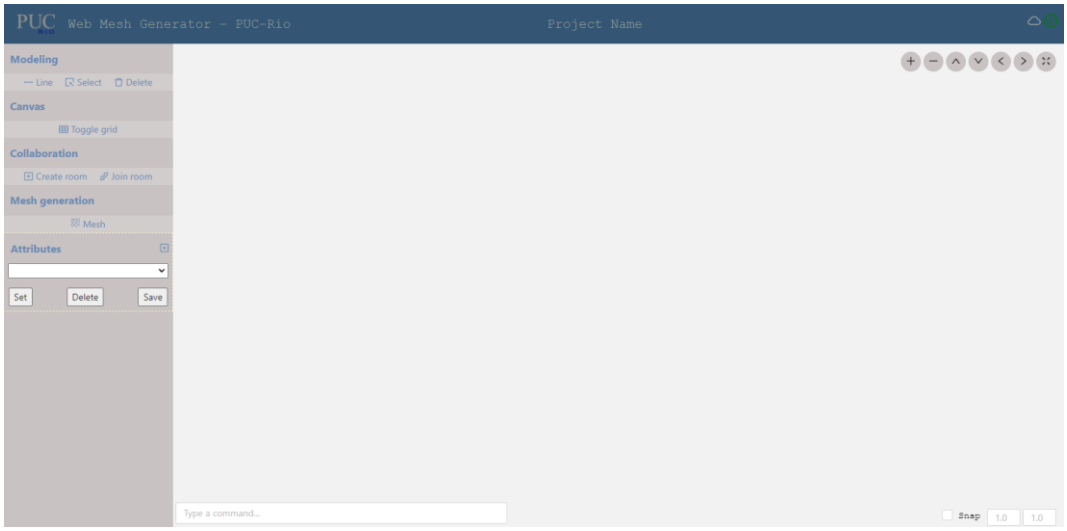
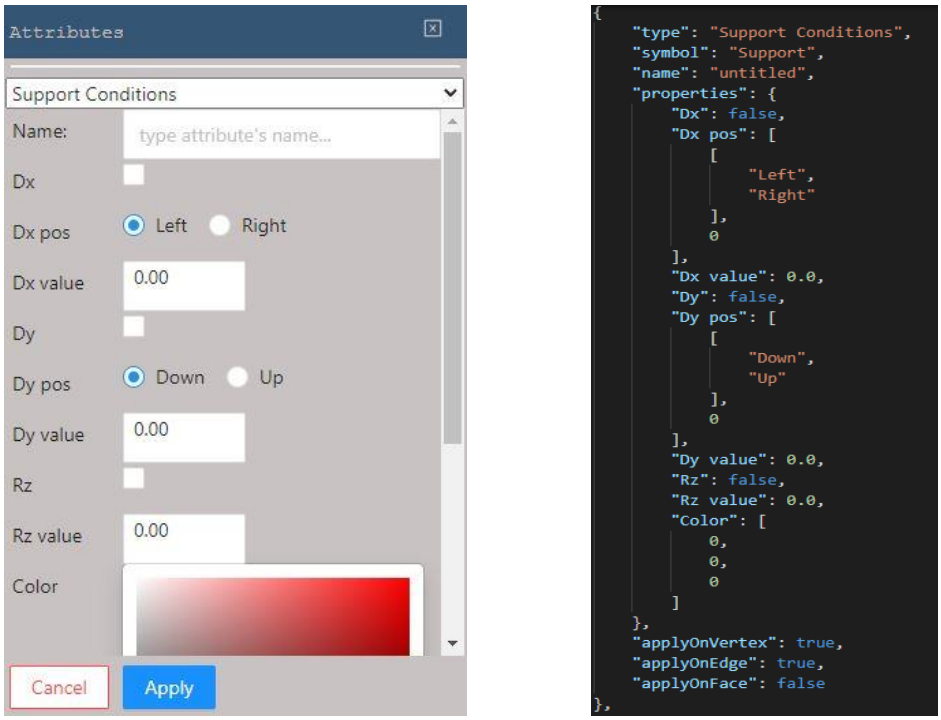


Figura 5.3 – Interface de usuário.

Para a definição de propriedade dos modelos são utilizados atributos. As entidades geométricas possuem atributos conforme definido pelo usuário, os atributos podem assumir forma, como exemplo, de condições de suporte, propriedades de material, carregamentos e entre outras.

O gerenciador de atributos é configurável permitindo ao usuário incluir protótipos de atributos em um arquivo de formato *JSON* contendo o nome do atributo e o tipo do atributo (*string*, *int*, *float*, *color*), e assim o gerenciador de atributos irá formatar a janela conforme os protótipos configurados. A janela para criação de atributos pode ser visualizado na Figura 5.4a, e o correspondente protótipo em formato *JSON* na Figura 5.4b, como exemplo é mostrado o atributo padrão de condições de suporte. Ao criar o atributo é possível selecioná-lo na lista de atributos no menu lateral.



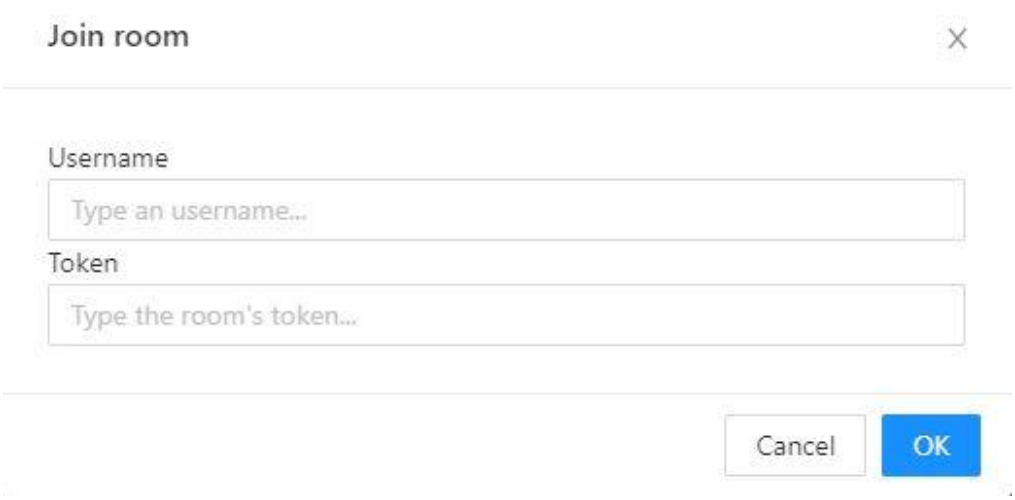
(a) (b)

Figura 5.4 – Componente de Gerenciador de atributos (a). Arquivo *JSON* de configuração de atributos (b).

No canto direito do cabeçalho ainda é possível visualizar a situação da aplicação em relação ao servidor, caso a aplicação esteja conectada um ícone verde será mostrado, caso não esteja conectado será um ícone vermelho. Como a aplicação necessita da estrutura de dados só é possível iniciar a modelagem após a conexão com o servidor, levando em consideração que o servidor é o responsável por guardar essa informação.

As janelas de criação de sala e de entrada em uma sala existente são mostradas na Figura 5.5. Para criar ou entrar em uma sala virtual é necessário fornecer um nome de usuário e para entrar na sala virtual é preciso inserir o *token* da sala criada por outro usuário anteriormente.

(a)



(b)

Figura 5.5 – Janelas de criação de sala (a) e para entrar em sala (b).

Após a criação de uma sala, múltiplos usuários podem realizar a modelagem em colaboração. A interface para usuários conectados à uma sala possui diferenças em relação a apresentada anteriormente, possuindo uma nova janela para realizar comentários em tempo real. A interface fica como mostrada na Figura 5.6.

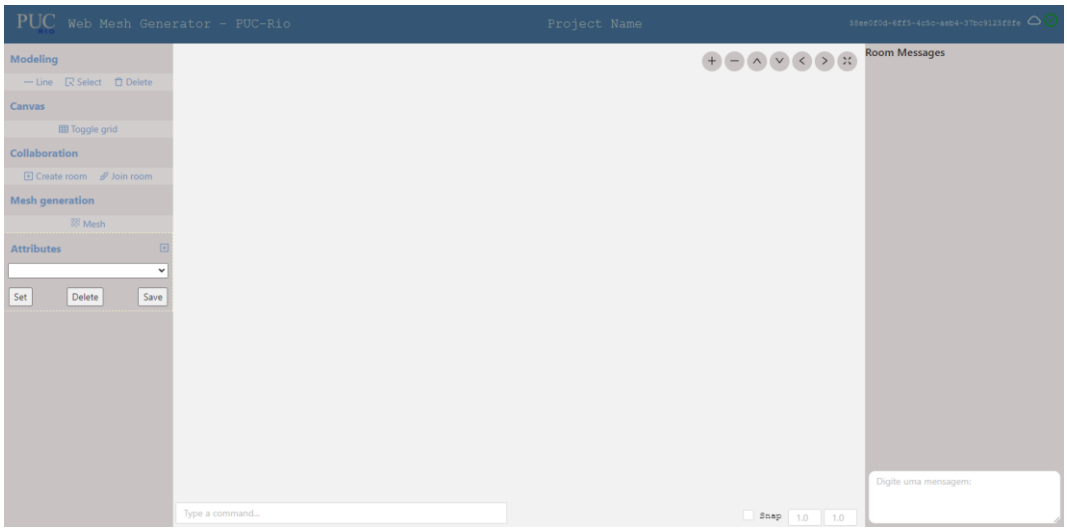


Figura 5.6 – Interface para usuários conectados a uma sala virtual.

5.3. Fluxo de utilização

Nesta seção é discutido o processo para a criação de modelos incluindo a criação da geometria, inserção de atributos (condições de contorno, propriedades dos materiais e subdivisões das entidades geométricas) e geração de malha. Como forma de facilitar o entendimento o processo é apresentado por meio de um exemplo didático seguido dos diagramas de sequência para cada passo.

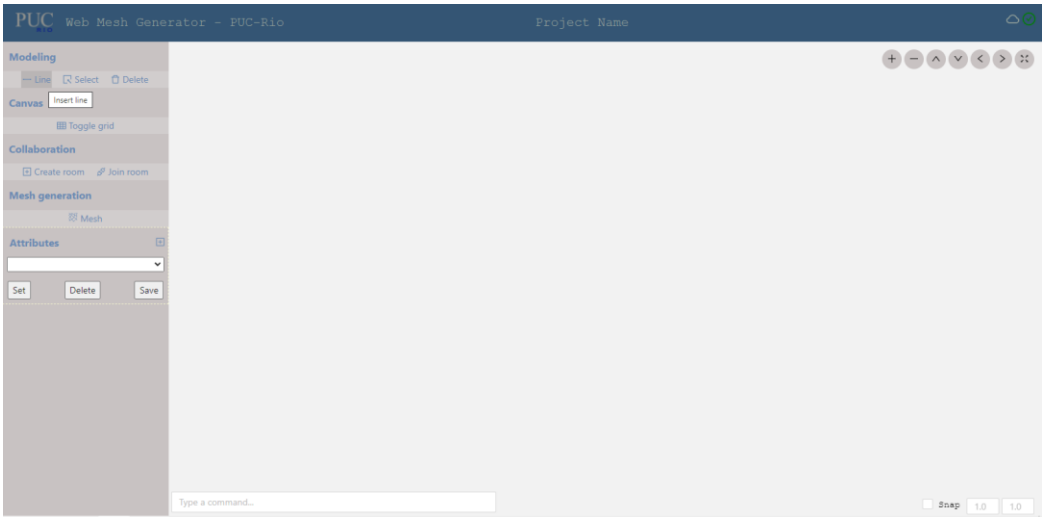
5.3.1. Inicialização

Inicialmente a aplicação estará conforme na Figura 5.3, possuindo o usuário a opção de realizar a modelagem individualmente ou em grupo, caso seja em grupo esse usuário deverá criar uma sala ou juntar-se a uma sala criado por outro usuário.

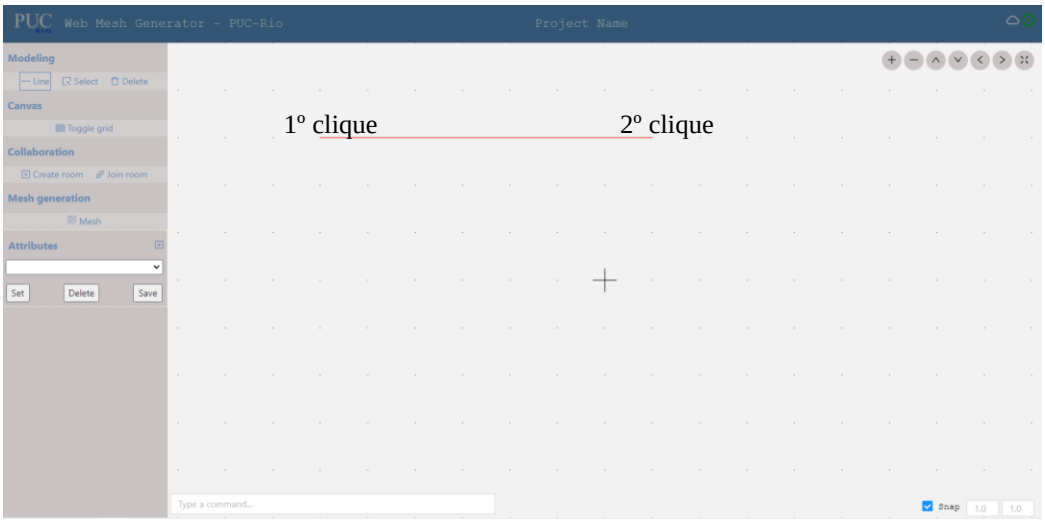
Para criar a sala o usuário deverá clicar em *Create Room* que irá ativar a janela da Figura 5.5 (a) após inserir o nome de usuário e clicar em *OK* a sala é criada. Para entrar em uma sala virtual criada anteriormente o usuário deverá clicar em *Join Room* que irá ativar a janela da Figura 5.5 (b) após inserir o nome de usuário e o token e clicar em *OK* o usuário é conectado à sala virtual. A interface estará conforme mostrada na Figura 5.6.

5.3.2.
Modelagem

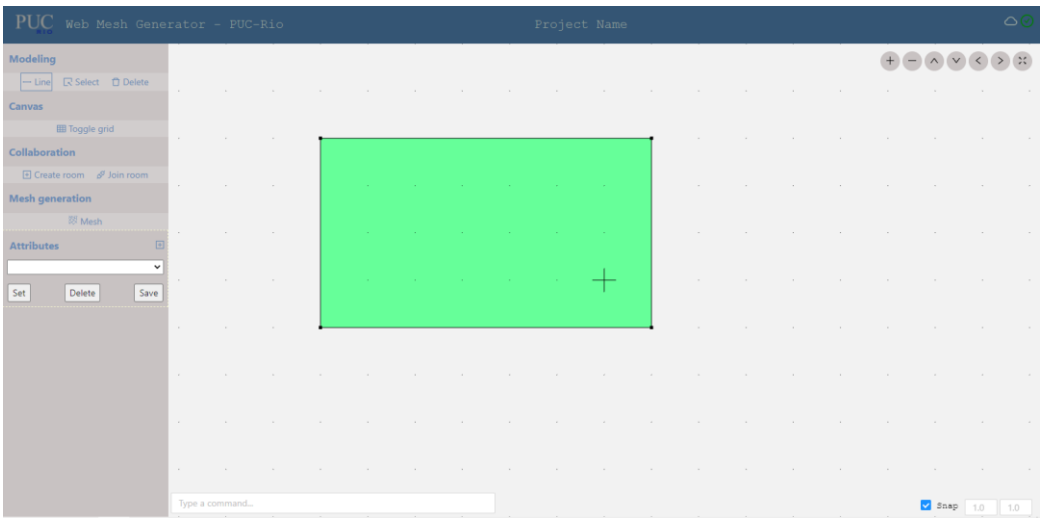
O primeiro passo para a modelagem é a definição da geometria do modelo. Nessa etapa o usuário escolhe a entidade geométrica que irá desenhar, no momento de publicação deste trabalho a aplicação possui apenas a possibilidade de inserção de linhas. A Figura 5.7 mostra o passo a passo de um exemplo de geometria, inicialmente é ativado a coleta de linhas (Figura 5.7a) clicando no botão de inserir linha no menu lateral. Ao clicar no botão de linha o método *changeMouse* é chamado iniciando a coleta de uma curva. A linha é então inserida através de dois cliques em pontos diferentes no Canvas, o método *onMouseUp* é responsável pelo tratamento da finalização da coleta da curva, caso a inserção de pontos para a curva esteja concluída a mesma é inserida no modelo chamando o método *insertCurve* da classe *Api* (Figura 5.7b). O método *insertCurve* é responsável por enviar a mensagem ao servidor para a inserção da curva coletada na estrutura de dados. Para realizar o envio de mensagem é utilizando o método *emit* do objeto *socket*, passando como parâmetros o nome do evento e os parâmetros da curva.



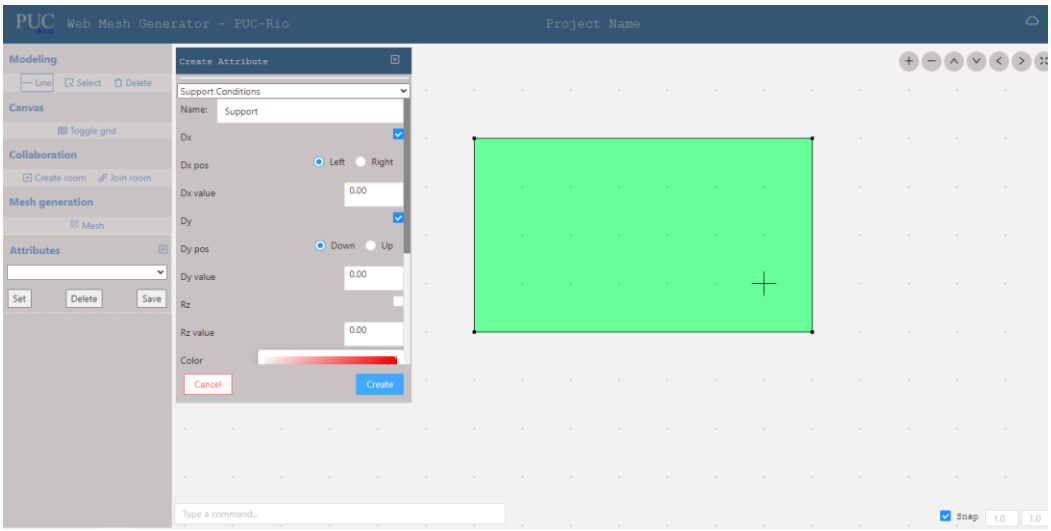
(a)



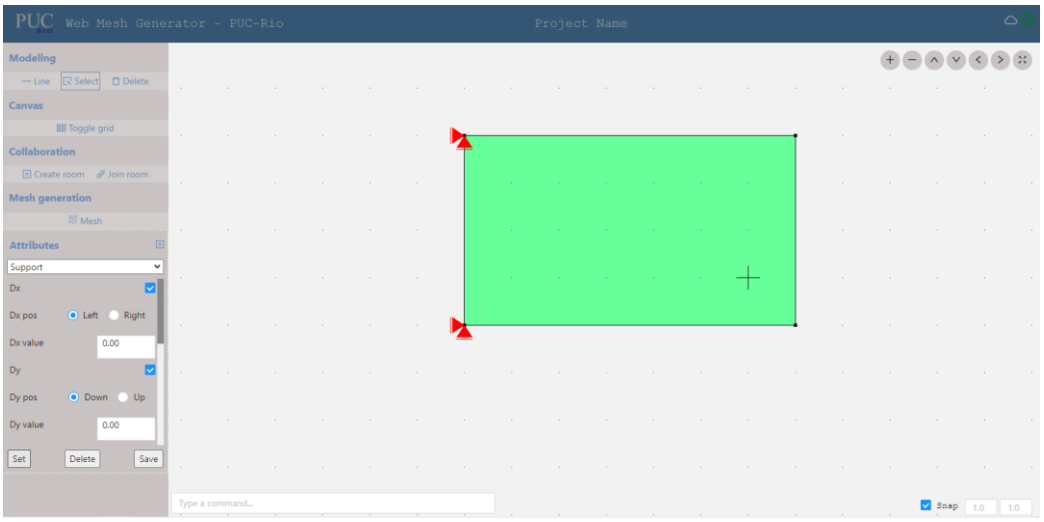
(b)



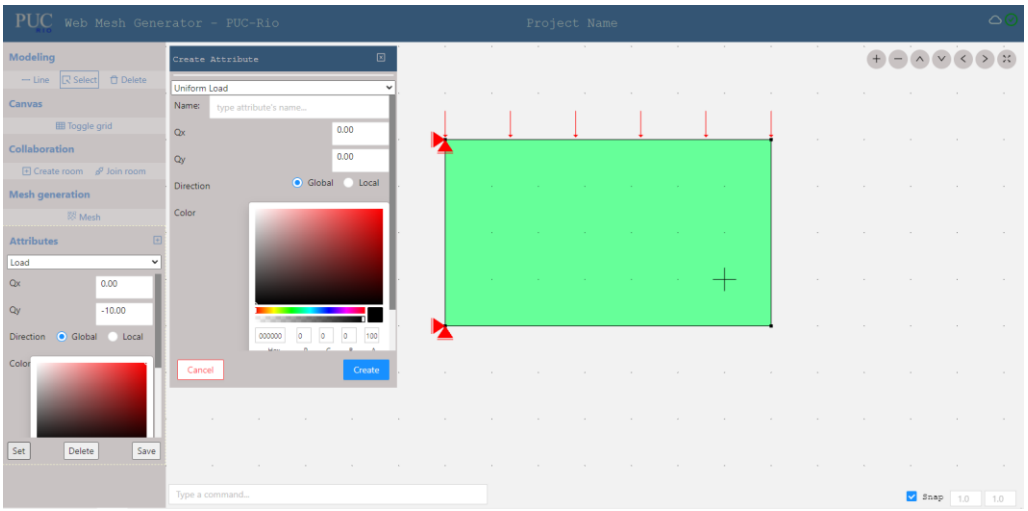
(c)



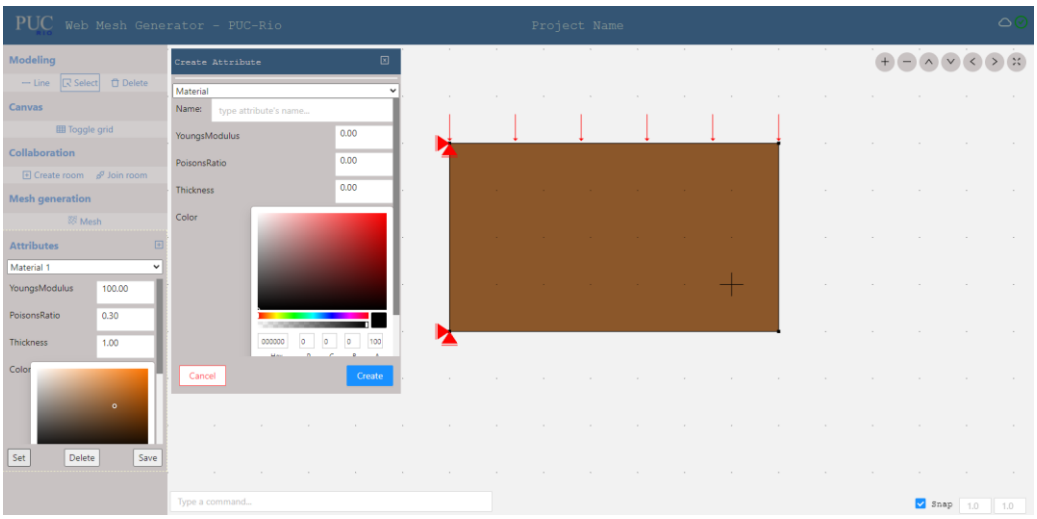
(d)



(e)



(f)



(g)

Figura 5.7 – Passo a passo delimitação geometria, condições de contorno e propriedades do material.

Após a inserção de quatro linhas, repetindo o passo anterior, e, fechando o polígono a região formada pelo interior do polígono é reconhecida automaticamente (Figura 5.7c). O modelo é enviado de volta à aplicação cliente a cada modificação realizada nele. Para isso foi criado o canal *update-model* para enviar a mensagem através do *WebSocket* com as informações do modelo. Ao receber o modelo atualizado, é chamado o método *updateCanvas* que o renderiza em tela. O diagrama de sequência para criação de linhas é mostrado na Figura 5.8.

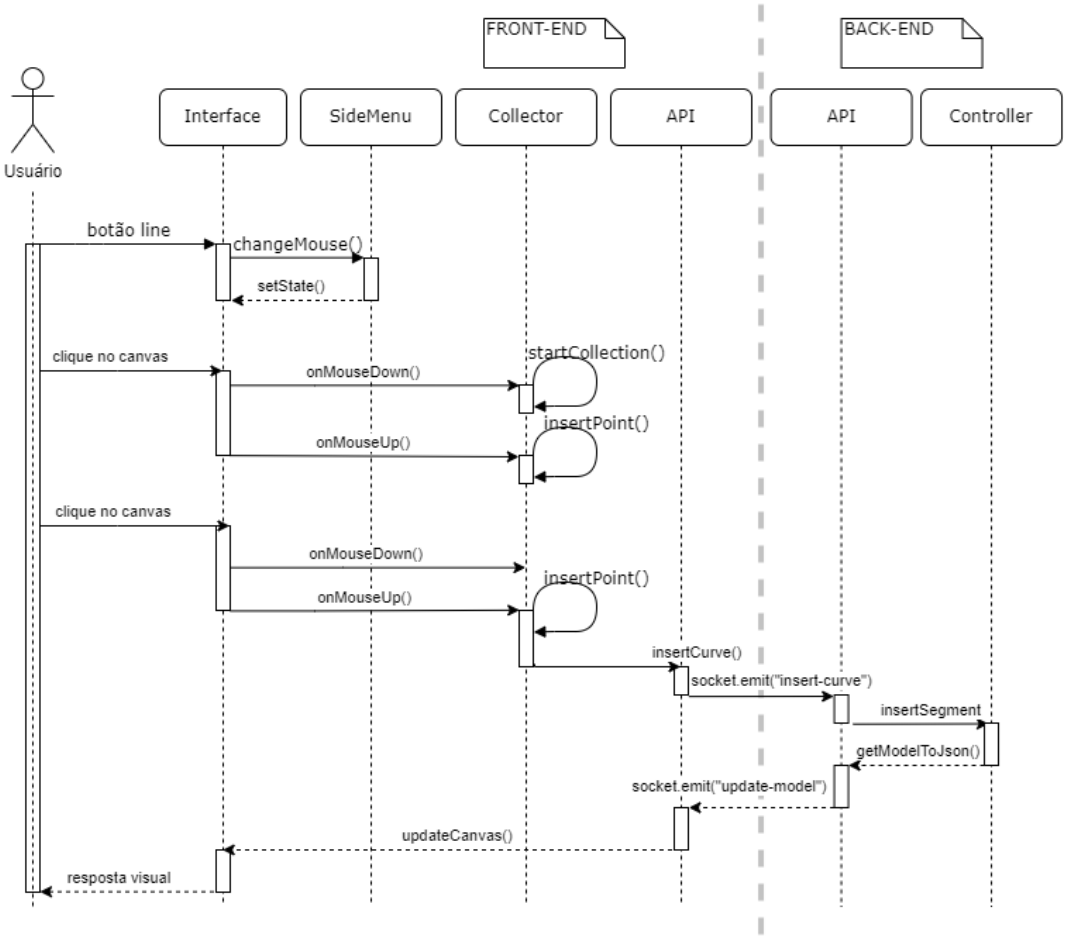


Figura 5.8 – Diagrama de sequência para criação de linhas.

Após definida a geometria são inseridas as condições de contorno, como apoios e carregamentos, bem como as propriedades do material. Essas condições são inseridas através de atributos.

Para isso o usuário deverá ativar a janela para criação de atributos. A janela é ativada ao clicar no botão de criação de atributos no menu lateral. Ao clicar no botão

de criação de atributos o método *setAttributesVisible* é chamado para ativar a janela de criação de atributos.

Após modificar o valor de uma propriedade do atributo o método *handleChangeProperty* atualiza o valor da propriedade para o atributo em rascunho.

Um atributo é definitivamente criado ao clicar no botão *Create* (Figura 5.7d). Ao clicar para criar o atributo, o método *createAttributes* da classe *Api* é chamado. Esse método é responsável por enviar a mensagem de criação de atributos contendo os parâmetros do atributo para o servidor.

Após isso, ao selecionar, no menu lateral, o atributo de condições de suporte criado anteriormente e clicar em *set* é inserido o atributo de condições de apoio na entidade selecionada (Figura 5.7e). O método *setAttributes* é chamado para enviar a mensagem de aplicação de atributos nas entidades que estejam selecionadas. O diagrama de sequência para a criação de atributos é mostrado na Figura 5.9.

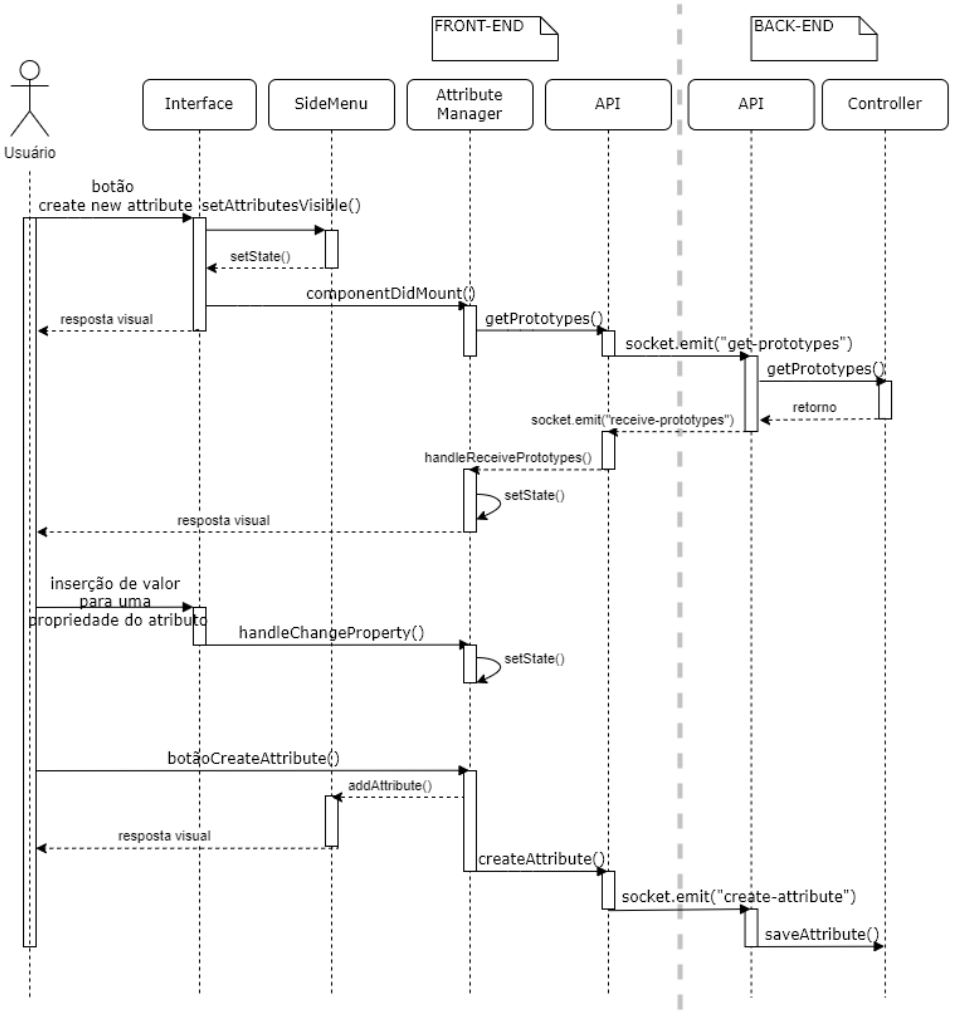


Figura 5.9 – Diagrama de sequência para criação de atributos.

Após a criação do atributo, o usuário poderá selecionar o atributo criado e aplicar o atributo na entidade geométrica. O diagrama de sequência para esse caso de uso é mostrado na Figura 5.10.

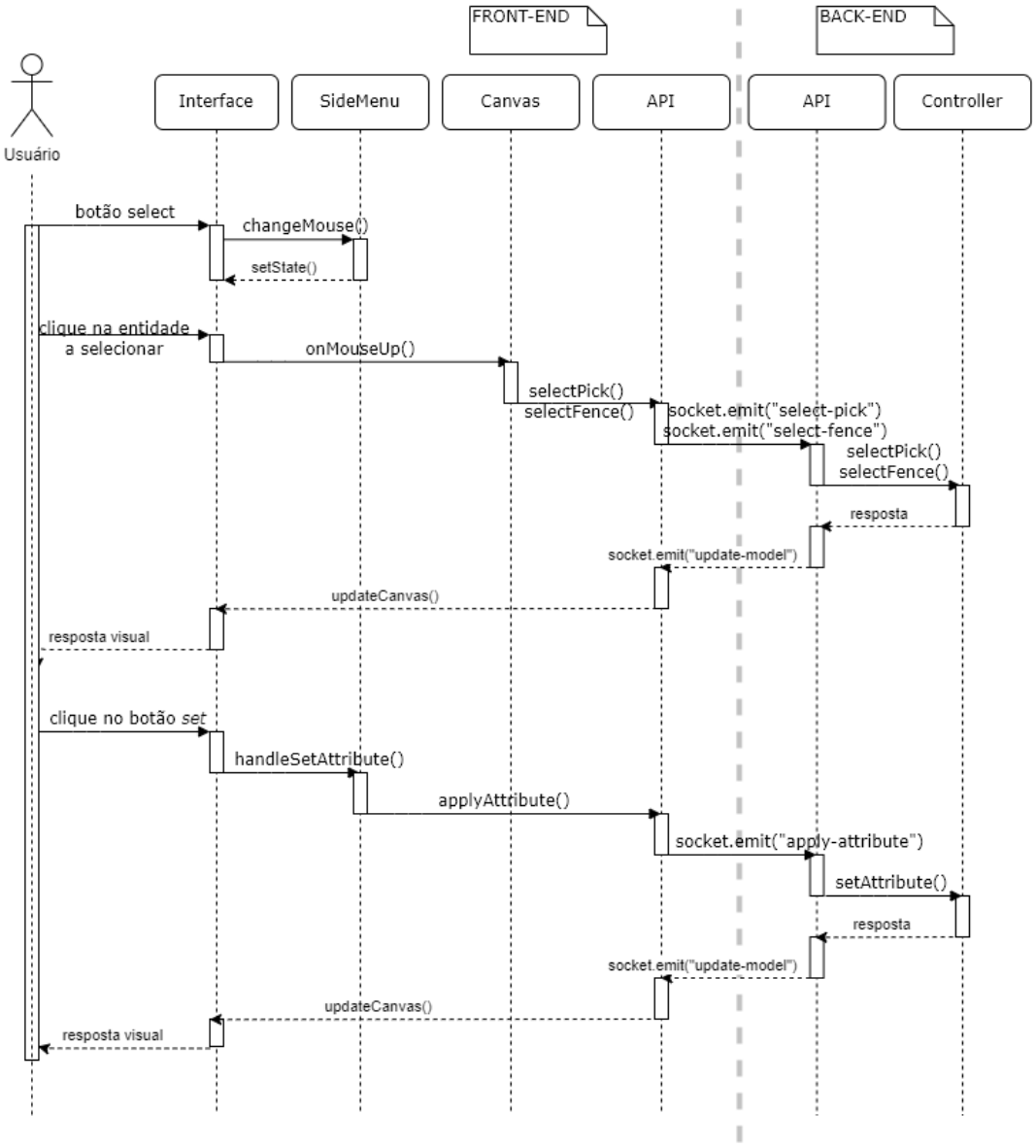
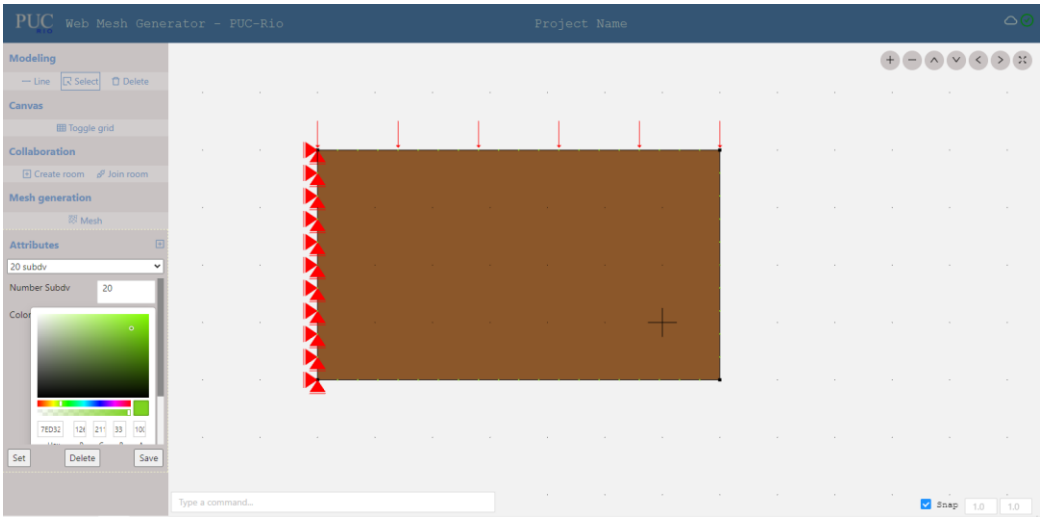


Figura 5.10 – Diagrama de sequência para aplicar atributo.

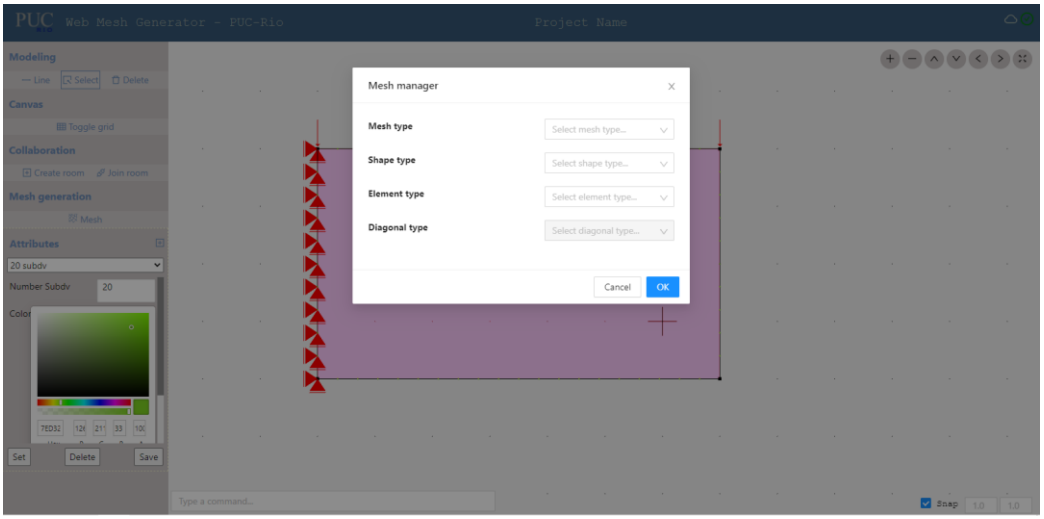
Repetindo o passo anterior, porém agora selecionando o protótipo de carregamento uniforme e selecionando a entidade geométrica a aplicar o carregamento, são inseridas as condições de carregamento (Figura 5.7f). Para inserir as propriedades do material, o usuário deve selecionar no gerenciador de atributos o protótipo de propriedades de material, inserir as propriedades e

selecionar a região delimitada pelas arestas para aplicar as propriedades de material ao modelo (Figura 5.7g).

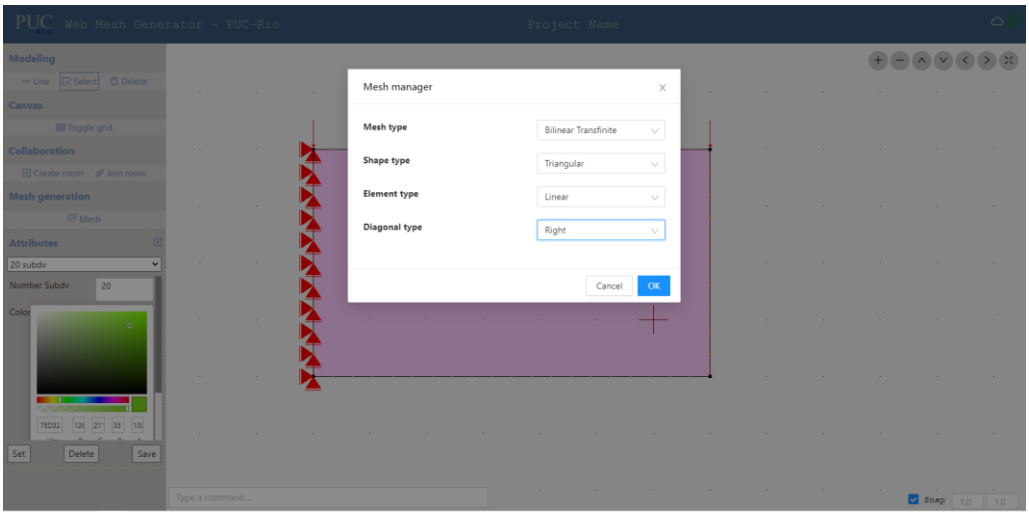
Após a definição da geometria e dos atributos do modelo, o usuário pode seguir para a discretização do problema através da geração de malha. O passo a passo para gerar a malha do modelo é mostrado na Figura 5.11.



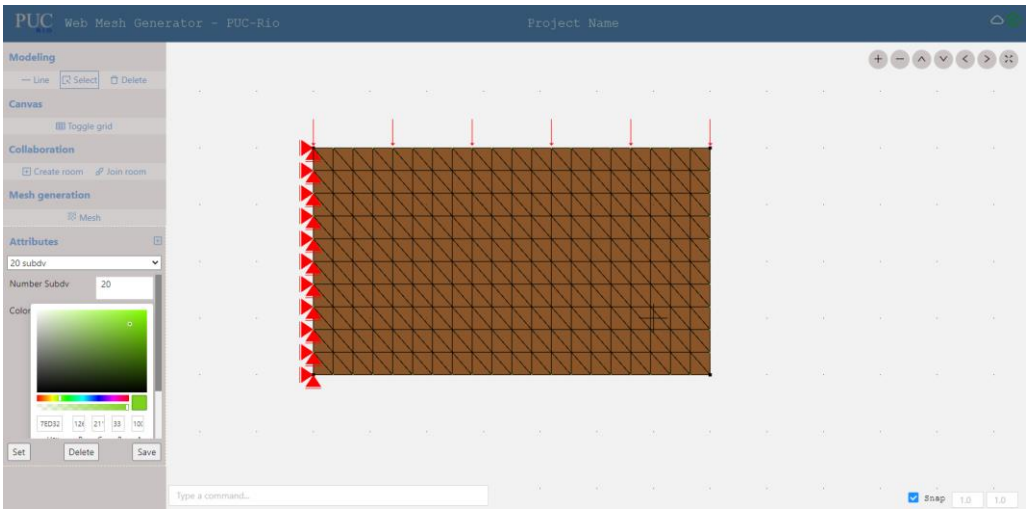
(a)



(b)



(c)



(d)

Figura 5.11 – Passo a passo discretização do modelo.

Inicialmente o usuário deve ainda no gerenciador de atributos realizar a discretização das arestas, para isso o usuário deve na aba de número de subdivisões inserir a quantidade de subdivisões desejadas para cada aresta, para o exemplo foram utilizadas vinte subdivisões para as arestas horizontais e dez subdivisões para as arestas verticais (Figura 5.11a). Após, o usuário deve selecionar a região a ser discretizada pela malha, e clicar no botão de malha no menu lateral, a janela de gerenciamento de malha irá abrir conforme mostra a Figura 5.11b. O usuário pode escolher diversos parâmetros para a geração de malha entre eles:

- Tipo de malha:
  - Bilinear transfinito

- Trilinear transfinito
- Quadrilateral template
- Quadrilateral seam
- Triangular boundary contraction

- Formato do elemento:
  - Triangular
  - Quadrilateral
- Tipo do elemento:
  - Linear
  - Quadrático
- Diagonal:
  - Direita
  - Esquerda
  - Union Jack
  - Ótima

Os recursos para geração de malha foram integrados a biblioteca *HETOOL* através da biblioteca *Mesh 2d* (Miranda & Martha, 2000). Essa biblioteca é resultante da pesquisa realizada por um grupo de pesquisa do Instituto Tecgraf/PUC-RIO com orientação do Prof. Luiz Fernando Martha.

Para o exemplo foram escolhidos os parâmetros mostrados na Figura 5.11c. O resultado pode ser visualizado na Figura 5.11d.

O diagrama de sequência da aplicação para a geração de malha de uma região é mostrado na Figura 5.12.

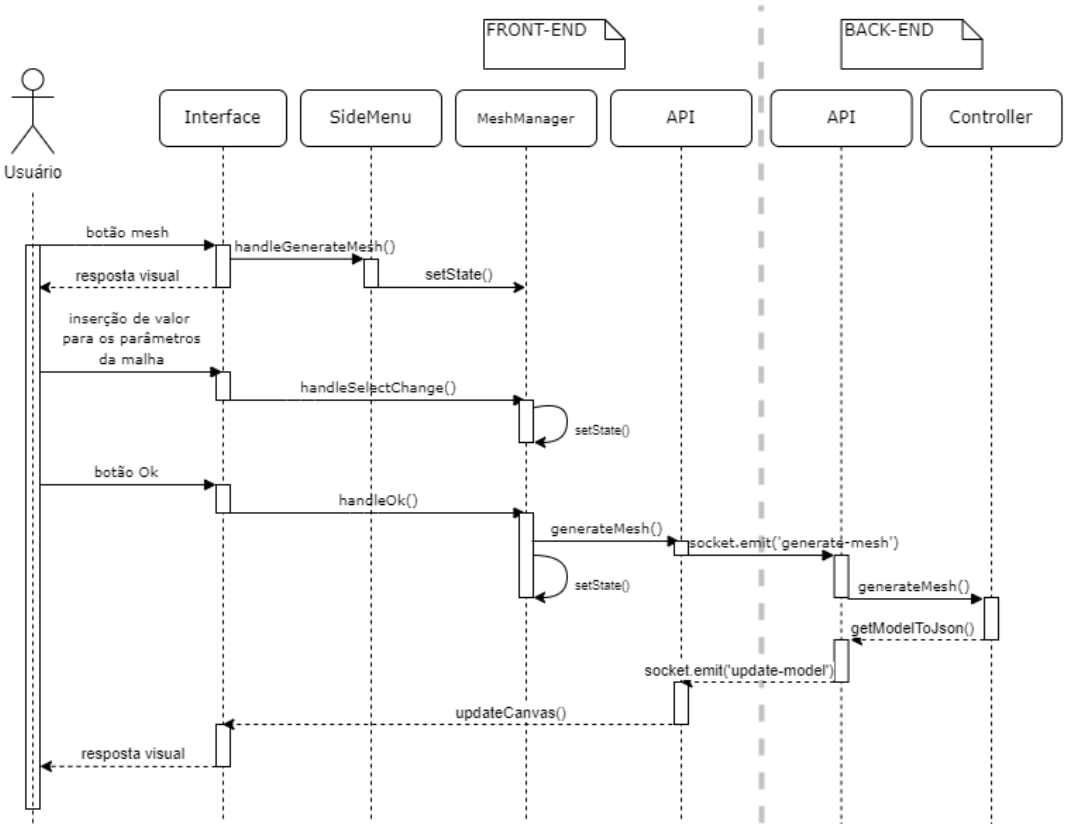


Figura 5.12 – Diagrama de sequência para inserção de malha de uma região.

Ao modificar um dos parâmetros da malha, o estado que armazena as informações é atualizado pelo método *handleSelectChange*. Após selecionar todos os parâmetros necessários para a geração da malha específica, o método *generateMesh* da classe *Api* é responsável por realizar a comunicação com o servidor repassando os dados necessários para geração da malha. Após a malha ser gerada pela biblioteca *HeTool* o modelo atualizado é enviado novamente para a aplicação cliente através do canal *update-model*. Quando o evento *update-model* é acionado na aplicação cliente, o método *updateCanvas* é chamado e renderiza o modelo atualizado com a malha em tela.

6
Resultados

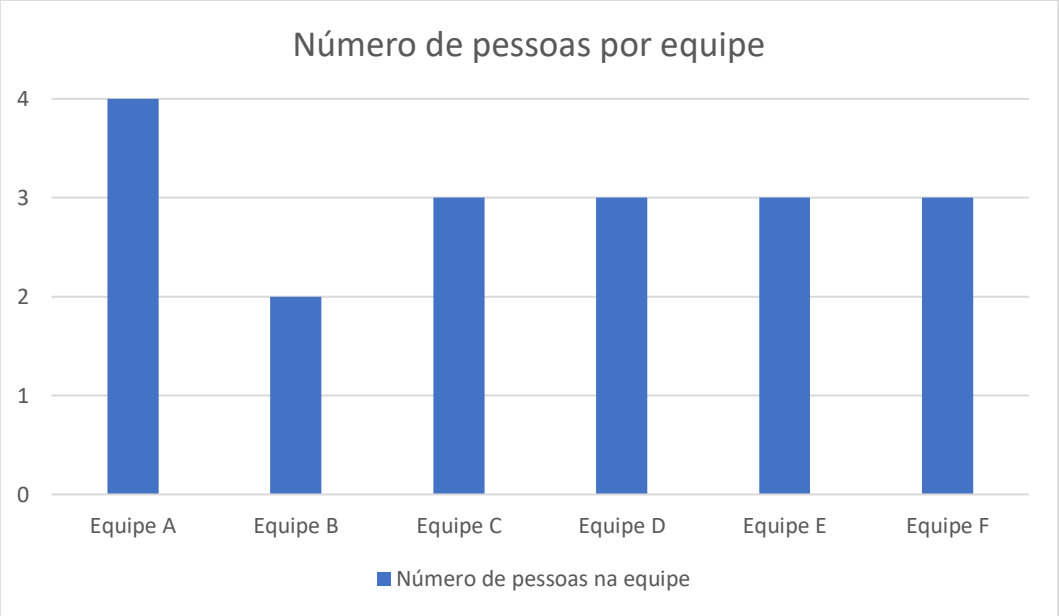
O objetivo desse capítulo é apresentar a metodologia de avaliação da estratégia de modelagem colaborativa de forma remota e pela *Web* através da utilização da aplicação de prova de conceito apresentada no Capítulo 5. É explicada a metodologia de avaliação e posteriormente são apresentados os resultados obtidos através da resposta dos integrantes das equipes participantes por meio da aplicação de um questionário.

6.1.
Metodologia

Como forma de avaliar a proposta de modelagem colaborativa e remota pela *Web* proposta neste trabalho, seis equipes de duas a quatro pessoas foram submetidas à utilização da aplicação para realizar a modelagem geométrica e geração de malhas de modelos de elementos finitos bidimensionais para simulação estática de problemas de elasticidade. As equipes possuíram a flexibilidade de utilizar a aplicação da forma que fosse mais conveniente, possibilitando a divisão das responsabilidades na tarefa de modelagem por parte dos participantes.

O Gráfico 6.1 mostra a divisão de número de integrantes por equipe que realizaram o experimento, ao total foram dezoito participantes. É importante informar que as equipes foram formadas por grupos espalhados geograficamente, não apenas em uma região específica, bem como não foi realizado algum controle sobre a forma que realizaram o experimento (videoconferência, ou apenas utilizando o *chat* da aplicação).

Gráfico 6.1 – Número de pessoas por equipe.



A Figura 6.1 mostra o modelo inicial do exemplo utilizado no experimento com a delimitação geométrica e indicação de atributos (propriedades do material, condições de suporte e carregamento) a serem adotados. E a Figura 6.2 mostra a malha a ser gerada após a conclusão da geometria e inserção de atributos do modelo inicial.

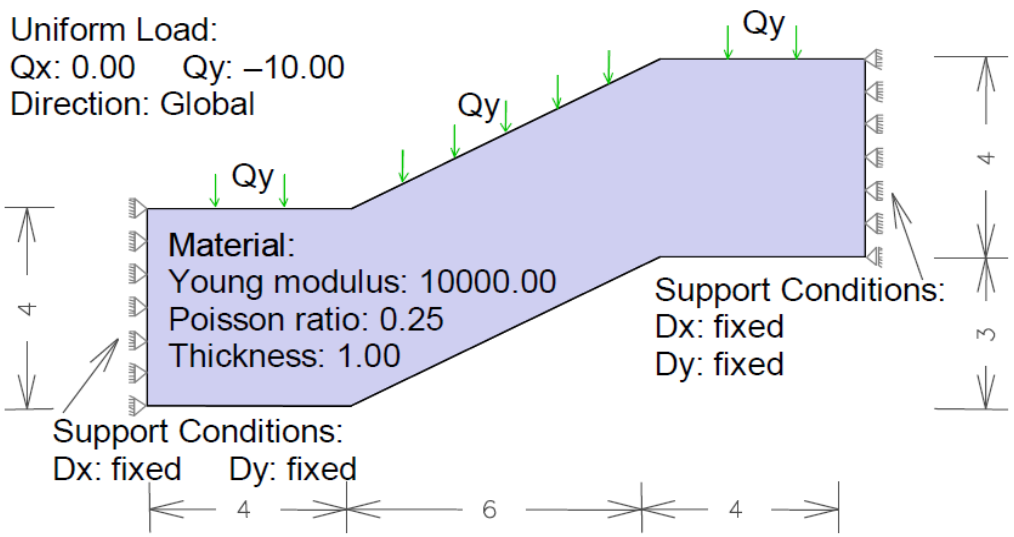


Figura 6.1 – Geometria e indicação de atributos do modelo inicial utilizado no experimento.

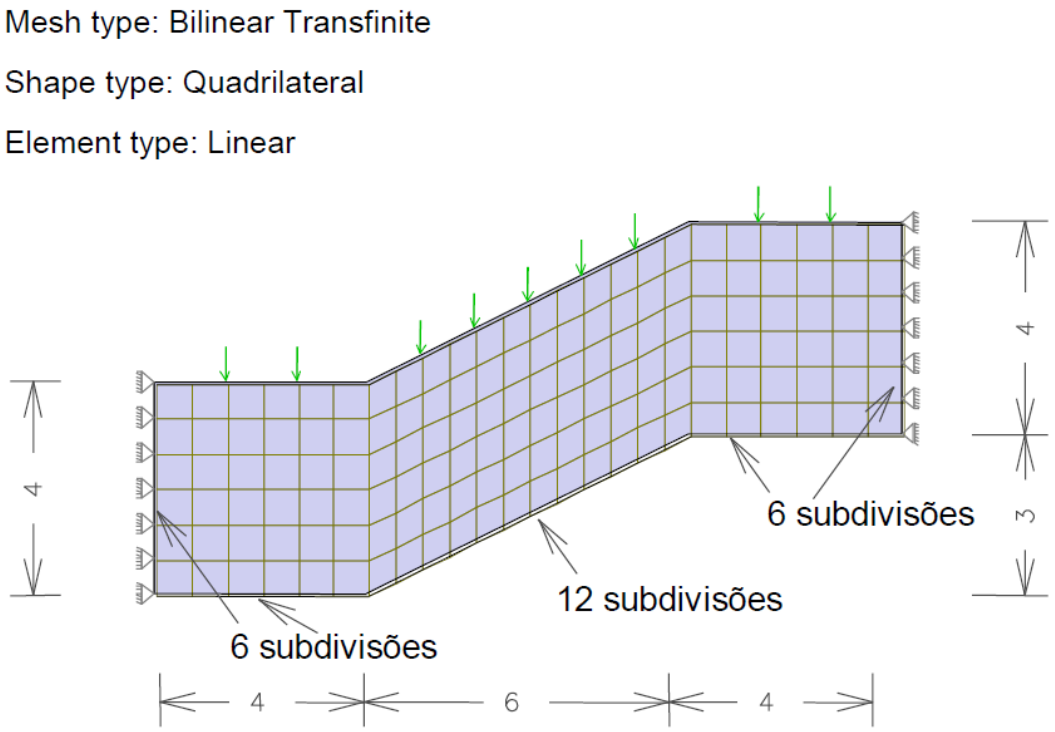


Figura 6.2 – Malha utilizada junto ao modelo inicial.

Após a conclusão do modelo inicial, são aplicadas modificações no modelo através da inserção de furos conforme mostrado na Figura 6.3. Em seguida é pedido às equipes que gerem dois tipos de malha para o modelo modificado. A primeira malha para esse modelo é mostrada na Figura 6.4 e a segunda malha é mostrada na Figura 6.5.

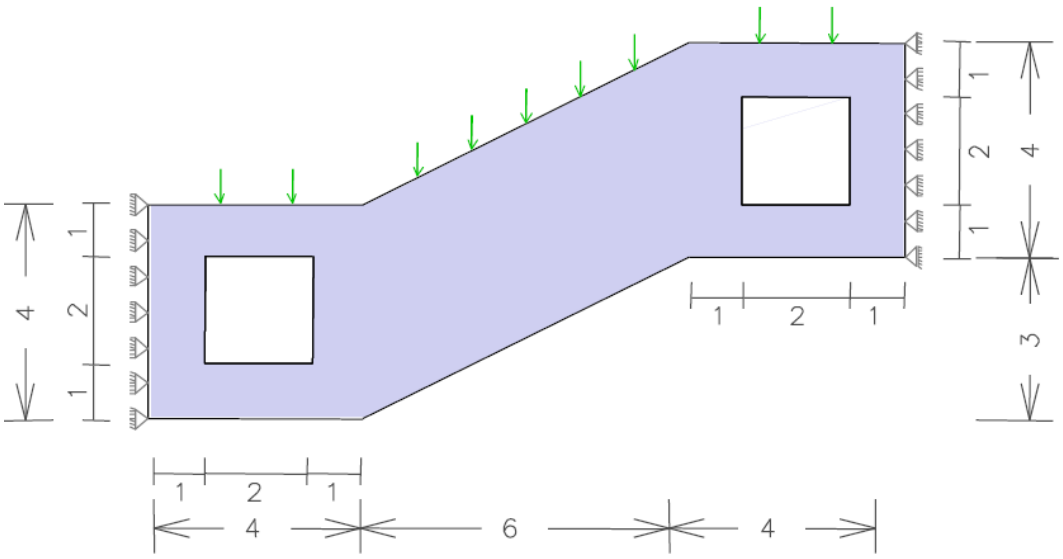


Figura 6.3 – Modelo modificado com furos.

6 subdivisões nas linhas das bordas dos furos

Mesh type: Triangular Boundary Contraction

Element type: Linear

Algorithm type: Optimal

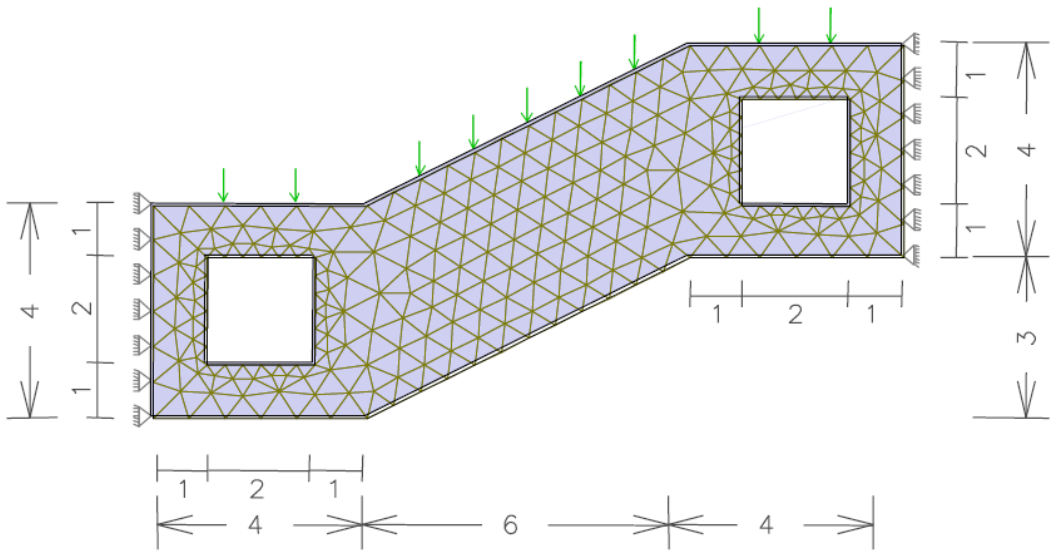


Figura 6.4 – Primeira malha para o modelo modificado.

Mesh type: Bilinear Transfinite

Shape type: Quadrilateral

Element type: Linear

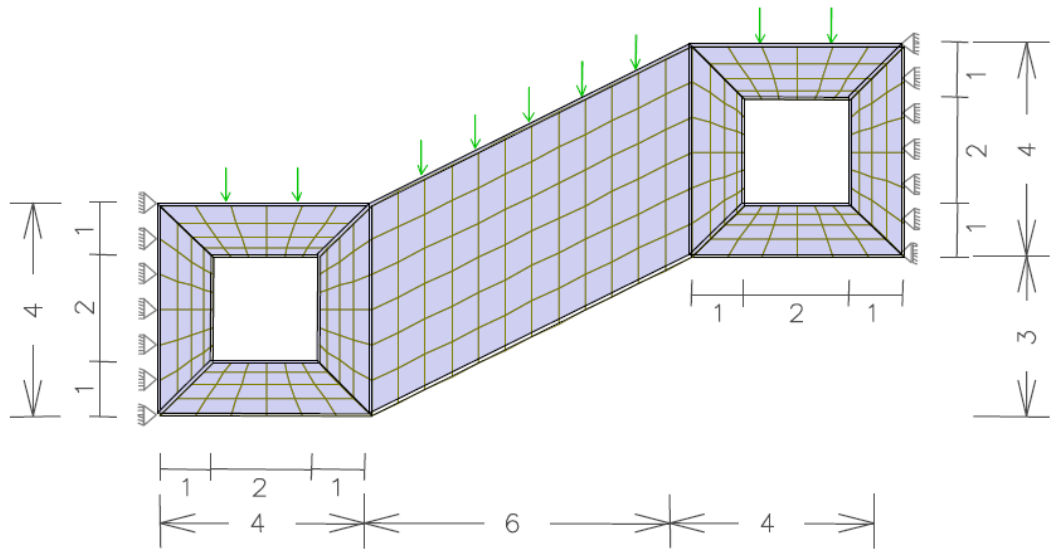


Figura 6.5 – Segunda malha para o modelo modificado.

6.2.
Questionário

O questionário aplicado após as equipes concluírem o experimento foi adaptado de ZISSIS et al. (2017) e possui perguntas com intuito de obter informações demográficas, subjetivas, facilidade de uso da aplicação, utilidade, intenção de uso e acessibilidade da estratégia de modelagem colaborativa proposta. A Tabela 6 mostra as perguntas, os padrões de resposta e os resultados obtidos em porcentagem referentes ao questionário para cada categoria elencada anteriormente.

Apesar de o questionário não possuir espaço para *feedbacks*, foram coletadas opiniões dos participantes e são apresentadas posteriormente na próxima Seção.

Tabela 6 – Questionário de avaliação de desempenho.

Categoria	Pergunta	Padrão de resposta	Porcentagem
Demográfica	Qual seu nível de educação?	Graduação incompleta	0
		Graduação completa	5,6%
		Pós-graduação	94,4%
	Quão experiente você é no uso de softwares CAD/CAE?	Nenhum	0
		Menos que 12 meses	5,6%
		Mais que 12 meses	11,1%
		Mais de 3 anos	27,7%
		Mais de 10 anos	55,6%
	Você possui acesso à internet em casa?	Sim	100%
		Não	0
Subjetiva	Quando na faculdade ou no trabalho você trabalha em colaboração com outros?	Sim	94,4%
		Não	5,6%
	Você acha ferramentas de colaboração importantes?	Sim	100%
		Não	0
Facilidade de uso	Você achou o sistema de fácil utilização?	Sim	94,4%
		Não	5,6%
	Foi fácil aprender a utilizar a aplicação?	Sim	100%
		Não	0

Tabela 7 – Questionário de avaliação de desempenho (continuação).

	Você achou o acesso à aplicação fácil (por meio do browser)?	Sim	100%
		Não	0
Utilidade da estratégia de colaboração remota	Você acha que a aplicação pode aumentar sua eficiência e produtividade?	Sim	88,9%
		Não	11,1%
	Você acha que a aplicação facilita o processo de modelagem?	Sim	100%
		Não	0
	Para o problema proposto, a estratégia de colaboração contribuiu para a solução do problema?	Sim	88,9%
		Não	11,1%
	Você acha que a aplicação proposta melhoraria processos colaborativos?	Sim	100%
		Não	0
	Você acha que a estratégia de modelagem colaborativa poderia ser implantada em ambientes educacionais?	Sim	100%
		Não	0
	Você utilizaria aplicações similares em caso de possuir computador com baixo recursos computacionais?	Sim	100%
		Não	
Intenção	Você gostaria de ficar informado acerca do desenvolvimento de novas funcionalidades?	Sim	100%
		Não	0
Acessibilidade	Você teve alguma dificuldade em acessar e utilizar a aplicação	Sim	11,1%
		Não	88,9%

Dessa forma, o experimento foi executado por pessoas que majoritariamente possuíam pós-graduação (94,4%) e que possuíam experiência com aplicações CAD/CAE maior que 12 meses (94,4%). E todos possuíam acesso à internet em casa.

A maioria dos participantes quando no trabalho ou na faculdade trabalham de forma colaborativa (94,4%) e acham ferramentas de colaboração importantes (100%).

Com relação a facilidade e acesso e uso da aplicação, 94,4% dos participantes acharam a aplicação de fácil utilização. Além disso, 100% dos participantes

acharam fácil aprender como utilizar a aplicação e acharam fácil o acesso por meio do navegador.

Com relação a utilidade da estratégia proposta, a maioria dos participantes achou que para o exemplo de modelagem proposto a estratégia contribuiu para solução do problema (88,9%), e todos concordaram que a estratégia pode facilitar o processo de modelagem através da colaboração. Todos os participantes acharam que a estratégia tem potencial de ser implantada em ambientes educacionais, e em caso de possuírem computadores de baixo recurso computacional utilizariam aplicações similares.

Alguns participantes possuíram dificuldade em acessar e utilizar a aplicação (11,1%), porém a maioria conseguiu utilizar sem maiores dificuldades (88,9%).

Além disso, os participantes demonstraram interesse em receber atualizações acerca do andamento da pesquisa e desenvolvimento da aplicação.

6.3. Feedbacks

Após a realização do experimento, algumas equipes realizaram *feedbacks* acerca da utilização da aplicação. Essas respostas foram coletadas e demonstram importância por meio de sugestões para o aprimoramento da estratégia e para os casos de uso da a aplicação desenvolvida, por esse motivo são registradas nessa Seção.

Durante a modelagem foram relatados problemas de conexão. Apesar de não ser a única possibilidade, devido à aplicação estar hospedada, conforme explicado no Capítulo 5, em uma opção gratuita da plataforma *Heroku*, pode haver problemas de conexão quando o fluxo for alto ou até mesmo quando o servidor estiver indisponível por não haver redundância[3]. Além disso, a aplicação no estágio atual de desenvolvimento não possui persistência dos modelos, no momento de desconexão o modelo é perdido e o usuário necessita realizar o processo de modelagem do início.

[3] Em computação em nuvem redundância é utilizado no sentido de manter a disponibilidade do sistema através de várias instâncias e redirecionar os acessos no caso de alguma instância enfrentar fluxo alto ou até mesmo estiver indisponível.

Com relação à colaboração, os participantes relataram que recursos de voz poderiam potencializar a comunicação e facilitar o processo de modelagem colaborativa. Não há a possibilidade de verificar quem conectou ou não à sala, apenas caso insiram alguma mensagem no *chat* da aplicação, o que dificulta saber se todos conseguiram conectar ou se estão enfrentando algum problema.

Nessa versão inicial da aplicação, a seleção de entidades geométricas foi tratada como global; no caso de um usuário selecionar uma entidade todos os outros usuários recebem o *feedback* visual da seleção da entidade geométrica selecionada. Essa forma de tratamento dificultou o processo de modelagem de alguns participantes, pois quando outro usuário selecionar outra entidade geométrica a anterior, que outro usuário estava editando, seria deselecionada o que pode ocasionar erros na aplicação de atributos.

7
Conclusão

Através de uma revisão sistemática foi possível encontrar lacunas no campo de pesquisa de aplicações colaborativas de modelagem geométrica. Trabalhos de diversas áreas do conhecimento possuíram foco na parte de visualização em navegadores, sem colaboração. Em alguns trabalhos foi implementada colaboração, porém sem possibilidade de funcionamento em tempo real, ou seja, há a necessidade de atualizar o navegador a cada atualização dos modelos.

Dessa forma, este trabalho foi guiado a partir da revisão sistemática e dos questionamentos iniciais, os quais eram:

1. Softwares de modelagem geométrica e análise estrutural podem ser utilizados de forma eficaz em ambientes baseados na *web*?
2. Como pode ser inserida a funcionalidade de colaboração em tempo real nessas aplicações?
3. Quais os requisitos para que esses programas possam ser adotados por um público maior?

Inicialmente, através da tecnologia do *WebGL* foi possível renderizar as informações geométricas dos modelos gerados com rapidez e eficácia nos navegadores mais usuais atualmente.

A tecnologia *WebGL* utiliza programas denominados *Shaders* que são compilados e funcionam na placa gráfica do processo cliente para renderizar dados na tela. A linguagem *JavaScript*, utilizada para o desenvolvimento da camada cliente, fornece uma interface para o *WebGL* o que solucionou a questão de renderização no navegador.

Somado a isso, através da biblioteca *React* são construídas interfaces de usuário com facilidade devido a possibilidade de criação e reutilização de componentes, reduzindo a quantidade de código necessário para o desenvolvimento da interface.

Com isso, torna-se possível o desenvolvimento de aplicações de modelagem geométricas de forma eficaz para ambientes *web.*

Através do protocolo *WebSocket* é possível a criação de um canal de comunicação bilateral eficiente para a comunicação entre os processos clientes e o servidor. O *WebSocket* permite, diferentemente do protocolo HTTP, que qualquer um dos processos inicie o transporte de dados, com isso é possível que o servidor envie mensagens para o cliente, sem a necessidade de o cliente enviar uma requisição. Esse fator é extremamente importante para a colaboração entre diferentes usuários, pois após a alteração na estrutura de dados o servidor pode enviar essa atualização para todos os clientes conectados à mesma sala virtual.

O terceiro questionamento é analisado após a avaliação da estratégia proposta através da prova de conceito desenvolvida. Nessa etapa foi possível mapear os pontos fortes e os pontos de melhoria da estratégia adotada.

A interação dos usuários com a estratégia de modelagem colaborativa pode ser aprimorada através do mapeamento de interações que podem ser feitas de forma síncrona. Por exemplo, usuários podem ter a seleção de entidades geométricas tratadas como própria do usuário, implementando as regras de negócio necessárias para manter a coesão do modelo.

A adição de recursos de voz é um fator importante como forma de potencializar a colaboração dos usuários, que atualmente possuem apenas o *chat* da aplicação para comunicação dentro da aplicação.

Por fim, relacionada à arquitetura adotada, para evitar desconexões e indisponibilidade da aplicação pode ser aplicada redundância na camada do servidor e realizar o redirecionamento do fluxo de mensagens da camada cliente para outros servidores disponíveis e com menor fluxo.

Por fim, a estratégia proposta demonstrou bastante flexibilidade e potencial para utilização, principalmente em ambientes educacionais. A aplicação é multiplataforma e funciona nos navegadores mais modernos, inclusive de dispositivos móveis, apesar de que ainda é necessário aprimorar a responsividade da aplicação cliente para que a experiência em dispositivos móveis seja similar à experiência em computadores pessoais.

Referências

ANGEL, E.; SHREINER, D. **Interactive Computer Graphics. A Top-Down Approach with WebGL**. 7. ed. Pearson, 2014.

BANKS, A.; PORCELLO, E. **Learning React Modern Patterns for Developing React Apps**. 2. ed. O'Reilly, 2020.

BONFIM, D. S.; MARTHA, L. F. **Uma estratégia de modelagem aberta e extensível para criação de modelos de subdivisões planares para mecânica computacional**. Rio de Janeiro, 2021. 148p. Dissertação de Mestrado - Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

DIEZ, H. V. et al. 3D model management for e-commerce. **Multimedia Tools and Applications**, v. 76, n. 20, p. 21011–21031, 1 out. 2017.

ERVIK, Å.; MEJÍA, A.; MÜLLER, E. A. Bottled SAFT: A Web App Providing SAFT-γ Mie Force Field Parameters for Thousands of Molecular Fluids. **Journal of Chemical Information and Modeling**, v. 56, n. 9, p. 1609–1614, 26 set. 2016.

GOUGH, D.; OLIVER, S.; THOMAS, J. **SYSTEMATIC REVIEWS**.

HASIUK, F. J. et al. TouchTerrain: A simple web-tool for creating 3D-printable topographic models. **Computers and Geosciences**, v. 109, p. 25–31, 1 dez. 2017.

HU, Z. Z. et al. Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with web-based tools. **Automation in Construction**, v. 66, p. 29–42, 1 jun. 2016.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down**. 6. ed. Pearson, 2013.

LI, L. et al. **Rendering Optimization for Mobile Web 3D Based on Animation Data Separation and On-Demand Loading.** IEEE Access, v. 8, p. 88474–88486, 2020.

LI, X. et al. **Application of 3D Printing and WebGL-Based 3D Visualisation Technology in Imaging Teaching of Ankle Joints**. Journal of Shanghai Jiaotong University (Science), v. 26, n. 3, p. 319–324, 1 jun. 2021.

LOMBARDI, A. **WebSocket: lightweight client-server communications**. 1. ed. O'Reilly, 2015.

LUPAS, A. N.; GRUBER, M. **The Structure of α-Helical Coiled Coils**. In: **Fibrous Proteins: Coiled-Coils, Collagen and Elastomers**. Advances in Protein Chemistry. Academic Press, 2005. v. 70p. 37–38.

MATSUDA, K.; LEA, R. **Webgl Programming Guide: Interactive 3D Graphics Programming with Webgl**. 1. ed. Addison-Wesley Professional, 2013.

MIRANDA, A.; MARTHA, L. F. **Uma Biblioteca Computacional para Geração de Malhas Bidimensionais e Tridimensionais de Elementos Finitos.** Proceedings of the XXI CILAMCE – 21st Iberian Latin-American Congress on Computational Methods in Engineering. Rio de Janeiro, 2000.

NYAMSUREN, P. et al. **A web-based collaborative framework for facilitating decision making on a 3D design developing process**. Journal of Computational Design and Engineering, v. 2, n. 3, p. 148–156, 1 jul. 2015.

PERKINS, S. J. et al. **Atomistic modelling of scattering data in the collaborative Computational Project for Small Angle Scattering (CCP-SAS).** Journal of Applied Crystallography, v. 49, n. 6, p. 1861–1875, 1 dez. 2016.

ROBLEDO, J. et al. **From video games to solar energy: 3D shading simulation for PV using GPU**. Solar Energy, v. 193, p. 962–980, 15 nov. 2019a.

ROBLEDO, J. et al. **From video games to solar energy: 3D shading simulation for PV using GPU**. Solar Energy, v. 193, p. 962–980, 15 nov. 2019b.

SALEHI, V.; WANG, S. Web-based visualization of 3D factory layout from hybrid modeling of CAD and point cloud on virtual globe DTX solution. **Computer-Aided Design and Applications**, v. 16, n. 2, p. 243–255, 2019.

TAO, A. et al. EzCADD: A Rapid 2D/3D Visualization-Enabled Web Modeling Environment for Democratizing Computer-Aided Drug Design. **Journal of Chemical Information and Modeling**, v. 59, n. 1, p. 18–24, 28 jan. 2019.

TRANFIELD, D.; DENYER, D.; SMART, P. **Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic Review.** British Journal of Management. V. 14, n. 3. p. 207-222. 16 set. 2003.

WOOD, C. W.; WOOLFSON, D. N. CCBuilder 2.0: Powerful and accessible coiled-coil modeling. **Protein Science**, v. 27, n. 1, p. 103–111, 1 jan. 2018.

XIA, J. Y. et al. Interactive WebGL-based 3D visualizations for EAST experiment. **Fusion Engineering and Design**, v. 112, p. 946–951, 15 nov. 2016.

XU, Z.; ZHANG, Y.; XU, X. 3D visualization for building information models based upon IFC and WebGL integration. **Multimedia Tools and Applications**, v. 75, n. 24, p. 17421–17441, 1 dez. 2016.

ZHANG, S.; JIANG, P. **Implementation of BIM + WebGIS Based on Extended IFC and Batched 3D Tiles Data: An Application in RCC Gravity Dam for Republication of Design Change Model**. KSCE Journal of Civil Engineering, v. 25, n. 11, p. 4045–4064, 1 nov. 2021.

ZHAO, S. **Using artificial neural network and WebGL to algorithmically optimize window wall ratios of high-rise office buildings**. Journal of Computational Design and Engineering, v. 8, n. 2, p. 638–653, 1 abr. 2021.

ZISSIS, D. et al. **Collaborative CAD/CAE as a cloud service.** International Journal of Systems Science: Operations and Logistics, v. 4, n. 4, p. 339–355, 2 out. 2017.