UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

SUPORTE À AVALIAÇÃO DE PLANOS DE EMERGÊNCIA UTILIZANDO TÉCNICAS DE TESTE DE ENGENHARIA DE SOFTWARE

Igor Marracho Carriço César
Magno Nascimento de Azevedo Cruz

Orientadora
Bruna Diirr Gonçalves da Silva

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2020

Catalogação informatizada pelo(a) autor(a)

N955 Nascimento de Azevedo Cruz, Magno
Suporte à Avaliação de Planos de Emergência Utilizando Técnicas de Teste de Engenharia de Software / Magno Nascimento de Azevedo Cruz. -- Rio de Janeiro, 2020.
55

Orientadora: Bruna Diirr.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2020.

1. Gestão de Emergência. 2. Teste de Mutação. 3. Revisão Técnica Formal. 4. NodeJS. 5. ReactJS. I. Diirr, Bruna, orient. II. Título.

M421 Marracho Carriço César, Igor
Suporte à Avaliação de Planos de Emergência Utilizando Técnicas de Teste de Engenharia de Software / Igor Marracho Carriço César. -- Rio de Janeiro, 2020.
55

Orientadora: Bruna Diirr.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2020.

1. Gestão de Emergência. 2. Teste de Mutação. 3. Revisão Técnica Formal. 4. NodeJS. 5. ReactJS. I. Diirr, Bruna, orient. II. Título.

SUPORTE À AVALIAÇÃO DE PLANOS DE EMERGÊNCIA UTILIZANDO TÉCNICAS DE TESTE DE ENGENHARIA DE SOFTWARE

Igor Marracho Carriço César
Magno Nascimento de Azevedo Cruz

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

BRUNA DIIRR GONÇALVES DA SILVA (UNIRIO)

JOBSON LUIZ MASSOLLAR DA SILVA (UNIRIO)

TADEU MOREIRA DE CLASSE (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.
FEVEREIRO DE 2020

RESUMO

Emergências são situações complexas e imprevisíveis, que podem ter impactos negativos em pessoas, serviços e/ou ambientes. Uma das formas de se preparar para esse tipo de situação é por meio de um plano de emergência. Os planos de emergência têm como objetivo orientar as equipes envolvidas durante a resposta à determinada situação de emergência, estabilizando assim essa situação ou pelo menos minimizando seus danos.

Por se tratar de uma abordagem que visa prevenir grandes impactos, é de suma importância que os planos sejam devidamente testados e avaliados para diminuir as chances de que possíveis problemas ocorram e acarretem em mais perdas. Entretanto, gerar e avaliar um plano de emergência pode ser caro e demandar bastante tempo.

Esse trabalho se utiliza dos conceitos de teste de software para auxiliar na geração e avaliação de planos de emergência e, desta forma, possibilitar uma alternativa mais prática e menos custosa. Desta forma um sistema web foi desenvolvido com o objetivo de oferecer apoio à abordagem proposta.

Palavras-chave: Gestão de emergência, Teste de mutação, Revisão Técnica Formal, NodeJS, ReactJS.

ABSTRACT

Emergencies are complex and unpredictable situations, which can have negative impacts on people, services and / or environments. One way to prepare for this type of situation is through an emergency plan. Emergency plans aim to guide the teams involved during the response to a given emergency situation, thus stabilizing the situation or at least minimizing its damage.

As it is an approach that aims to prevent major impacts, it is of utmost importance that the plans are properly tested and evaluated to reduce the chances of possible problems occurring and result in more losses. However, generating and evaluating an emergency plan can be expensive and time-consuming.

This work uses the concepts of software testing to assist in the generation and evaluation of emergency plans and, in this way, enable a more practical and less costly alternative. In this way, a web system was developed with the aim of offering support to the proposed approach.

Keywords: Emergency management, Mutation Testing, Formal Technical Review, NodeJS, ReactJS.

Sumário

1 INTRODUÇÃO 10

2 REFERENCIAL TEÓRICO 13

2.1 GESTÃO DE EMERGÊNCIA 13

2.2 REVISÃO TÉCNICA FORMAL 15

2.3 TESTE DE MUTAÇÃO 18

3 PROPOSTA DE TRABALHO 21

3.1 AVALIAÇÃO DE PLANOS DE EMERGÊNCIA UTILIZANDO TÉCNICAS DE TESTE 21

3.2 SUPORTE À AVALIAÇÃO DE PLANOS DE EMERGÊNCIA ATRAVÉS DE TÉCNICAS DE TESTE 25

4 DESENVOLVIMENTO DA APLICAÇÃO 28

4.1 INTRODUÇÃO À ARQUITETURA DO SISTEMA 28

4.2 TECNOLOGIAS ADOTADAS 29

4.2.1 React 29

4.2.2 Node.js 30

4.2.3 MongoDB 33

5 IMPLEMENTAÇÃO 43

5.1 SISTEMA PROPOSTO 43

5.2 NAVEGAÇÃO DO APLICATIVO WEB 44

5.2.1 Seção de Introdução 45

5.2.2 Seção de Planejamento 45

5.2.3 Seção de Preparação Individual 49

6 CONCLUSÃO 53

6.1 CONSIDERAÇÕES FINAIS 53

6.2 TRABALHOS FUTUROS 54

Lista de Tabelas

Tabela 1: Funcionalidades para suporte tecnológico à abordagem. 26
Tabela 2: Status das funcionalidades para suporte tecnológico à abordagem.....54

Lista de Figuras

Figura 1: O ciclo da gestão de emergência. (adaptado de ALEXANDER, 2002 apud: COPPOLA, 2006) 14
Figura 2: ISO 9216 referente à qualidade de software. (International Standards Organization, 1993) 16
Figura 3: As seis fases sequenciais do processo da FTR. 17
Figura 4: Process of Mutation Testing (adaptado de JIA e HARMAN, 2011) 19
Figura 5: Abordagem de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação. 22
Figura 6: Exemplificação de possíveis cenários em um plano de emergência. 23
Figura 7: Exemplo de planos mutantes. 24
Figura 8: Arquitetura do sistema. 29
Figura 9: Ciclo de vida dos componentes React. 29
Figura 10: Exemplo de arquivo raiz do React e a sua manipulação do DOM. 30
Figura 11: Arquivo package.js do servidor Node.js mostrando os módulos de terceiros usados na construção da aplicação. 31
Figura 12: Arquivo app.js do servidor Node.js mostrando a instanciação e utilização do Express.js e alguns de seus recursos. 32
Figura 13: Arquivo app.js do servidor Node.js mostrando a instanciação e utilização do Express.js e alguns de seus recursos. 33
Figura 14: Documento BSON de um Plano de Emergência criado no MongoDB Atlas a partir do uso da aplicação. 34
Figura 15: Arquivo emergencyPlanController.js do servidor Node.js mostrando o controlador dos Planos de Emergência e algumas operações no banco MongoDB utilizando o Mongoose. 35
Figura 16: Arquivo index.js do servidor Node.js mostrando a instanciação Mongoose e alguns métodos, entre eles o de conexão direta com o MongoDB. 36
Figura 17: Os dados necessários para realizar a conexão com banco de dados hospedado no MongoDB Atlas. 37
Figura 18: Arquivo do modelo de Plano de Emergência do servidor Node.js mostrando o schema e os campos relacionados. 38
Figura 19: Arquivo do modelo de Plano Mutante do servidor Node.js mostrando o schema e os campos relacionados. 38

Figura 20: Arquivo do modelo de Procedimento do servidor Node.js mostrando o schema e os campos relacionados. 39
Figura 21: Arquivo do modelo de Recurso do servidor Node.js mostrando o schema e os campos relacionados. 39
Figura 22: Arquivo do modelo de Cenário do servidor Node.js mostrando o schema e os campos relacionados. 40
Figura 23: Arquivo do modelo de Passo do servidor Node.js mostrando o schema e os campos relacionados. 40
Figura 24: Estrutura de arquivos no repositório monorepo, explicitando servidor e cliente, e as configurações básicas do Lerna. 42
Figura 25: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação. 43
Figura 26: Página de introdução da aplicação com menu expandido. 44
Figura 27: Página de introdução da aplicação com o menu reduzido. 45
Figura 28: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação. 46
Figura 29: Página de criação de recursos. 46
Figura 30: Página de criação de passos 47
Figura 31: Página de criação de procedimento. 47
Figura 32: Página de criação de cenário 48
Figura 33: Página de criação de um plano de emergência. 49
Figura 34: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação. 49
Figura 35: Página de visualização de planos. 50
Figura 36: Modal de detalhes do plano de emergência. 50
Figura 37: Modal de criação de mutantes para um plano de emergência. 51
Figura 38: Modal de detalhes do plano de emergência com um mutante do plano. 52
Figura 39: Botões para marcar o estado de um mutante como equivalente ou morto. 52

1 Introdução

Emergências são situações complexas e imprevisíveis, que podem ter impactos negativos em pessoas, serviços e/ou ambiente. Embora não seja possível antecipar completamente tais situações, políticas e ações devem ser estabelecidas para reduzir a vulnerabilidade da população ou reduzir os impactos negativos de possíveis desastres (HADDOW et al, 2011; VUSSC, 2008).

A Gestão de Emergências é um processo cíclico constituído de três fases: (a) Pré-desastre (Mitigação e Preparação); (b) Resposta; e (c) Pós-desastre (HADDOW et al, 2011; KHAN et al, 2008; VUSSC, 2008). Este trabalho está focado na primeira fase da gestão de emergência, isto é, na fase de pré-desastre, mais especificamente durante a preparação na qual um plano de emergência é elaborado e gerado.

A construção de um plano de emergência se inicia com a especificação de um conjunto de cenários que o plano deve cobrir. Para cada cenário, um conjunto de passos (por exemplo, ligar para os bombeiros e evacuar a área) são designados para se tomar o controle da situação, significando retorná-la à situação estável com o mínimo de perdas possíveis. Por exemplo, em um incêndio em um prédio, os procedimentos focam em limitar a extensão do fogo, resgate das pessoas em risco, e extinção do fogo. O sucesso ou falha dessas ações geram outros cenários para os quais um novo conjunto de passos deve ser realizado e assim por diante (LLAVADOR et al, 2006).

Um dos objetivos da avaliação de um plano de emergência é determinar se os passos descritos para um cenário específico são os mais apropriados e controlam a situação efetivamente. Outro objetivo é assegurar que todos os cenários prováveis estejam contemplados no plano, reduzindo a necessidade de uma improvisação. Por fim, há um terceiro objetivo, não tão óbvio, que se refere à padronização de passos com o objetivo de facilitar o treinamento e utilização dos recursos disponíveis (LEY et al., 2014; VUSSC, 2008).

Uma possível forma de avaliar planos antes que a emergência ocorra é por meio de simulação. Ela permite examinar como os procedimentos, recursos e equipe treinada

respondem aos cenários imaginados, permitindo identificar onde as mudanças nos procedimentos, a compra de mais ou melhores recursos ou melhora no treinamento são necessárias (HADDOW et al, 2011; VUSSC, 2008). Entretanto, apesar do potencial da simulação, ela pode ser muito cara e demandar a construção de ambientes e ferramentas complexas. Com isso, antes de enviar um plano de emergência para avaliação, todos os esforços devem ser tomados para reduzir erros básicos ou omissões no plano.

Há dois problemas básicos que devem ser cuidadosamente tratados antes de se iniciar um exercício de simulação: cenários faltantes e procedimentos não apropriados ou ineficazes. Quando o plano é elaborado por especialistas, eles tentam prever todos os cenários possíveis. Isso é particularmente difícil para cenários resultantes de uma evolução inesperada da emergência ou de ações ineficazes. Outro problema comum está em incluir procedimentos que não são necessários para certos cenários causando trabalho extra desnecessário para a equipe de resposta à emergência (LEY et al., 2014; VUSSC, 2008).

Diirr e Borges (2013) argumentam que a adoção de técnicas de teste da Engenharia de Software poderia ser útil na avaliação do plano de emergência, ajudando assim a melhorar tal plano e consequentemente os efeitos negativos de uma emergência, principalmente em ambientes com menos recursos para simulação. É argumentado que as técnicas utilizadas nos testes de software poderiam ser úteis na avaliação dos planos de emergência, em particular, para os dois primeiros objetivos. Assim, foram selecionadas duas técnicas que se acredita serem úteis no contexto de emergência: Revisão Técnica Formal (FTR) e Testes de Mutação (CAMPANHA et al, 2010; PRESSMAN, 2006; SOMMERVILLE, 2003).

Para avaliar se os passos para um cenário específico são adequados, é proposto o uso da FTR, que foca em assegurar a adequação do artefato ao encorajar participação e colaboração de experts técnicos que seguem procedimentos formais para encontrar erros e anomalias. E para determinar se um plano de emergência cobre um conjunto de cenários possíveis, é sugerido uma técnica baseada no teste de mutação, que propõe a inclusão de erros nos artefatos para identificar omissões e procedimentos dispensáveis em um plano (DIIRR & BORGES, 2013).

Este trabalho é uma abordagem prática com o objetivo de evitar erros e consequentemente diminuir os gastos na elaboração de um plano de emergência e desta forma auxiliar na avaliação de tais planos de emergência. Para isso, foi realizado um levantamento sobre gestão de emergência e uma aplicação que fosse capaz de auxiliar na elaboração e avaliação do plano de emergência foi desenvolvida. Essa aplicação foi desenvolvida com foco em web, utilizando as tecnologias mais atuais, e visa se tornar uma ferramenta que apoia às equipes na etapa de preparação do ciclo de gestão de emergências.

O presente trabalho está estruturado em capítulos e, além desta introdução, foi desenvolvido da seguinte forma: o Capítulo 2 apresenta o levantamento realizado sobre gestão de emergência e técnicas de testes da Engenharia de Software que podem auxiliar no processo de criação de um plano de emergência apropriado. O Capítulo 3 detalha a proposta de trabalho que foi realizada com base no referencial teórico. No Capítulo 4 é apresentada a especificação do sistema, resumindo a arquitetura projetada e as tecnologias adotadas para o desenvolvimento. O Capítulo 5 mostra o resultado da implementação com base na proposta realizada. Por fim, o Capítulo 6 reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior do trabalho.

2 Referencial Teórico

Neste capítulo serão abordados os temas relacionados a essa monografia. Os conceitos, métodos e técnicas serão apresentados, assim como os trabalhos e pesquisas relacionados.

2.1 Gestão de Emergência

Uma emergência é um fenômeno complexo, imprevisto e não completamente antecipado (LEY et al., 2014; VUSSC, 2008). As emergências são eventos de risco que resultam em perdas econômicas, sociais e humanas. Tais perdas são influenciadas por diversos aspectos, tais como: frequência, impacto, percepção social, complexidade de comunicação e ações de resposta aos eventos. A alta dificuldade de analisar as causas iniciais destes desastres (o ambiente promotor e as causas secundárias dos eventos principais), está na urgência de tomar ações preventivas ou corretivas (ÁVILA et al., 2020).

A partir da necessidade de tomar ações preventivas ou corretivas é que a gestão de emergência entra em ação. A gestão de emergência é um processo sistemático que visa minimizar os impactos negativos e consequências dos eventos adversos nas pessoas ou propriedades (LEY et al., 2014; VUSSC, 2008). I.e., é sabido que nem sempre é possível prevenir a ocorrência de emergências, mas seus efeitos podem ser minimizados se forem adotadas ações que permitam a compreensão das causas das emergências e como agir face suas potenciais consequências.

A gestão de emergência é composta por quatro etapas distintas (ALEXANDER, 2002 apud: COPPOLA, 2006): mitigação, preparação, resposta e recuperação, como visto na Figura 1. É importante ressaltar que tal Figura é uma generalização, pois exceções podem ser identificadas em cada etapa:

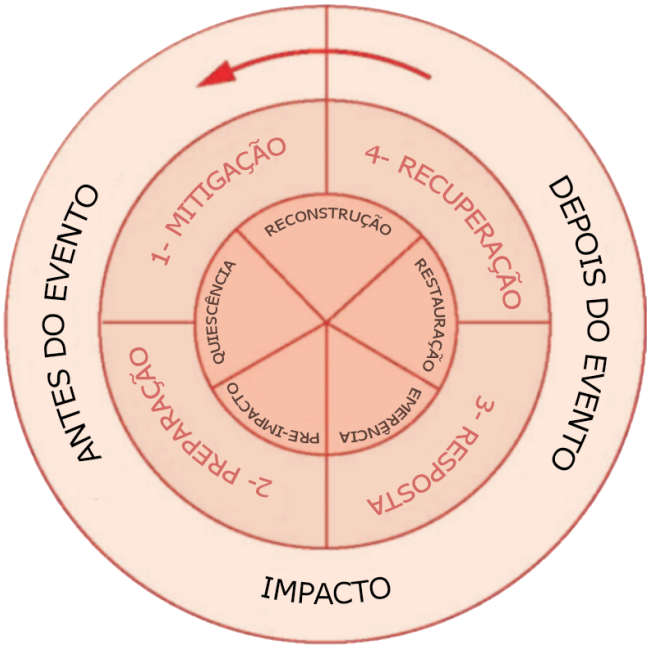


Figura 1: O ciclo da gestão de emergência. (adaptado de ALEXANDER, 2002 apud: COPPOLA, 2006)

1. *Mitigação*. É composta pelas fases de reconstrução e quiescência. A fase de reconstrução visa a avaliação de situações e respostas a desastres passados para aprender lições que sirvam de preparação para desastres futuros, já a fase de quiescência representa o tempo de calmaria entre desastres. Juntas essas fases compõem a mitigação, que envolve reduzir ou eliminar a probabilidade ou as consequências de um risco, ou ambos. A mitigação busca "tratar" o risco de forma que seu impacto seja de um grau menor.
2. *Preparação*. É composta pelas fases de quiescência e pré-impacto. A fase de quiescência representa o tempo de calmaria entre desastres, já a fase de pré-impacto visa desenvolver e implantar estratégias contra desastres futuros. Juntas essas fases compõem a preparação, que envolve equipar as pessoas que possam ser impactadas por um desastre ou que podem estar aptas a ajudar aqueles impactados. Podem ser utilizadas ferramentas que aumentem a chance de sobrevivência e minimização de perdas financeiras e de outros tipos.
3. *Resposta*. É composta pelas fases de emergência e restauração. A fase de emergência representa o período em que o desastre acontece, já a fase de restauração visa tomar ações para restaurar ou mitigar os impactos causados por desastres. Juntas essas fases compõem a resposta, que envolve a tomada de ações para reduzir ou eliminar o impacto dos desastres que tenham ocorrido ou que estão

ocorrendo atualmente. O intuito está em prevenir sofrimento adicional, perdas financeiras ou uma combinação de ambos.

4. *Recuperação*. É composta pelas fases de restauração e reconstrução. A fase de restauração visa tomar ações para restaurar ou mitigar os impactos causados por desastres, já a fase de reconstrução visa a avaliação de situações e respostas a desastres passados para aprender lições que sirvam de preparação para desastres futuros. Juntas essas fases compõem a recuperação, que envolve retornar as vítimas às suas vidas normais após o impacto do desastre. A fase de recuperação geralmente começa após a resposta imediata ter terminado, e pode persistir por vários meses ou anos.

2.2 Revisão Técnica Formal

Os softwares estão envolvidos em todos os aspectos da vida das pessoas e das organizações empresariais. Em softwares críticos, como sistema de aviação ou saúde, a correta e detalhada análise dos requisitos de software e a produção de produtos isentos de defeitos podem significar resguardar patrimônio financeiro, informações pessoais privadas e até vidas humanas (PRESSMAN, 2006). Portanto, assim como cresce a importância do software e a exigência por qualidade, a Engenharia de Software passa a desenvolver diversas abordagens para melhorar o desenvolvimento e produzir software com mais qualidade (PRESSMAN, 2006).

Qualidade e confiabilidade dos sistemas de software estão entre as maiores preocupações dos profissionais de computação. Nos anos 70, Barry Boehm, James McCall e seus colegas criaram modelos para medir a qualidade dos softwares, que acabaram sendo transformados em um padrão internacional, a ISO 9216, apresentado na Figura 2 (DENNING, 2016).

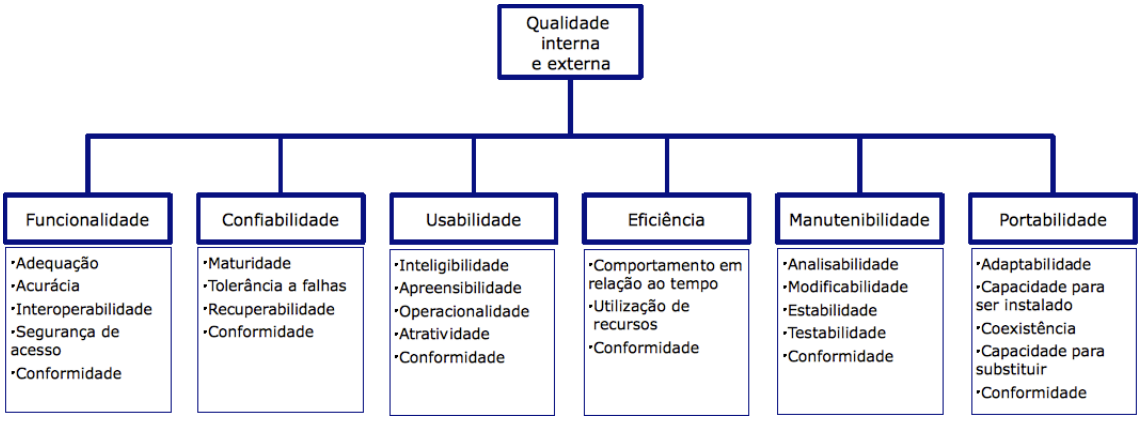


Figura 2: ISO 9216 referente à qualidade de software. (International Standards Organization, 1993)

Várias técnicas podem ser utilizadas para garantir a qualidade de software, sendo uma delas a Revisão Técnica Formal (*Formal Technical Review* - FTR) que por sua vez é um componente essencial de toda técnica de avaliação de qualidade de software, garantia e melhoria. A FTR sempre envolve utilizar um grupo de especialistas técnicos para analisar um artefato do processo de desenvolvimento de software. Eles seguem um conjunto de procedimentos formais, tipicamente com o objetivo de descobrir erros ou anomalias, garantindo assim que o artefato criado é consistente com sua especificação. A FTR sempre resulta em um documento estruturado especificando o resultado da revisão realizada (JOHNSON, 1994).

Os objetivos da FTR sobre a qualidade do software são (AINAPURE, 2014):

1. Descobrir erros em funções, lógica ou implementação para qualquer representação do software;
2. Verificar que o software sob revisão reconhece seus requisitos;
3. Assegurar que o software tenha sido representado de acordo com padrões predefinidos;
4. Conseguir ter o software desenvolvido de uma maneira uniforme e;
5. Tornar os projetos mais gerenciáveis.

O processo da FTR é composto por seis fases sequenciais (JOHNSON, 1994): planejamento, orientação, reunião de revisão, preparação individual, retrabalho e verificação, como visto na Figura 3.

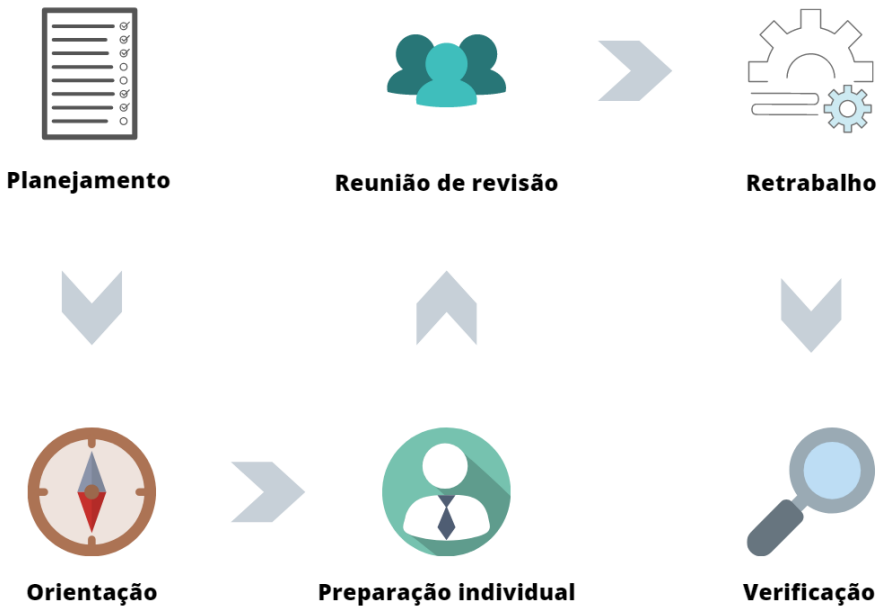


Figura 3: As seis fases sequenciais do processo da FTR.

1. *Planejamento*. Inclui a seleção do time, alocação de funções e definições dos critérios de entrada/saída.
2. *Orientação*. Inclui a distribuição de documentos e explicação dos objetivos/processos/documentos aos participantes.
3. *Preparação individual*. Onde os participantes tomam notas dos problemas do produto, dúvidas e comentários.
4. *Reunião de revisão*. Onde os participantes discutem sobre os possíveis problemas do produto e registram a aceitação/rejeição do produto.
5. *Retrabalho*. Envolve resolver os problemas identificados no produto.
6. *Verificação*. Envolve analisar se os problemas foram resolvidos, além da criação de métricas.

As reuniões da FTR possuem algumas restrições, algumas delas são (AINAPURE, 2014):

1. O número de membros da reunião deve ser entre três e cinco;
2. Cada pessoa deve se preparar para a reunião e não deve tomar de mais do que duas horas para o preparo;
3. A duração da reunião de revisão deve ser menor do que duas horas.

Estudos realizados na indústria sugerem que atividades de *design* introduzem de 50% a 60% de todos os defeitos ou erros durante o processo de criação de software. As técnicas de revisão têm se mostrado ter até 75% de efetividade em descobrir falhas de *design*, o que reduz o custo de atividades subsequentes no processo de software (AINAPURE, 2014).

2.3 Teste de Mutação

O desenvolvimento de software envolve uma série de atividades de produção onde as oportunidades de ocorrência de falhas humanas são enormes. Erros podem ocorrer desde a concepção do processo, onde os objetivos podem ser especificados com erros ou imperfeições, e da mesma forma, esses erros podem ocorrer em uma das etapas futuras de *design* ou desenvolvimento (PRESSMAN, 2006). Pela incapacidade humana de realizar ações ou se comunicar com perfeição, não é possível garantir que software funcione sem falhas. Por esse motivo, o desenvolvimento de software deve ser acompanhado sempre por atividades de garantia de qualidade (PRESSMAN, 2006).

O teste de software é um elemento crítico na garantia de qualidade de software e representa a revisão da especificação, *design* e geração de código. O aumento na visibilidade de software como um sistema e os custos associados com a falha de um software são forças motivantes para um bom planejamento através de testes. Não é incomum para uma empresa de desenvolvimento de software gastar entre 30% a 40% dos esforços totais do projeto apenas para a realização de testes. Em casos extremos, testes de softwares com classificação humana (e.g., controle de voo, monitoramento de reator nuclear) podem custar de três a cinco vezes mais do que todas as outras etapas de engenharia combinadas! (PRESSMAN, 2006)

As seguintes regras podem servir como exemplificação dos objetivos do teste (MYERS, G. apud: PRESSMAN, 2006):

1. Teste é um processo de execução de um programa com o intuito de revelar falhas;
2. Um bom caso de teste é um que possui uma alta probabilidade de revelar uma falha ainda não descoberta;
3. Um teste de sucesso é aquele que revela falhas ainda não descobertas.

Um exemplo de teste de software é o teste de mutação. Ele é baseado na inclusão de pequenos erros no software (por isso o nome mutante) e assim identificar possíveis erros pré-existentes. É uma técnica utilizada para avaliar a adequação, em teste do software, de um conjunto de casos de teste. Na Figura 4, está descrito o processo de teste de mutação para um operador mutante.

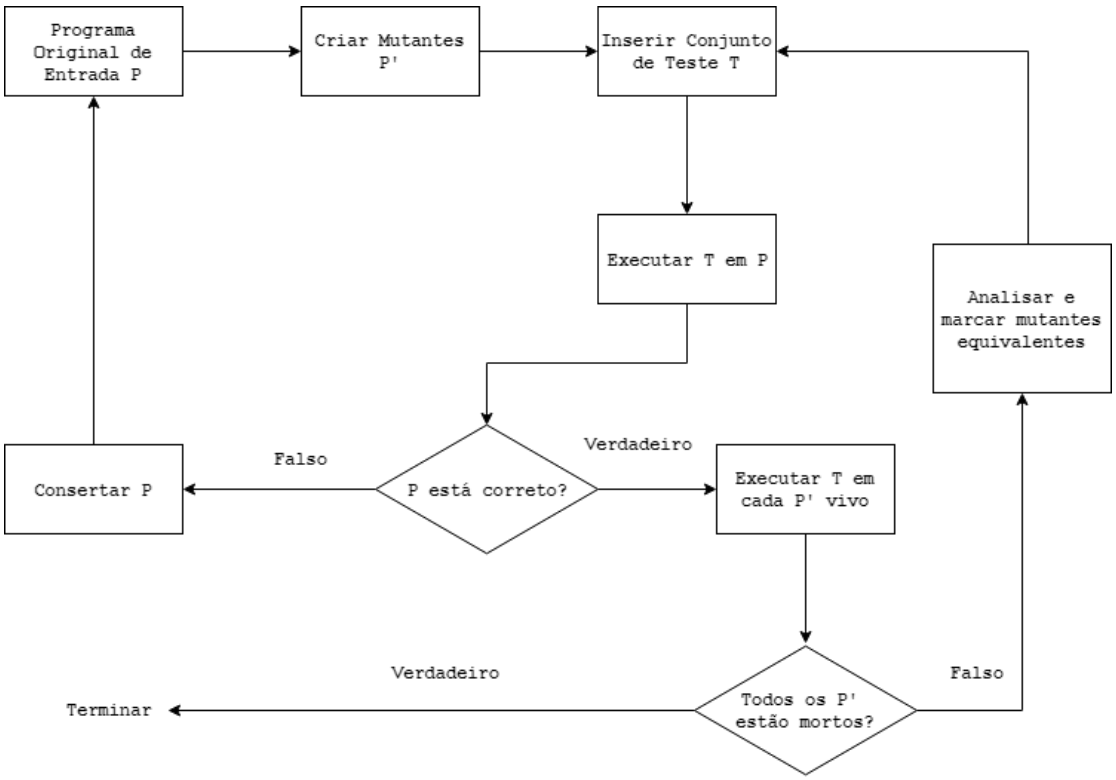


Figura 4: Processo de Teste de Mutação (adaptado de JIA e HARMAN, 2011)

Para executar o processo de teste de mutação, é necessário, inicialmente, atribuir um software de entrada “P” a um caso de teste. A partir deste software de entrada “P”, mutantes “P`” do software de entrada são gerados. Um conjunto de teste “T” é inserido e executado em “P” para verificar se o conjunto de teste é válido em “P”. Caso “P” esteja incorreto, o mesmo deve ser corrigido e o fluxo se inicia novamente, sendo gerado um novo software de entrada e novos mutantes. Caso “P” esteja correto, o conjunto de teste “T” é inserido e executado em todos os mutantes “P`” vivos. Caso o resultado contenha algum mutante “P`” não morto é necessário a análise e marcação de mutante equivalente. Neste caso, um novo conjunto de teste “T” é gerado e executado contra “P” e cada mutante “P`” até que todos os mutantes “P`” sejam marcados como mortos (JIA & HARMAN, 2011).

Para realçar as diferenças entre o software e o mutante, os resultados gerados por eles são avaliados. Se um caso de teste pode distinguir o mutante e o software, ao gerar saídas diferentes, nós os chamamos de “mortos”. Se o mutante gera o mesmo resultado que o software, ele é chamado de “equivalente”. Para esses mutantes equivalentes, não há casos de teste possíveis para eliminá-los, sendo assim necessário intervenção humana para distingui-los (FERREIRA, 2019). Com o intuito de realçar possíveis diferenças entre os equivalentes, novos casos de teste devem ser construídos. O foco do teste de mutação é eliminar todos os mutantes gerados porque, com isso, no final o melhor conjunto de casos de teste.

Para verificar se um conjunto de casos de teste é suficiente para avaliar o software, a pontuação de mutação é utilizada:

$$\text{pontuação de mutação} = \frac{\textit{número de mutantes morto}}{\textit{número de mutantes totais} - \textit{número de mutantes equivalentes}}$$

Quanto maior a pontuação mutante é, mais apropriado é o conjunto de casos de teste (CAMPANHA et al, 2010; DELAMARO et al, 2007; FRANZOTTE & VERGILIO, 2006; LIMAYE, 2009; PRESSMAN, 2006).

3 Proposta de trabalho

Neste capítulo será apresentada uma proposta de implementação completa que abrange todo o ciclo da gestão de emergência.

3.1 Avaliação de planos de emergência utilizando técnicas de teste

Diirr e Borges (2013) argumentam que é possível utilizar técnicas de teste de software para avaliação de planos de emergência. Foi feita uma analogia entre a Gestão de Emergências e a Engenharia de Software, onde o plano de emergência pode ser visto como o software que precisa ser testado e os diferentes cenários podem ser utilizados como massa de dados durante o teste. De forma equivalente à definição de casos de teste de software, é impossível identificar todos os cenários que podem ocorrer durante uma emergência. Argumenta-se que, se for possível determinar que o conjunto de cenários identificados pode revelar erros nos documentos, então é garantido que o plano de emergência funcione adequadamente nessas situações.

Assim, a proposta de avaliação de planos de emergência sugere o uso de técnicas de testes da Engenharia de Software, mais especificamente envolvendo o uso de testes de mutação durante as fases do FTR. Os passos dessa proposta podem ser vistos na Figura 5.

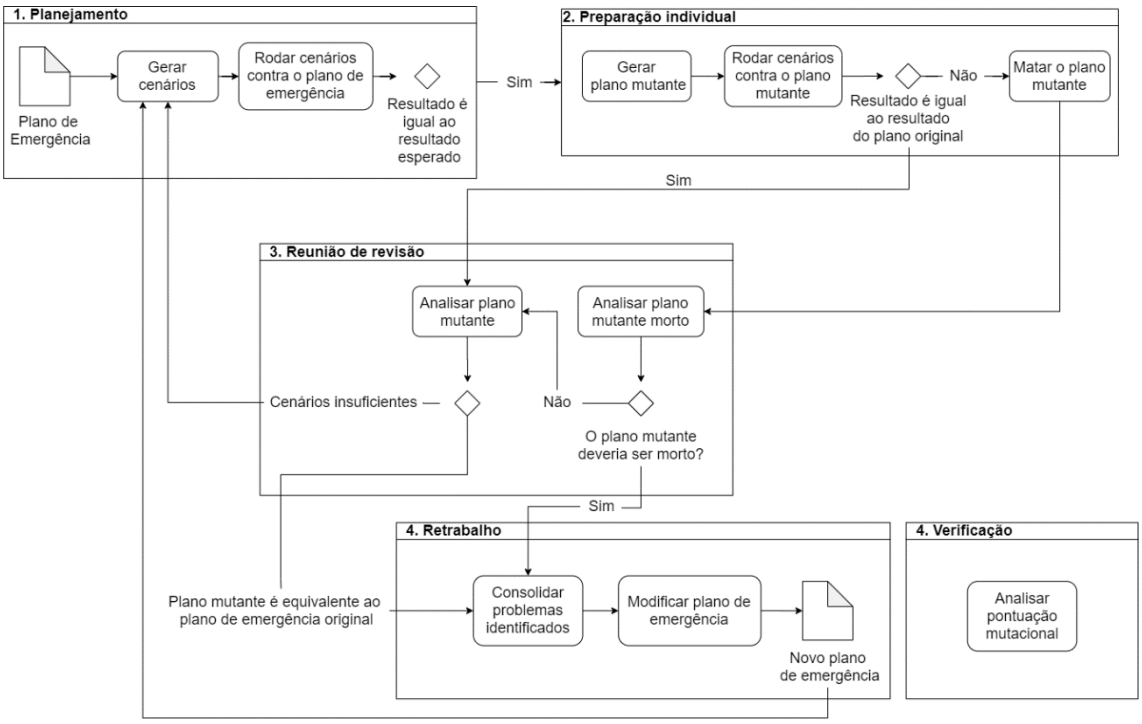


Figura 5: Abordagem de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação.

Durante o planejamento, um plano de emergência é recebido e o time de revisão gera um conjunto de cenários para serem utilizados durante os testes. Estes cenários representam situações que podem ser enfrentadas durante uma emergência e devem detalhar os procedimentos a serem tomados em tais situações. Estes cenários são aplicados ao plano de emergência e, se as saídas estiverem de acordo com o esperado, este conjunto de cenários devem ser executados com base nos planos mutantes. Em um exemplo apresentado na Figura 6, possíveis cenários para o plano podem ser incêndio, acidente químico, envenenamento e explosão.

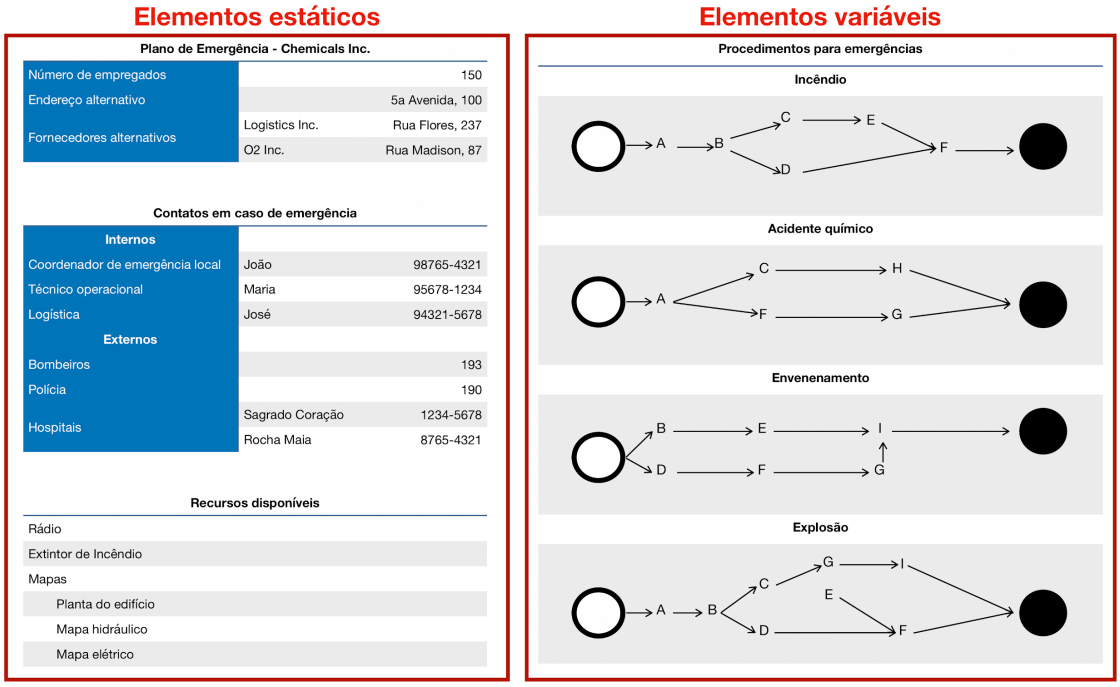


Figura 6: Exemplos de possíveis cenários em um plano de emergência.

Durante a preparação individual, os revisores geram um conjunto de planos mutantes através da remoção de partes do plano de emergência. Para auxiliar esta atividade, é proposta uma organização formal do plano de emergência em (a) **elementos estáticos**, informação que não varia de acordo com o cenário (por exemplo, infraestrutura da organização, riscos que podem afetar a organização, recursos disponíveis, etc.), e (b) **elementos variáveis**, informação que pode variar de acordo com o cenário (os procedimentos que devem ser aplicados durante a fase de resposta). A remoção de elementos estáticos serve para avaliar se os cenários existentes identificam mudanças no plano de emergência original. Se eles não identificarem, novos cenários que utilizem esses elementos precisam ser gerados. A remoção de elementos variáveis serve para avaliar se há ações desnecessárias sendo realizadas ou se faltam ações para tratar um cenário específico. Ao avaliar estes dois tipos de elementos, é possível assegurar que o plano de emergência apresenta todas as informações relevantes para os cenários e não apresenta qualquer informação que não é utilizada. A Figura 7 mostra o plano de emergência de uma companhia de produtos químicos organizado em elementos estáticos e variáveis.

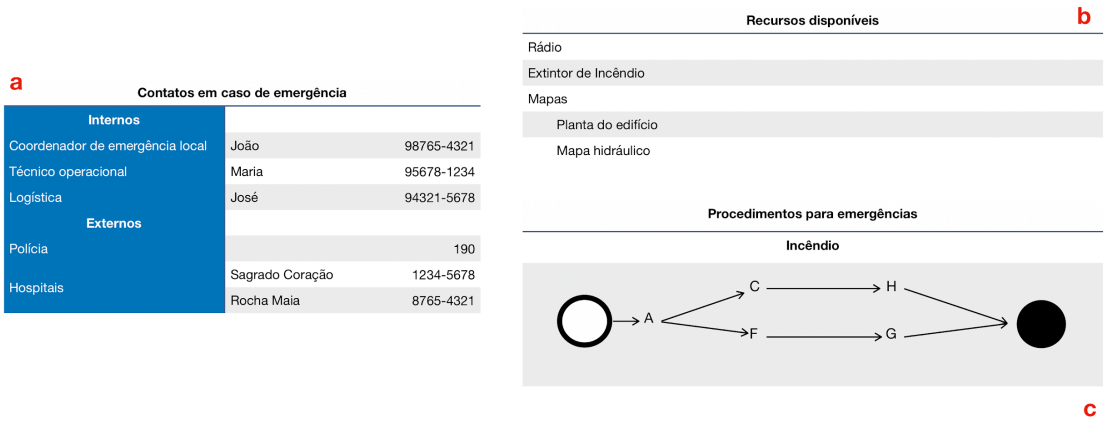


Figura 7: Exemplo de planos mutantes.

Ao remover alguns desses elementos, pode-se gerar três planos mutantes: “Plano Mutante 1” que não contém informações sobre o corpo de bombeiros (7a); “Plano Mutante 2” que não contém informações sobre o mapa elétrico (7b); e “Plano Mutante 3” que não contém as atividade necessárias para tratamento de acidentes químicos (7c).

Os revisores devem também aplicar os cenários nos planos mutantes e avaliarem as diferenças entre as saídas geradas pelo plano mutante e o plano de emergência original. No “Plano Mutante 1”, utilizando o primeiro cenário (incêndio), o plano não pode ser utilizado por falta de informações do corpo de bombeiros. Sendo assim, o plano mutante pode ser considerado “morto”, já que é possível distingui-lo do plano de emergência original por gerar saídas diferentes. Utilizando os mesmos cenários no “Plano Mutante 2”, não foi identificado qualquer problema ao tentar aplicar os cenários no plano. Os cenários também devem ser aplicados ao “Plano Mutante 3” e, como no “Plano Mutante 2”, nenhum problema foi identificado.

Durante a reunião de revisão, todos os planos mutantes são analisados. Se um plano mutante foi considerado “morto”, o time analisa se ele deve realmente manter esse status. Se um plano mutante foi morto por um erro ou não pode ser morto pelos cenários, o time verifica se o plano mutante pode ser classificado como “equivalente” ao plano original ou se é necessário gerar novos cenários para realçar as diferenças entre os planos. É importante enfatizar que a identificação de novos cenários é muito útil pois, ao aumentar o número de possíveis cenários para uma emergência específica, o escopo do plano de emergência é aumentado e reduz-se a necessidade de improvisação ou desenvolvimento de um plano de emergência no momento de resposta a um cenário não previsto.

Na análise do “Plano Mutante 1”, a equipe concorda com os revisores e considera o plano mutante como “morto”. Na análise do “Plano Mutante 2”, a equipe concluiu que o mapa elétrico é um recurso que deve ser utilizado em emergências que danificam as instalações elétricas da companhia. Entretanto, este cenário (acidentes elétricos) não foi explorado durante o projeto do plano de emergência, sendo identificado somente durante a avaliação do plano. Na avaliação do “Plano Mutante 3”, a equipe precisa entender se a atividade removida é dispensável ou se é necessário para um cenário que não foi identificado anteriormente. Eles concluem que o plano não está errado visto que, durante o projeto do plano de emergência, a equipe de planejamento decidiu agrupar todos os tipos de acidentes químicos em um único procedimento a ser aplicado. Assim, recomenda-se a separação do procedimento atual em procedimentos para cada possível acidente químico ou explicar que o procedimento atual pode ser utilizado para diferentes tipos de acidentes químicos, mesmo que algumas atividades sejam utilizadas somente para casos específicos.

Depois da reunião de revisão, todos os problemas identificados são enviados para a equipe de planejamento de emergência, que será responsável por resolver as inconsistências identificadas (fase de “Retrabalho”). Os planos de emergência revisados devem ser encaminhados novamente à equipe de revisão para aprovação. Além disso, a qualidade dos cenários é avaliada através da pontuação dos mutantes (fase de “Verificação”). Ele determina se a massa de dados (no caso, o conjunto de cenários) é suficiente para avaliar um plano de emergência e, como resultado, a confiabilidade deles. No exemplo acima, o plano de emergência original deve ser modificado para refletir todos os problemas identificados durante a revisão realizada (por exemplo, a falta de procedimentos em caso de acidentes elétricos e a falta de explicação que o procedimento pode ser utilizado em diferentes tipos de acidentes químicos). O novo plano de emergência será avaliado através dos passos descritos acima e pode auxiliar na identificação de novas inconsistências.

3.2 Suporte à avaliação de planos de emergência através de técnicas de teste

O software proposto para este trabalho tem como finalidade servir de apoio à avaliação de planos de emergências, recebendo como entrada o plano de emergência a ser avaliado e os possíveis cenários de emergência que podem ocorrer. Com isso, a aplicação deve ser capaz de dizer quais informações são desnecessárias e ajudar na identificação do

que está faltando para a completude do plano. Desta forma, é possível listar, por fases da abordagem, as funcionalidades desejáveis na Tabela 1.

Tabela 1: Funcionalidades para suporte tecnológico à abordagem.

Fase da abordagem	**Funcionalidade**
Planejamento	Desenvolvimento e/ou envio de planos de emergência que precisam ser avaliados: Os autores podem desenhar um novo plano de emergência, enviar um existente ou modificar seus planos utilizando a ferramenta.
	Formatar o plano de emergência a partir dos elementos estáticos e variáveis: A ferramenta deve apoiar a criação de planos de emergência utilizando elementos estáticos e variáveis. Se o plano de emergência não foi criado utilizando a ferramenta, deve ser possível formatar o plano de acordo com estes elementos.
	Geração de cenários para o plano de emergência sob avaliação: O time de revisão pode gerar cenários que serão utilizados nos planos de emergência. Estes cenários podem ser previamente armazenados em um banco de cenários e serem recuperados de acordo com o plano de emergência no escopo.
	Simulação de diferentes cenários sobre plano de emergência em avaliação: O time de revisão pode recuperar um conjunto de cenários e executá-los contra o plano de emergência. Através dessa ação, é possível analisar o comportamento do plano de emergência, verificando se ele parece estar correto.
Preparação Individual	Geração de planos mutantes: Os revisores podem remover partes do plano de emergência sob avaliação, gerando um conjunto de planos mutantes. Estes planos mutantes devem ser compartilhados entre aqueles que estão envolvidos com a avaliação do plano de emergência, com o objetivo de evitar duplicidade dos planos mutantes.
	Apoio à revisão de planos mutantes: A ferramenta deve apoiar as atividades do revisor e do time de revisão, incluindo a análise dos planos mutantes e registro do status deles (morto, requer avaliação durante reunião de revisão, equivalente ao plano original, requer novos cenários etc.).
Reunião de revisão	Registro dos problemas do plano de emergência: A ferramenta precisa registrar os problemas identificados no plano de emergência durante o processo de avaliação e permitir a recuperação pelos autores durante os ajustes no plano de emergência.

Retrabalho	Monitoramento da evolução do plano de emergência: A ferramenta pode rastrear a evolução do plano de emergência, permitindo o registro de mudanças feitas no plano de emergência original (histórico de mudanças) e o registro de como e se os problemas identificados no plano de emergência foram resolvidos.
Verificação	Geração de estatísticas: A ferramenta deve apoiar a geração do recorde de mutação, que pode ser utilizado como uma métrica para avaliar o nível de qualidade do plano de emergência e consequentemente a confiança nele.
Todas	Notificação de participantes: A ferramenta deve apoiar o envio de notificações aos que estão envolvidos no processo de avaliação do plano de emergência. Essas pessoas devem ser informadas quando um plano de emergência precisa ser avaliado, quando planos mutantes podem ser gerados, quando um plano de emergência precisa ser modificado, etc.

4 Desenvolvimento da Aplicação

Neste capítulo, serão descritas a arquitetura e as tecnologias utilizadas no desenvolvimento do aplicativo proposto.

4.1 Arquitetura do sistema

O sistema proposto adota uma arquitetura clássica cliente-servidor com 3 camadas: cliente, aplicação servidora e persistência.

Para o cliente, a tecnologia adotada foi o React, uma biblioteca JavaScript para desenvolvimento de interfaces web, criada e mantida pelo Facebook.

Para a aplicação servidora, foi utilizado o padrão REST (Representational State Transfer - Transferência Representacional de Estado), sendo adotado o Node.js para o seu desenvolvimento.

E, para a terceira camada, foi adotado para a persistência de dados SGBD (Sistema de Gerenciamento de Banco de Dados) MongoDB, sendo hospedado no MongoDB Atlas, um serviço para bancos de dados MongoDB, que está descrito na Seção 4.2.3.

Para unificação das camadas cliente e aplicação servidora, foi utilizado também o Lerna, um gerenciador de pacotes, que está descrito na Seção 4.2.4.

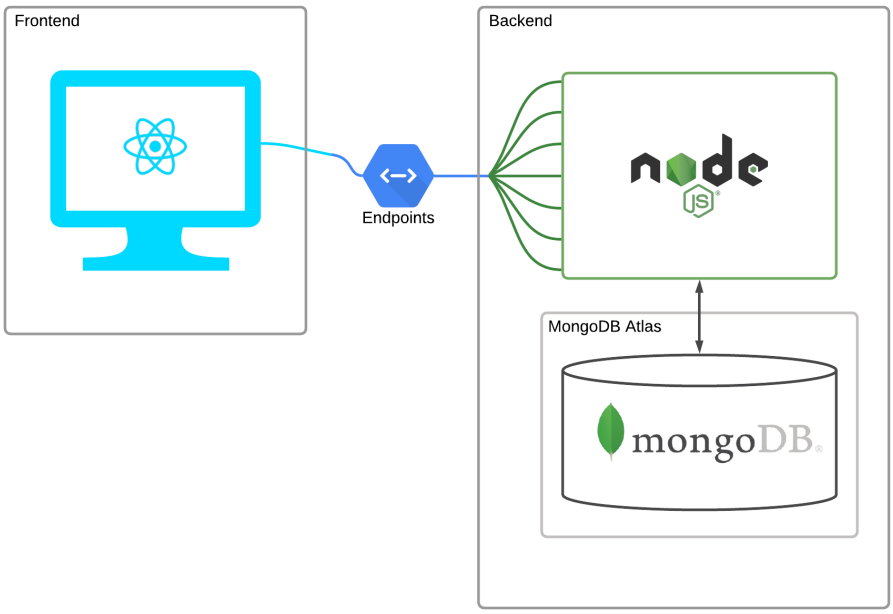


Figura 8: Tecnologias do sistema.

4.2 Tecnologias Adotadas

4.2.1 React

React (também denominado React.js ou ReactJS) é uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces visuais, comumente usado em páginas web. É mantida e distribuída em regime de código aberto pelo Facebook, Instagram, outras empresas e por uma grande comunidade de desenvolvedores individuais (REACT, 2019).

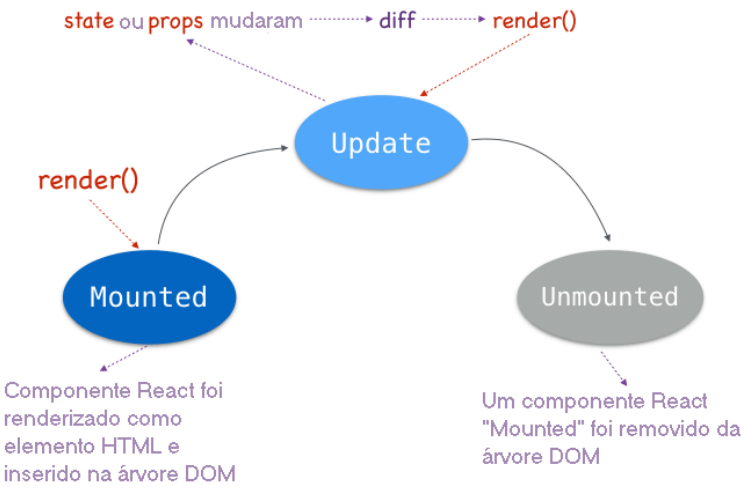


Figura 9: Ciclo de vida dos componentes React.

O React tem uma pretensão de curva de aprendizado bastante suave quando comparada com os seus concorrentes. O que o separa das demais bibliotecas que manipulam o DOM (Document Object Model - Modelo de Documentos de Objetos) são os seus componentes, unidades individuais reutilizáveis, e o seu ciclo de vida, presente na Figura 9.

O React manipula o DOM, como pode ser visto na Figura 10, e tem uma alta performance, pois, para cada objeto DOM, existe um "objeto DOM virtual" correspondente. Um objeto DOM virtual é a representação de um objeto DOM, como uma cópia em memória. Um objeto DOM virtual tem as mesmas propriedades do DOM real, mas diferenças quando algo é alterado e sua representação na tela. Manipular DOM é lento. Manipular o DOM virtual é muito mais rápido, pois não modifica nada previamente desenhado na tela (GOGÓ, 2017).

```
IgorMCesar, 12 days ago | 1 author (IgorMCesar)
1  import React from 'react';  8.4K (gzipped: 3.4K)
2  import ReactDOM from 'react-dom';
3  import { BrowserRouter as Router } from 'react-router-dom';  26.5K (gzipped: 8.9K)
4
5  import App from './App';
6
   Complexity is 4
7  const Root = () => {
8    return (
9      <Router>
10       <App />
11     </Router>
12   );
13 };
14
15 const rootElement = document.getElementById('root');
16 ReactDOM.render(<Root />, rootElement);
17
```

Figura 10: Exemplo de arquivo raiz do React e a sua manipulação do DOM.

4.2.2 Node.js

JavaScript é comumente usado no desenvolvimento client-side, ou seja, aplicações que rodam no computador do cliente, em navegadores, sendo usado para manipular o DOM (Document Object Model - Modelo de Documentos de Objetos) por exemplo. O DOM não é, por entanto, tipicamente provido pelo motor JavaScript, mas sim pelo navegador. O mesmo é verdade para a engine V8, O Google Chrome provê o DOM. A engine V8, no entanto, provê todos os tipos de dados, operadores, objetos e funções especificados no padrão ECMA (ECMA, 2019).

A engine V8 possibilita que qualquer aplicação C++ exponha seus próprios objetos e funções para códigos JavaScript. Desta forma, o Node.js consegue usar uma arquitetura conhecida como “single thread event loop model” (modelo de loop de eventos utilizando apenas uma thread), onde todas as requisições são interpretadas de forma assíncrona em uma única thread. Essa abordagem torna o Node.Js extremamente rápido ao lidar com um número alto de requisições (NODE, 2019).

Node.js é um ambiente de execução de Javascript assíncrono orientado a eventos, projetado para o desenvolvimento de aplicações escaláveis de rede com alto desempenho, construída sobre a engine V8 (NODE, 2019). A V8, desenvolvida pela Google e mantida e distribuída em regime de código aberto, é utilizada como interpretador JavaScript no Chrome, seu navegador (V8, 2019).

As aplicações desenvolvidas em Node.js podem ser constituídas por três tipos de módulos: os módulos presentes, por padrão, no Node.js; os módulos de terceiros que são adquiridos utilizando o NPM (Node Package Manager - Gerenciador de Pacotes Node), a Figura 11 mostra um exemplo de módulos utilizados na construção do servidor Node.js da aplicação; e os módulos proprietários desenvolvidos pelos desenvolvedores da própria aplicação (NPM, 2019).

Figura 11: Arquivo package.js do servidor Node.js mostrando os módulos de terceiros usados na construção da aplicação.

O módulo mais importante da aplicação desenvolvida é o Express, adquirido utilizando o NPM, citado acima. O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos. É possível observar alguns destes recursos na Figura 12, para aplicativos web e móvel (EXPRESS, 2019).

```
IgorMCesar, 10 days ago | 1 author (IgorMCesar)
import express from 'express';
import logger from 'morgan';  41.1K (gzipped: 13.2K)
import cors from 'cors';  5.2K (gzipped: 2.1K)
import helmet from 'helmet';  47.1K (gzipped: 11.4K)

import routes from './routes/routes';

IgorMCesar, 10 days ago | 1 author (IgorMCesar)
class AppController {
  constructor() {
    this.express = express();

    this.middlewares();
    this.routes();
  }

  middlewares() {
    if (this.express.get('env') === 'development') {
      this.express.use(logger('dev'));
    }
    this.express.use(express.json());
    this.express.use(cors());

    this.express.use(helmet());
    this.express.use(helmet.permittedCrossDomainPolicies());
  }

  routes() {
    this.express.use(routes);
  }
}

export default new AppController().express;
```

Figura 12: Arquivo app.js do servidor Node.js mostrando a instanciação e utilização do Express.js e alguns de seus recursos.

O Express possui uma miríade de métodos utilitários HTTP e middlewares, o que possibilita a criação de uma API (Application Programming Interface - Interface de Programação de Aplicações) e seus respectivos endpoints (rotas), como mostrado na Figura 13.

```
1  import express from 'express';
2
3  import emergencyPlan from './emergencyPlan/emergencyPlan';
4  import mutantPlan from './mutantPlan/mutantPlan';
5  import scenario from './scenario/scenario';
6  import resource from './resource/resource';
7  import procedure from './procedure/procedure';
8  import step from './step/step';
9
10 const routes = express();
11
12 // Check service health
13 routes.get('/health-check', (req, res) => res.send('OK'));
14
15 // Define Routes
16 routes.use('/emergency-plan', emergencyPlan);
17 routes.use('/mutant-plan', mutantPlan);
18 routes.use('/scenario', scenario);
19 routes.use('/resource', resource);
20 routes.use('/procedure', procedure);
21 routes.use('/step', step);
22
23 export default routes;
24
```

Figura 13: Arquivo app.js do servidor Node.js mostrando a instanciação e utilização do Express.js e alguns de seus recursos.

4.2.3 MongoDB

O MongoDB é um software de banco de dados NoSQL (Not only SQL), orientado a documentos livre, de código aberto e multiplataforma. Diferentemente dos bancos de dados relacionais, relaxa os princípios ACID (atomicidade, consistência, isolamento e durabilidade) em favor de flexibilidade, escalabilidade, redução da complexidade e performance. Suas principais características permitem com que as aplicações modelem informações de modo muito mais natural, sendo mais flexível em relação aos *schemas* e possuindo identificadores únicos universais (MONGODB, 2019).

Por ser um SGDB orientado a documentos, a aplicação fica responsável por implementar as coleções e suas respectivas propriedades. Desta forma, seus *schemas* se tornam flexíveis em comparação com os bancos relacionais, visto que seus atributos são, propositalmente, pouco delimitados. Por ser bem genérico, o MongoDB oferece uma variedade de opções extremamente úteis, como operadores similares aos operadores SQL (Structured Query Language), TTL (Time to Live), MapReduce e indexação secundária.

JSON (Javascript Object Notation) é uma forma de representar objetos derivados do JavaScript, sendo um formato amplamente utilizado nos ambientes web. Os documentos utilizados pelo MongoDB são BSON (Binary JSON), um formato de dados

binários que é similar ao JSON, mas inclui dados adicionais. Um exemplo desse tipo de documento pode ser visto na Figura 14 (MONGODB, 2019).

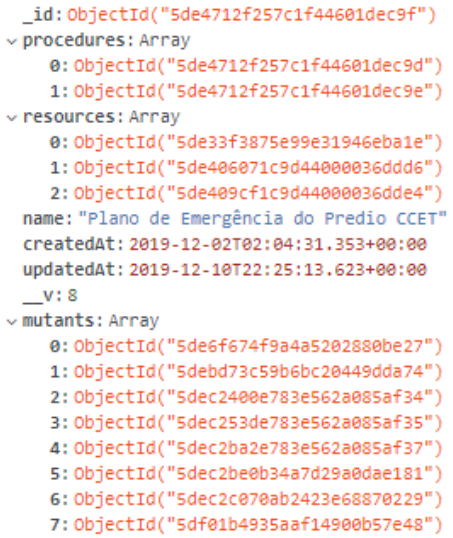


Figura 14: Documento BSON de um Plano de Emergência criado no MongoDB Atlas a partir do uso da aplicação.

Para auxiliar o uso do MongoDB na aplicação feita em NodeJS, foi utilizado a biblioteca Mongoose, que oferece uma solução simples e robusta, baseada em *schemas* para modelar os dados. Isso facilita as operações básicas em um banco de dados, como consulta, escrita e atualização de documento. Um exemplo pode ser visto na Figura 15.

Figura 15: Arquivo emergencyPlanController.js do servidor Node.js mostrando o controlador dos Planos de Emergência e algumas operações no banco MongoDB utilizando o Mongoose.

O Mongoose também inclui conversão de tipos incorporada, validação, criação de queries, ganchos de lógica de negócios e muito mais, prontos para uso (MONGOOSE, 2019). O Mongoose também possibilita criar modelos de dados similares aos de um banco de dados relacional, onde é possível especificar o tipo dos dados do *schema*. As consultas também utilizam métodos mais simples e robustos. Além disso, se conecta facilmente com o banco MongoDB diretamente a partir de um único método, como é possível ver na Figura 16.

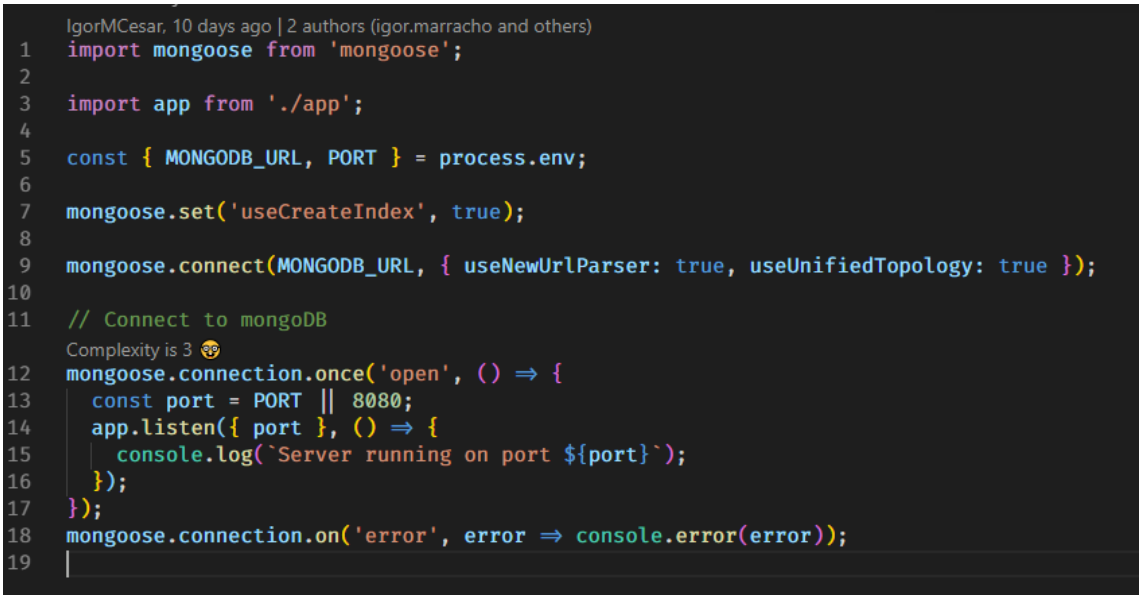


Figura 16: Arquivo index.js do servidor Node.js mostrando a instanciação Mongoose e alguns métodos, entre eles o de conexão direta com o MongoDB.

O serviço usado no projeto é o MongoDB Atlas, um serviço de banco de dados global na nuvem que permite usuários cadastrados utilizarem, de forma gratuita, um espaço de armazenamento limitado (MONGODB ATLAS, 2019). A Figura 17 ilustra a página de configuração da conexão com o MongoDB, em que é disponibilizada a URL de acesso ao banco de dados.

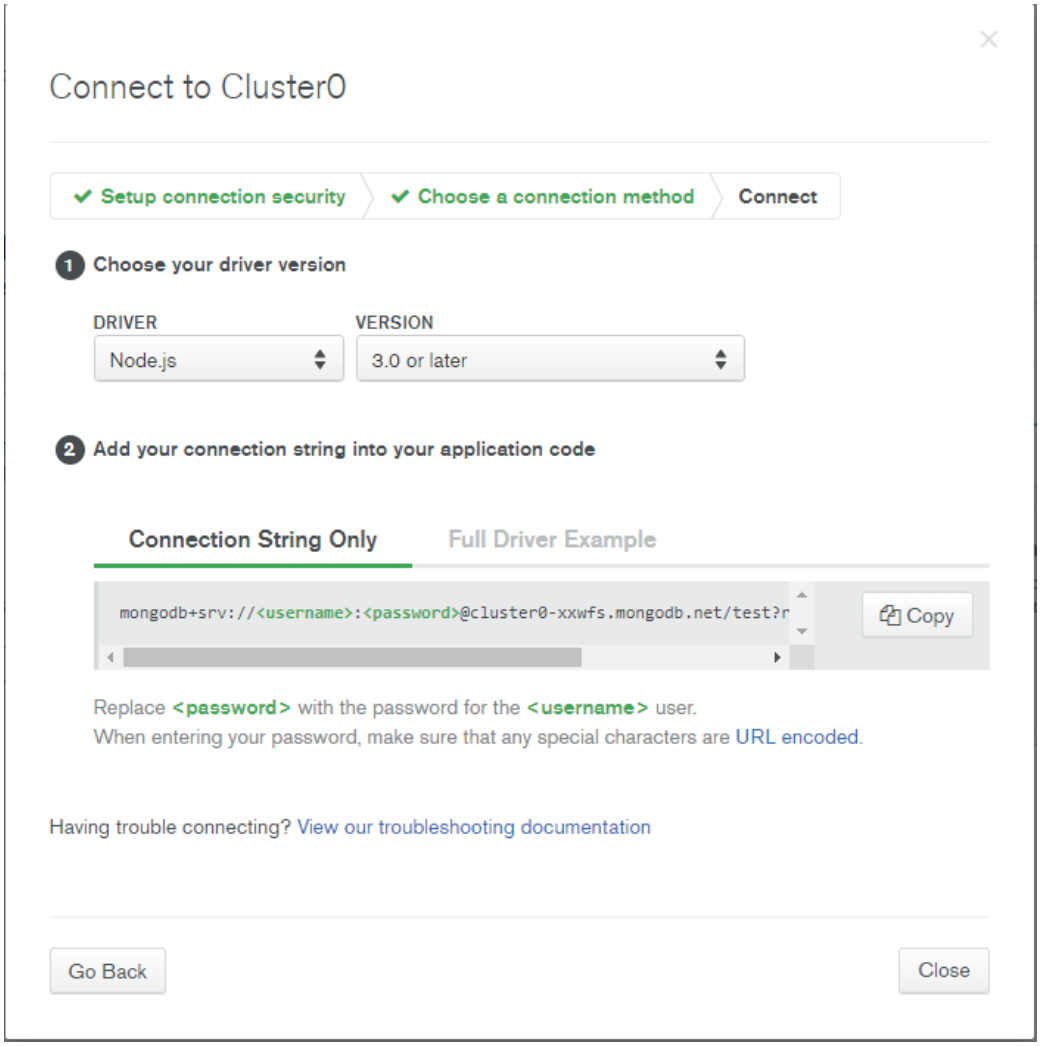
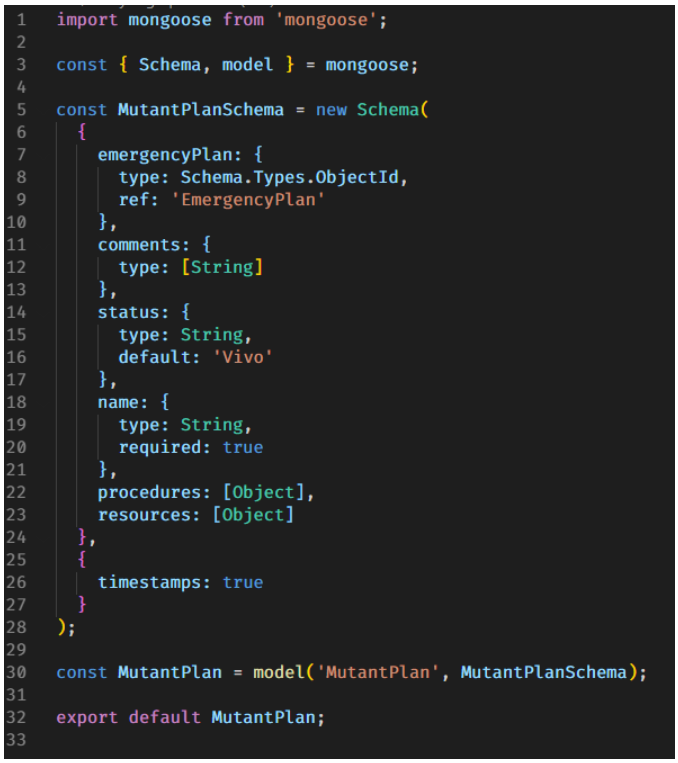


Figura 17: Os dados necessários para realizar a conexão com banco de dados hospedado no MongoDB Atlas.

O banco de dados deste projeto foi desenvolvido apoiado em nos seguintes *schemas*: Plano de Emergência, Plano Mutante, Procedimento, Recurso, Cenário e Passo. Todos os schemas e suas características podem ser vistos da Figura 18 à Figura 23.

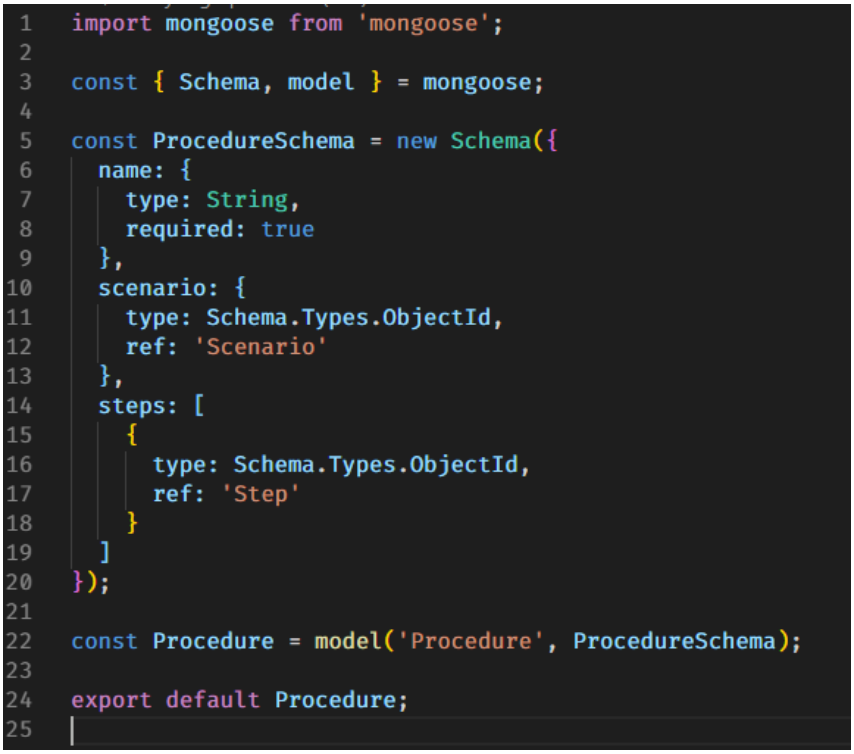
```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const EmergencyPlanSchema = new Schema(
  {
    name: {
      type: String,
      required: true
    },
    procedures: [
      {
        type: Schema.Types.ObjectId,
        ref: 'Procedure'
      }
    ],
    resources: [
      {
        type: Schema.Types.ObjectId,
        ref: 'Resource'
      }
    ],
    mutants: [
      {
        type: Schema.Types.ObjectId,
        ref: 'MutantPlan'
      }
    ]
  },
  {
    timestamps: true
  }
);

const EmergencyPlan = model('EmergencyPlan', EmergencyPlanSchema);

export default EmergencyPlan;
```

Figura 18: Arquivo do modelo de Plano de Emergência do servidor Node.js mostrando o schema e os campos relacionados.



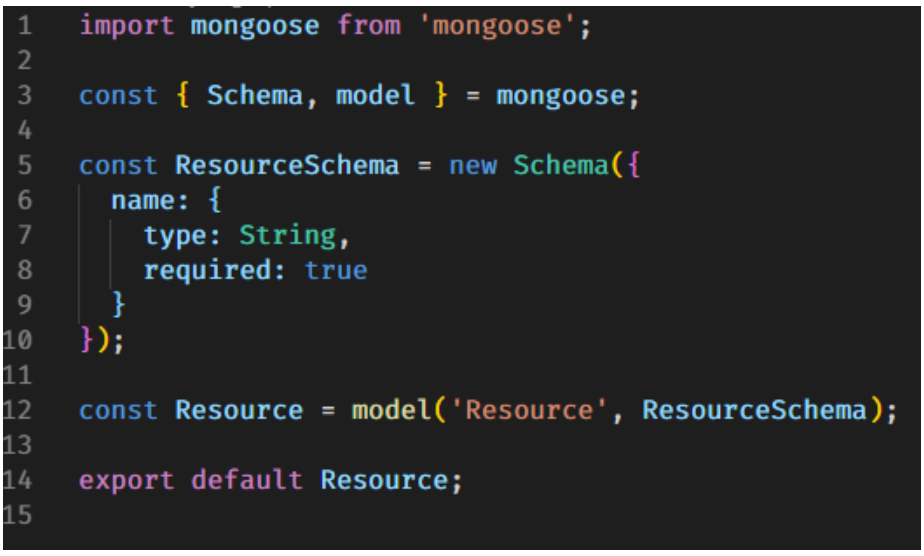
```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const MutantPlanSchema = new Schema(
  {
    emergencyPlan: {
      type: Schema.Types.ObjectId,
      ref: 'EmergencyPlan'
    },
    comments: {
      type: [String]
    },
    status: {
      type: String,
      default: 'Vivo'
    },
    name: {
      type: String,
      required: true
    },
    procedures: [Object],
    resources: [Object]
  },
  {
    timestamps: true
  }
);

const MutantPlan = model('MutantPlan', MutantPlanSchema);

export default MutantPlan;
```

Figura 19: Arquivo do modelo de Plano Mutante do servidor Node.js mostrando o schema e os campos relacionados.



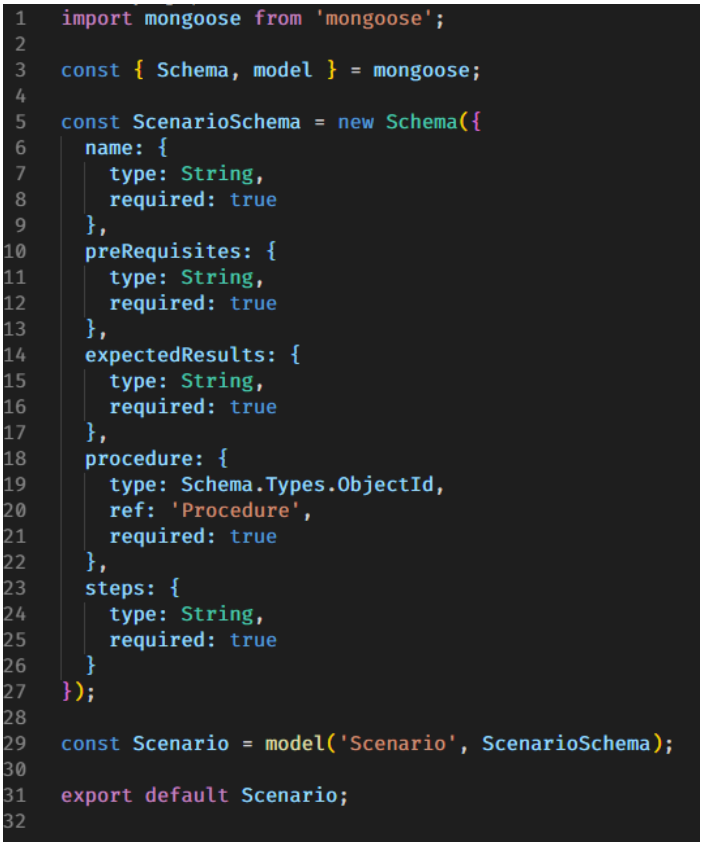
```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const ProcedureSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  scenario: {
    type: Schema.Types.ObjectId,
    ref: 'Scenario'
  },
  steps: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Step'
    }
  ]
});

const Procedure = model('Procedure', ProcedureSchema);

export default Procedure;
```

Figura 20: Arquivo do modelo de Procedimento do servidor Node.js mostrando o schema e os campos relacionados.



```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const ResourceSchema = new Schema({
  name: {
    type: String,
    required: true
  }
});

const Resource = model('Resource', ResourceSchema);

export default Resource;
```

Figura 21: Arquivo do modelo de Recurso do servidor Node.js mostrando o schema e os campos relacionados.



```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const ScenarioSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  preRequisites: {
    type: String,
    required: true
  },
  expectedResults: {
    type: String,
    required: true
  },
  procedure: {
    type: Schema.Types.ObjectId,
    ref: 'Procedure',
    required: true
  },
  steps: {
    type: String,
    required: true
  }
});

const Scenario = model('Scenario', ScenarioSchema);

export default Scenario;

```

Figura 22: Arquivo do modelo de Cenário do servidor Node.js mostrando o schema e os campos relacionados.

```
import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const StepSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  resources: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Resource'
    }
  ]
});

const Step = model('Step', StepSchema);

export default Step;

```

Figura 23: Arquivo do modelo de Passo do servidor Node.js mostrando o schema e os campos relacionados.

4.2.4 Lerna

Lerna é uma ferramenta para gerenciar projetos JavaScript com vários pacotes. A divisão de grandes bases de código em pacotes separados com versões independentes é extremamente útil para o compartilhamento de código. No entanto, fazer alterações em muitos repositórios é confuso e difícil de rastrear, e o teste entre repositórios se torna complicado muito rapidamente. Para resolver esses (e muitos outros) problemas, alguns projetos são organizados em repositórios com vários pacotes com suas bases de código. Projetos como Babel, React, Angular, Ember, Meteor, Jest e muitos outros desenvolvem todos os seus pacotes em um único repositório. O Lerna é uma ferramenta que otimiza o fluxo de trabalho em torno do gerenciamento de repositórios de vários pacotes com git e npm (LERNA, 2019).

O Lerna foi escolhido pois possibilita uma abordagem de repositório monorepo, um repositório que contém mais de um projeto lógico e estes são, provavelmente, fracamente conectados entre si ou podem se relacionar por outros meios, como um servidor e um cliente. (ATLASSIAN, 2019)

O projeto desenvolvido, como foi apresentado a seção 4.1, está dividido em dois subsistemas distintos que se complementam, um servidor e um cliente ambos na linguagem JavaScript. Para tal, seria necessário dividir as bases de código em dois repositórios, porém o gerenciamento e exposição do código se tornariam bem mais complexos com esta abordagem. Desta forma, o Lerna foi utilizado com o intuito de gerenciar um projeto monorepo composto de cliente React e servidor NodeJS. A estrutura de arquivos do repositório e a configuração do Lerna podem ser vistos na Figura 24.

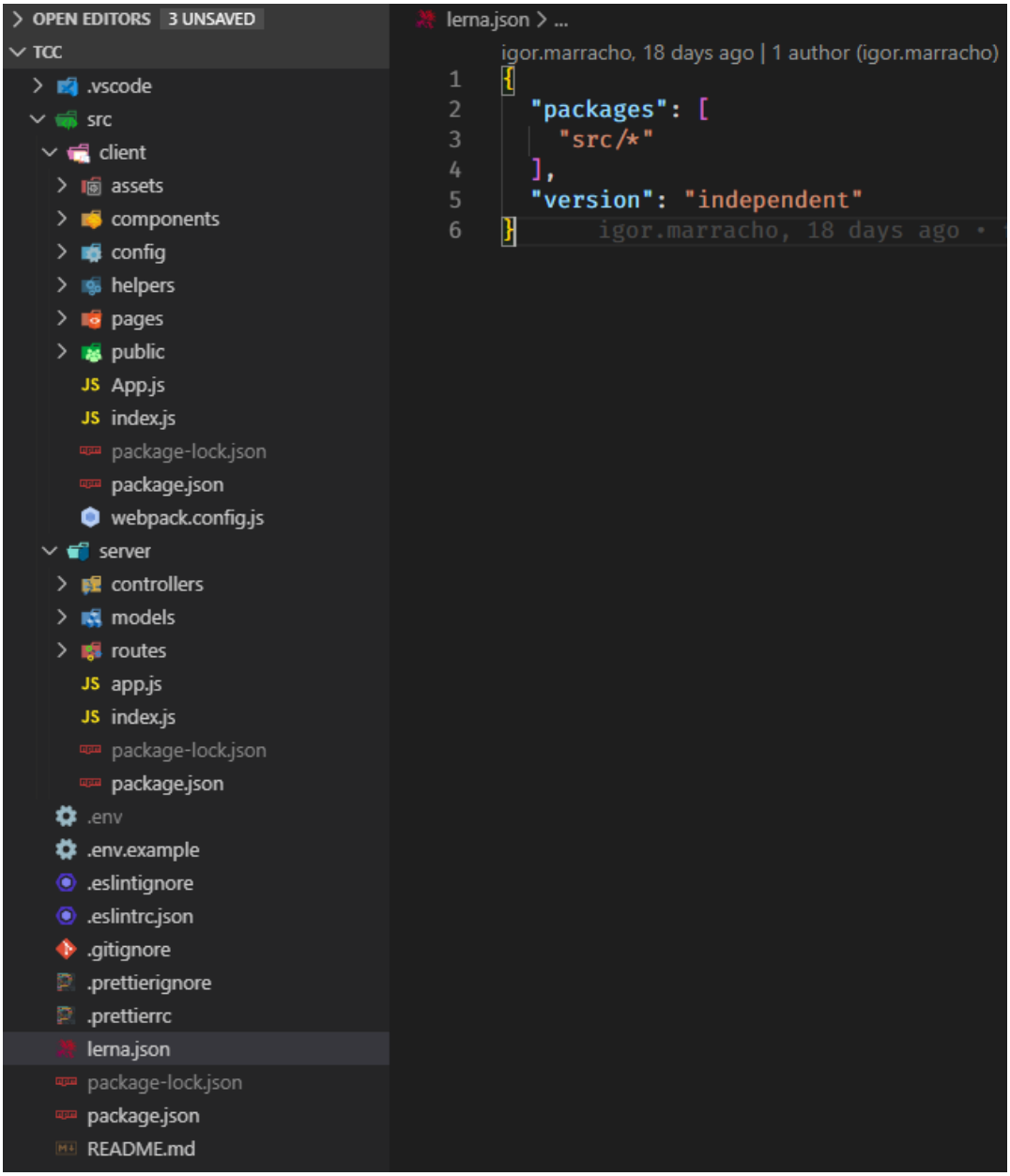


Figura 24: Estrutura de arquivos no repositório monorepo, explicitando servidor e cliente, e as configurações básicas do Lerna.

5 Implementação

Neste capítulo o funcionamento do aplicativo web desenvolvido será apresentado. Suas diversas telas e funcionalidades serão denotadas por meio de figuras, que simulam a navegação, acompanhadas por explicações textuais sobre elas.

5.1 Sistema Proposto

O sistema proposto é um aplicativo web que serve de apoio à decisão na avaliação de um plano de emergência. Ele auxilia os dois primeiros passos da abordagem de avaliação de planos de emergência utilizando FTR, sendo estas planejamento e preparação individual, que podem ser vistas na Figura 25.

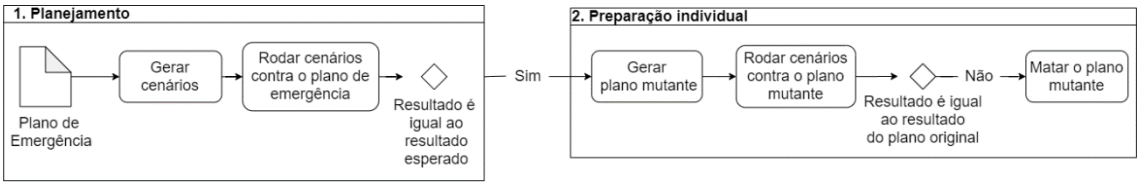


Figura 25: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação.

Nesse sistema é possível a criação de cada componente de um plano de emergência separadamente, de forma a reforçar o reuso de informações semelhantes. Os componentes do plano de emergência foram separados em recurso, passo, procedimento, cenário e o próprio plano de emergência. Além de ser possível visualizar todos os planos de emergência e seus respectivos componentes e, por fim, a criação e visualização de planos mutantes para que sejam avaliados e assim auxiliando o time de revisão.

5.2 Navegação do Aplicativo Web

O layout escolhido para o site faz uso de um menu lateral que pode ser recolhido, assim proporcionando mais espaço para o conteúdo principal. Como apresentado na Figura 26, as páginas do site foram agrupadas em 3 seções:

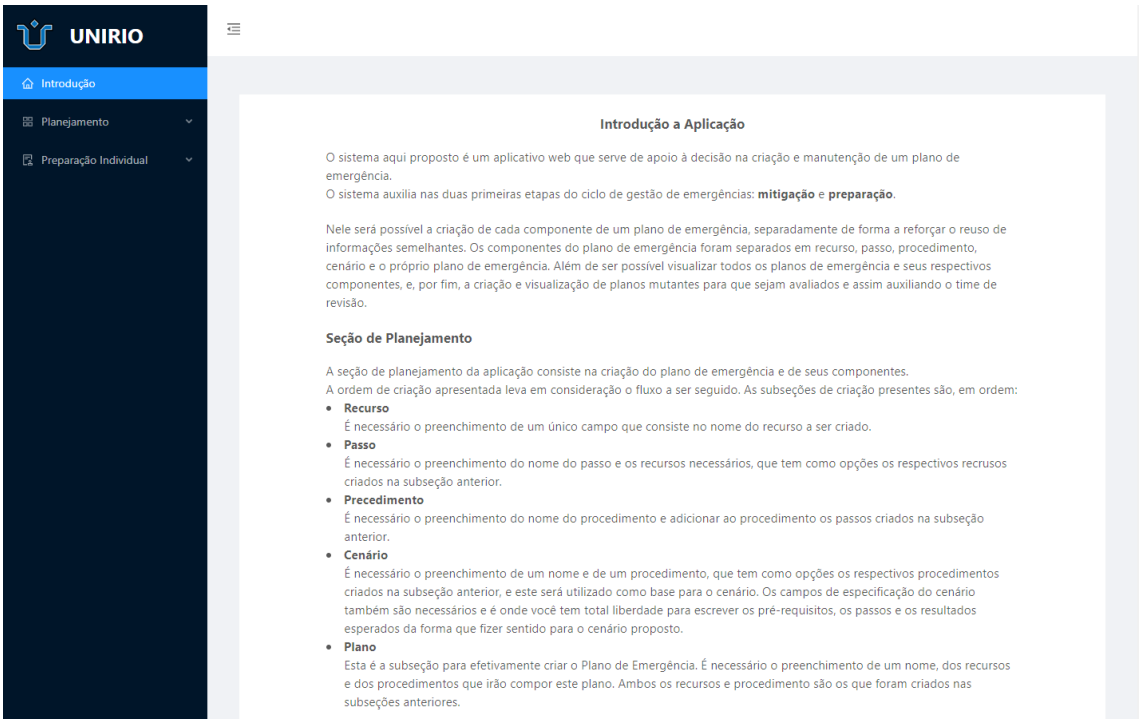


Figura 26: Página de introdução da aplicação com menu expandido.

1. *Seção de introdução*. É nessa seção que estão as informações importantes relacionadas ao uso do aplicativo e uma pequena explicação dos componentes;

2. *Seção de planejamento*. É a seção onde ocorre a criação do plano de emergência e de seus componentes.

3. *Seção de preparação individual*. É a seção que apresenta os planos de emergências previamente criados, além de permitir gerar, visualizar e avaliar seus respectivos mutantes. É por meio destas informações que se torna possível avaliar os planos de emergência.

Na Figura 26 é apresentado também o menu em sua forma expandida. Já na Figura 27 é apresentado como o menu se comporta ao ser recolhido, exibindo as outras páginas em um menu flutuante que aparece sobre o conteúdo sem que seja necessário a expansão do menu original.

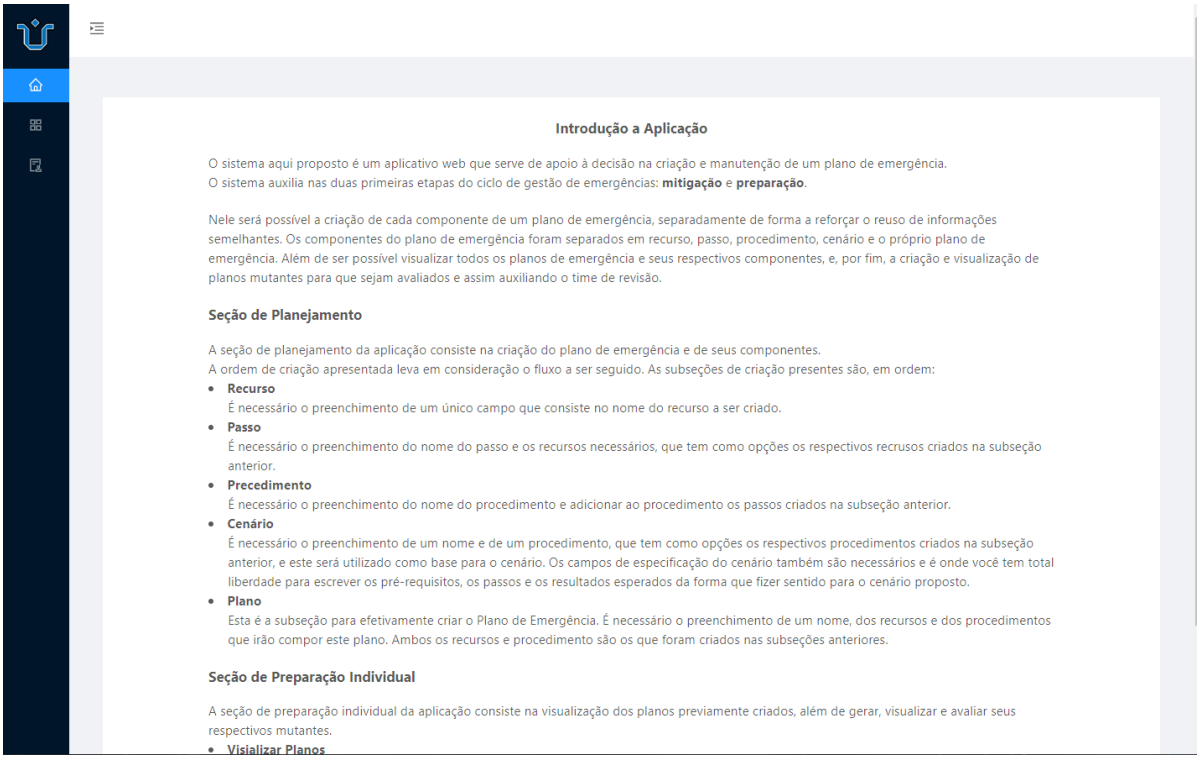


Figura 27: Página de introdução da aplicação com o menu reduzido.

5.2.1 Seção de Introdução

A seção de introdução é a página inicial da aplicação. Ou seja, todos os usuários que acessarem a aplicação, antes de verem qualquer outra informação, verão esta seção (Figuras 26 e 27). O propósito dela é disponibilizar uma pequena explicação do que se trata a aplicação e como usá-la.

5.2.2 Seção de Planejamento

A seção de planejamento da aplicação foi criada pensando no primeiro passo da abordagem de avaliação de planos de emergência utilizando FTR do ciclo de gerenciamento de emergência (Figura 28). Com isso, o principal foco dela é a criação e avaliação de planos de emergência. Desta forma, esta seção consiste na criação do plano de emergência e de seus componentes, ambos estão explicitados nas subseções a seguir.

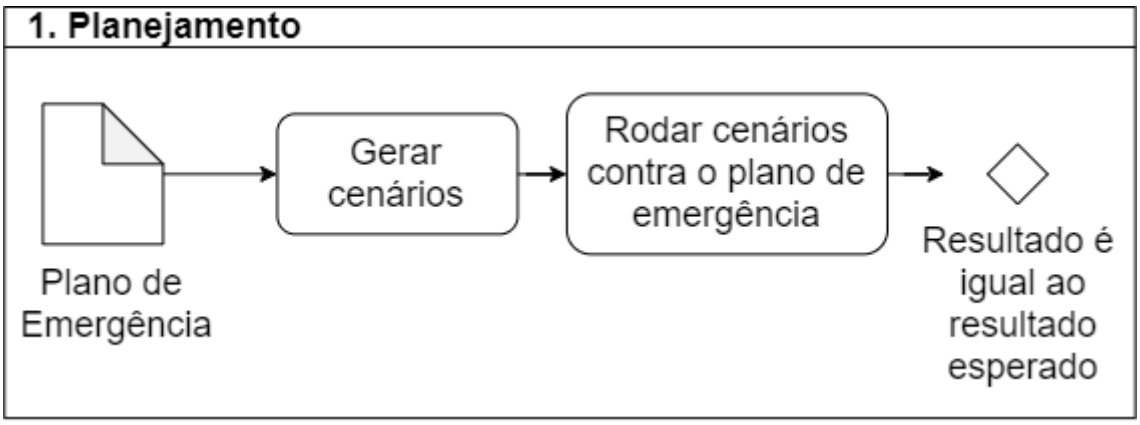


Figura 28: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação.

5.2.2.1 Página de Criação de Recurso

Na Figura 29 é apresentada a página de criação de recurso (i.e., elementos estáticos de um plano necessários para realização de uma ação durante a emergência), onde é necessário o preenchimento de um único campo, que consiste no nome do recurso a ser criado, e um botão, que deve ser clicado para efetivamente criar o recurso no banco de dados. Após a criação do recurso, ele estará disponível para ser utilizado nas outras etapas que utilizam recursos em sua criação.

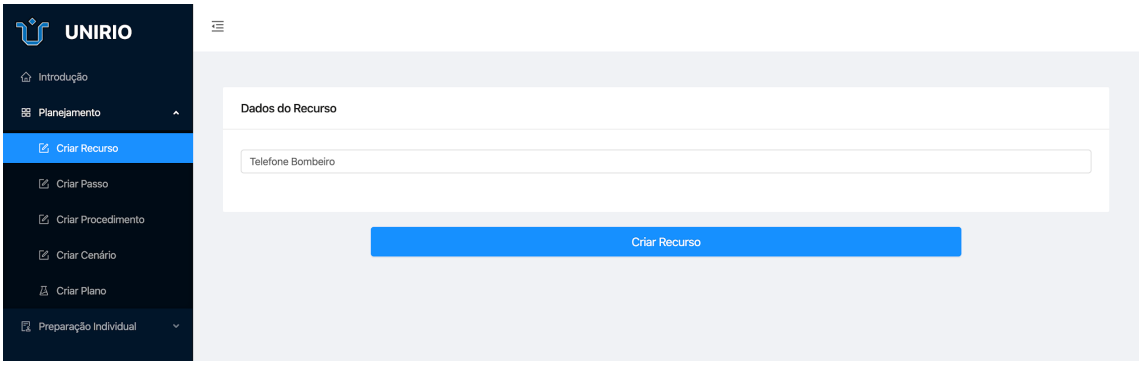


Figura 29: Página de criação de recursos.

5.2.2.2 Página de Criação de Passo

Na Figura 30 é apresentada a criação de um passo (i.e., uma ação a ser realizada), onde é necessário o preenchimento do nome do passo e os recursos necessários, que tem como opções os respectivos dados disponíveis no banco de dados para sua execução e um botão, que deve ser clicado para efetivamente criar o passo no banco de dados. Após a

criação do passo, ele estará disponível para ser utilizado nas outras etapas que adotam passos em sua criação.

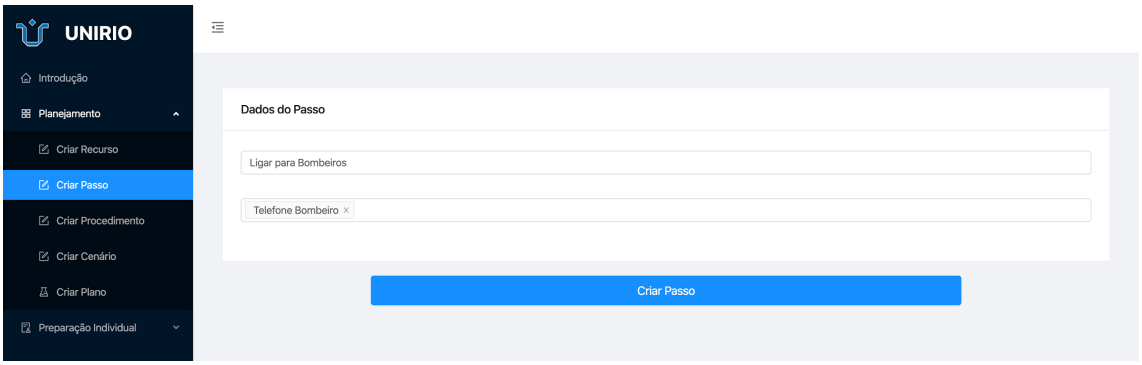


Figura 30: Página de criação de passos

.

5.2.2.3 Página de Criação de Procedimento

Na Figura 31 é apresentada a criação de um procedimento (i.e., um conjunto de passos a serem seguidos), onde é necessário o preenchimento do nome do procedimento e os passos previamente criados, exemplificado na Figura 30, e um botão que deve ser clicado para efetivamente criar o procedimento no banco de dados. Nesta página é possível adicionar e remover quantos passos forem necessários para o procedimento a ser criado. Após a criação do procedimento, ele estará disponível para ser utilizado nas outras etapas que adotam procedimentos em sua criação.

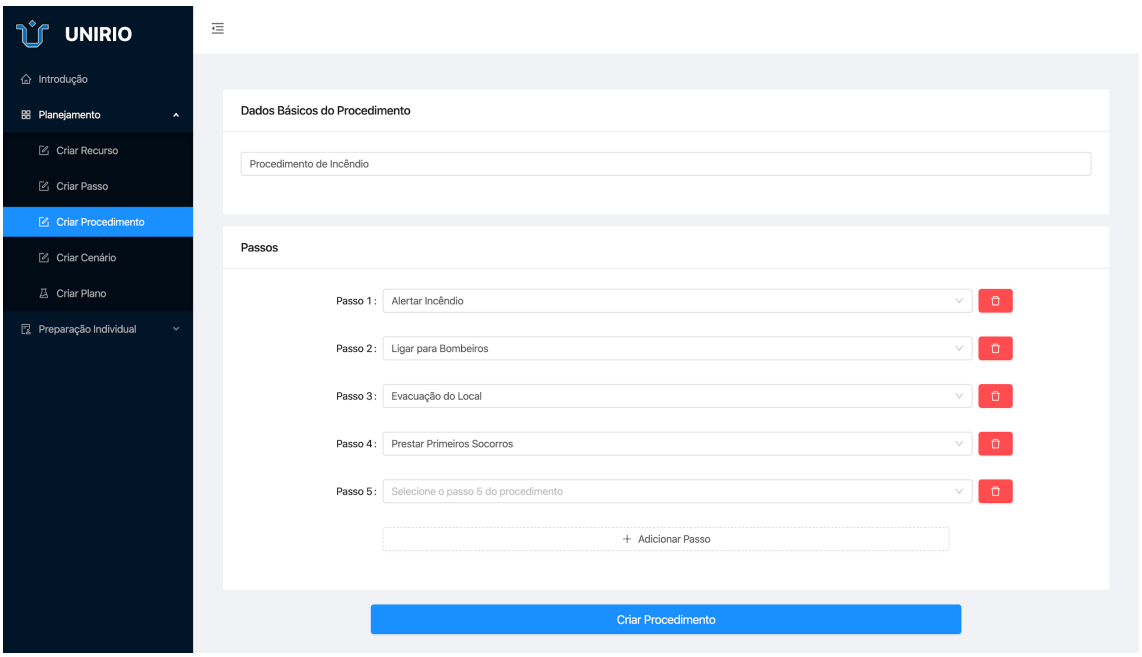


Figura 31: Página de criação de procedimento.

5.2.2.4 Página de Criação de Cenário

Na Figura 32 é apresentada a criação de um cenário[1], onde é necessário o preenchimento de um nome e de um procedimento, que tem como opções os respectivos dados disponíveis no banco de dados, e este é utilizado como base para o cenário. Os campos de especificação do cenário também são necessários e é onde o usuário tem total liberdade para escrever os pré-requisitos, os passos e os resultados esperados da forma que fizer sentido para o cenário proposto, além de um botão que deve ser clicado para efetivamente criar o cenário no banco de dados.

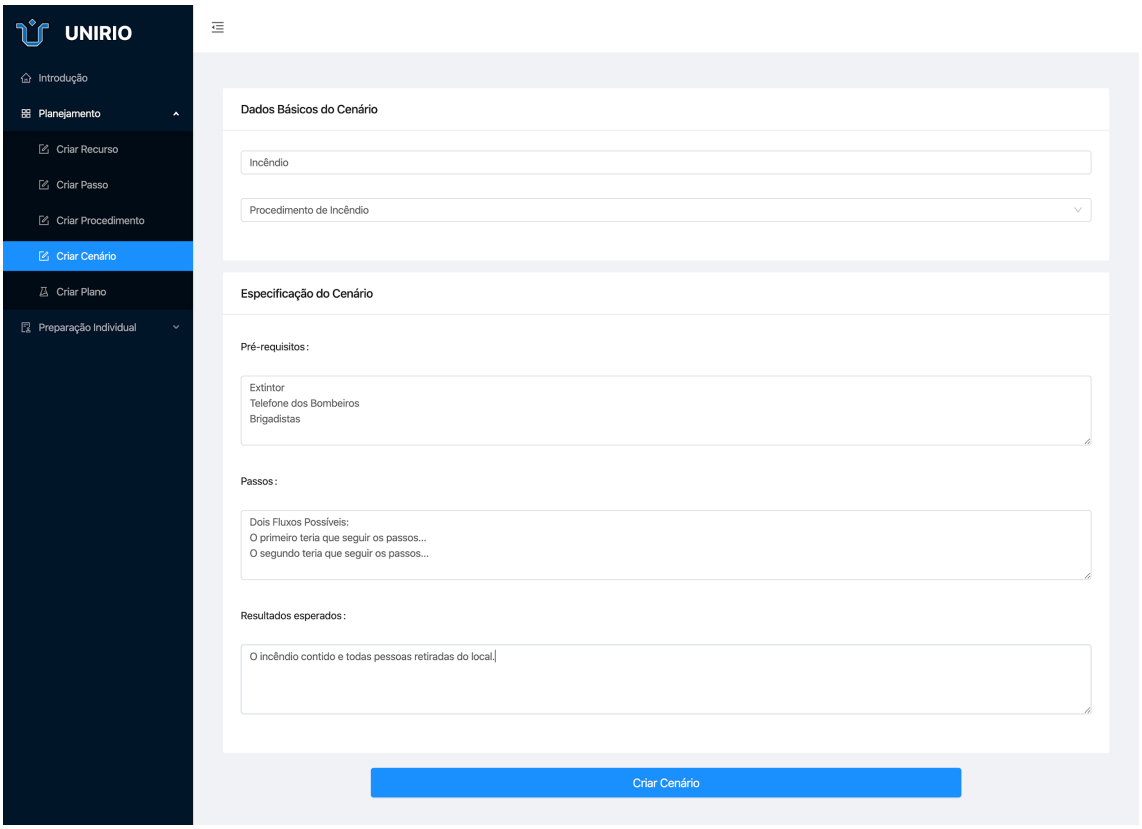


Figura 32: Página de criação de cenário.

5.2.2.5 Página de Criação de Plano de Emergência

Na Figura 33 é apresentada a criação de um plano de emergência, onde é necessário o preenchimento de um nome, dos recursos e dos procedimentos que irão compor este plano, além do botão que deve ser clicado para efetivamente criar o plano de

[1] Um procedimento que deve ser levado como base para um caso de teste. Por exemplo, um procedimento A pode ser utilizado para ser executado em um cenário de incêndio. Nesse caso, o procedimento A deve seguir os passos necessários definidos neste cenário.

emergência no banco de dados. O campo de preenchimento dos recursos e dos procedimentos tem como opções os respectivos dados disponíveis no banco de dados. O usuário pode adicionar ou remover livremente qualquer um desses atributos desde que eles tenham sido previamente adicionados nas etapas anteriores de criação.

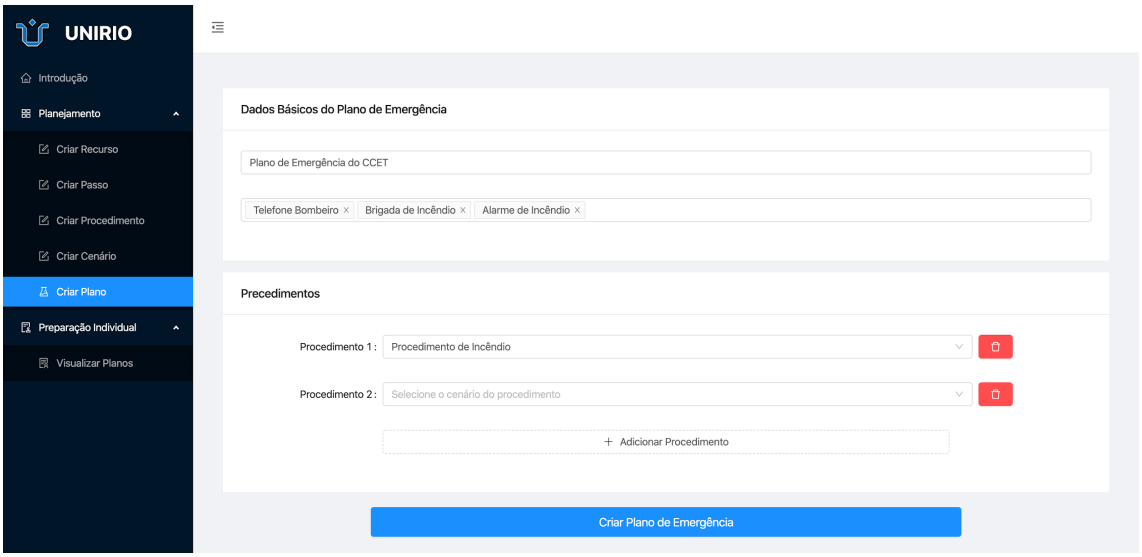


Figura 33: Página de criação de um plano de emergência.

5.2.3 Seção de Preparação Individual

A seção de preparação individual da aplicação foi criada pensando no primeiro passo da abordagem de avaliação de planos de emergência utilizando FTR do ciclo de gerenciamento de emergência. Com isso, o principal foco dessa seção é a geração e avaliação de mutantes originados dos planos de emergência. Desta forma, esta seção consiste na visualização dos planos previamente criados, além de gerar, visualizar e avaliar seus respectivos mutantes.

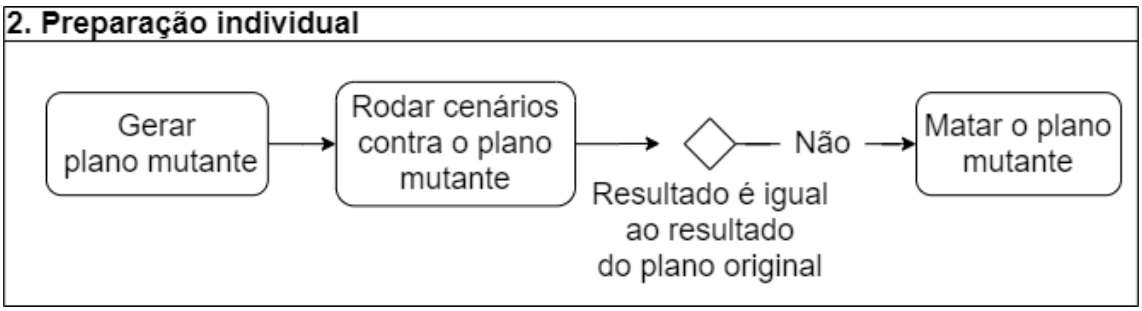


Figura 34: Processo de avaliação de planos de emergência utilizando as técnicas de teste de Engenharia de Software FTR e Teste de Mutação.

5.2.3.1 Página de Visualização de Planos de Emergência

Na Figura 35 é apresentada a listagem dos planos de emergência. Nesta página é possível a visualização de todos os planos já criados no sistema.

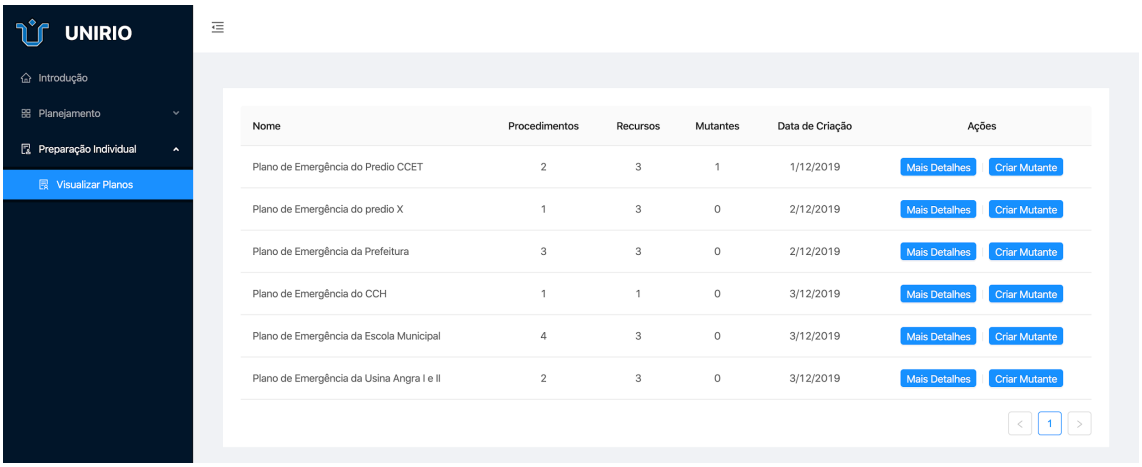


Figura 35: Página de visualização de planos.

Na página de visualização dos planos de emergência, ao selecionar a opção "Mais Detalhes" sobre um plano de emergência, uma tela é aberta exibindo os componentes e os mutantes do plano de emergência selecionado, como mostrado na Figura 36. Esta tela apresenta informações como nome, data de criação, os recursos presentes, os procedimentos, e os mutantes.

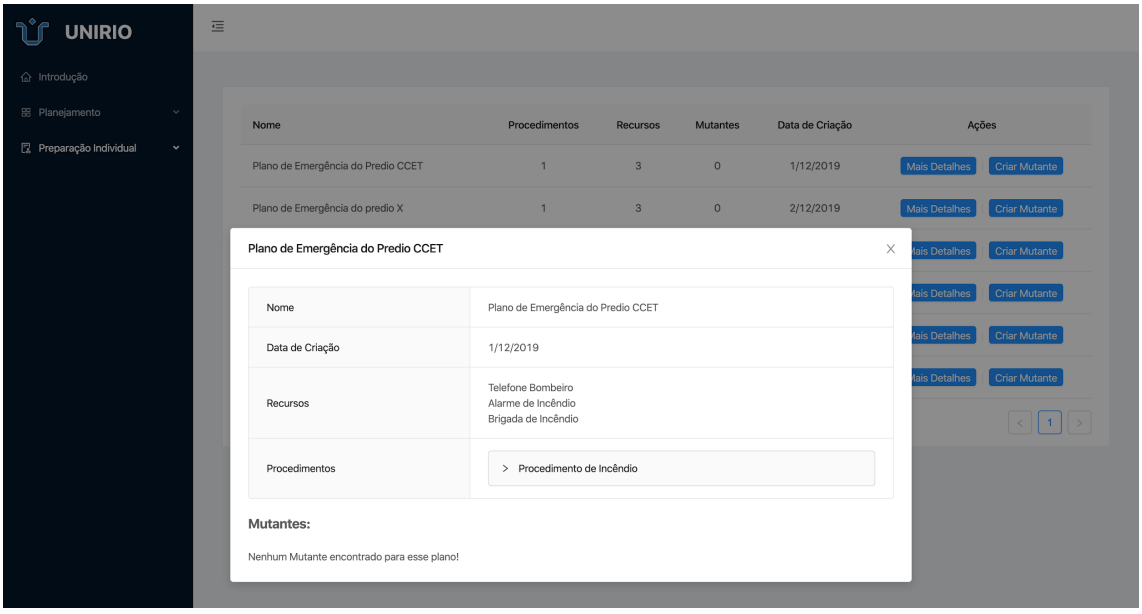


Figura 36: Modal de detalhes do plano de emergência.

Na página de visualização de planos de emergência, também existe a opção de "Criar Mutante". Ao selecioná-la, uma outra tela é aberta, onde é possível selecionar quais

recursos, procedimentos ou passos desses procedimentos devem ser removidos, como pode ser visto na Figura 37.

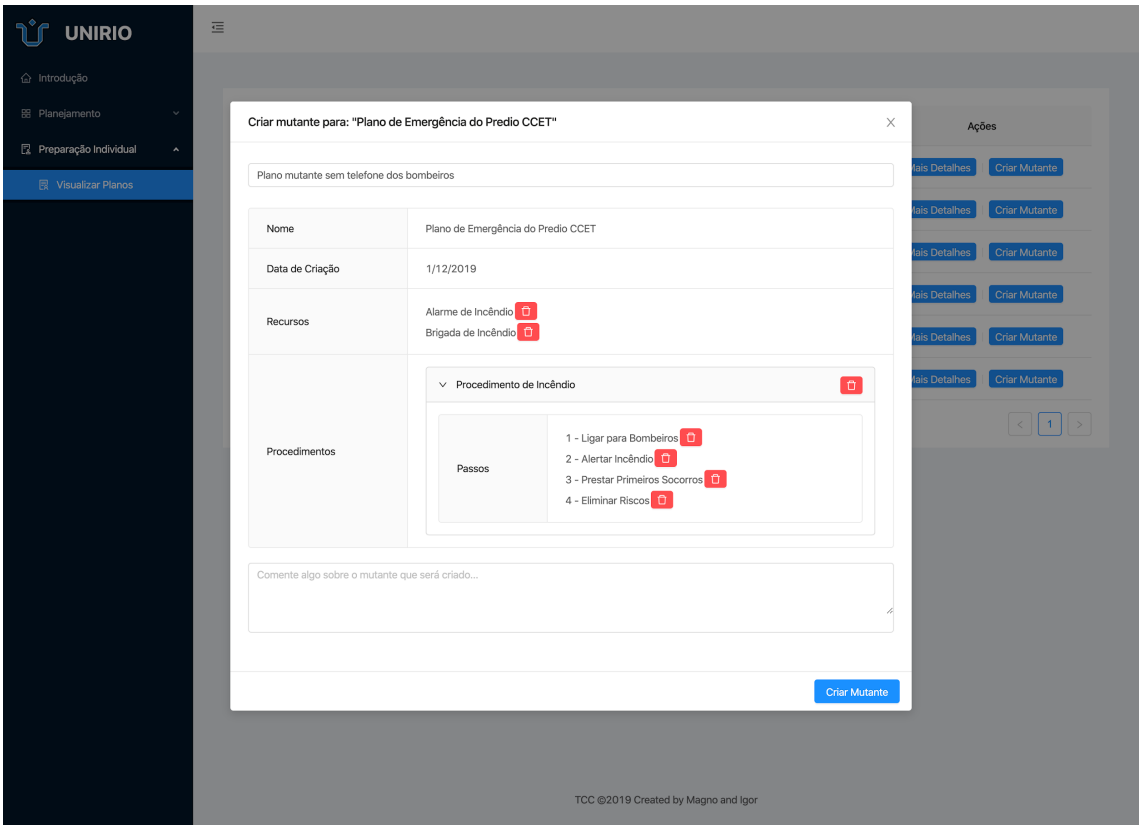


Figura 37: Modal de criação de mutantes para um plano de emergência.

Após a criação do plano mutante, o mesmo pode ser visualizado no modal de detalhes do plano de emergência, como mostrado na Figura 38, adicionando ainda mais informações relacionadas ao plano em questão. As informações do mutante são muito similares ao plano de emergência original, possuindo apenas as informações que não foram retiradas pelo usuário a partir da criação do mutante.

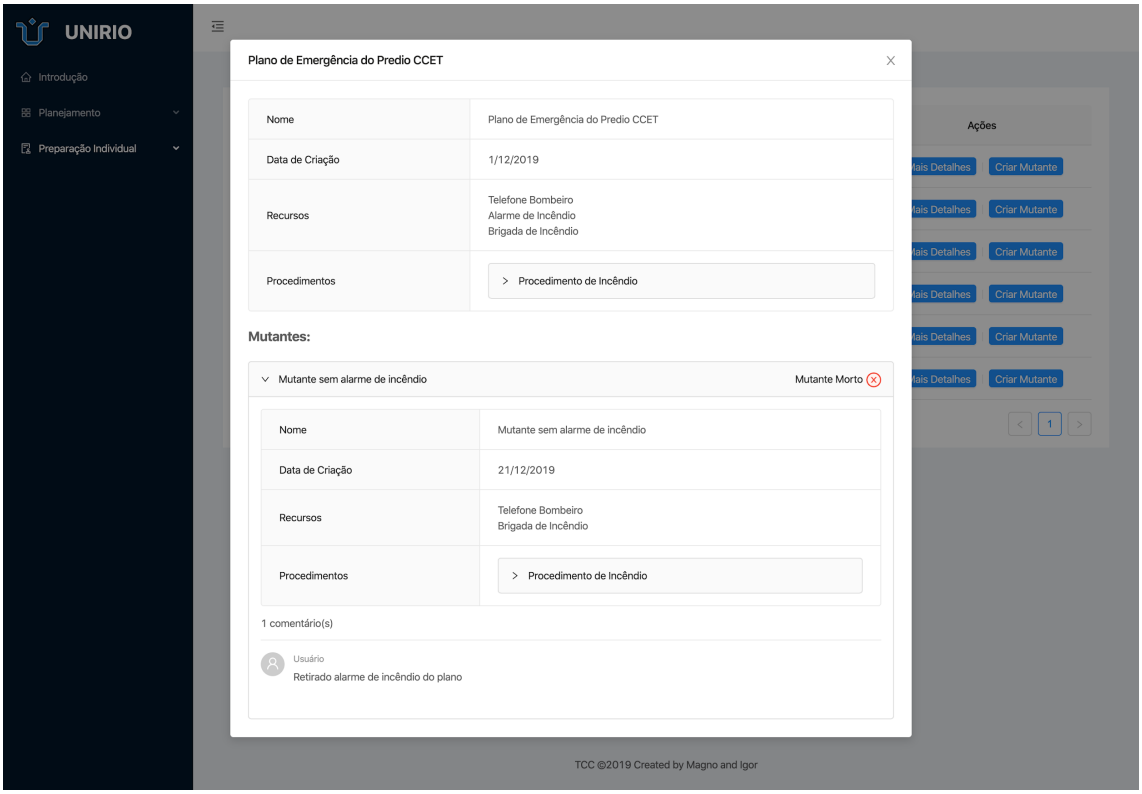


Figura 38: Modal de detalhes do plano de emergência com um mutante do plano.

Após a criação do mutante também é possível marcar, uma única vez, o estado do mutante, e, desta forma, identificar se o mutante é equivalente ou morto. Tal interação pode ser vista na Figura 39.

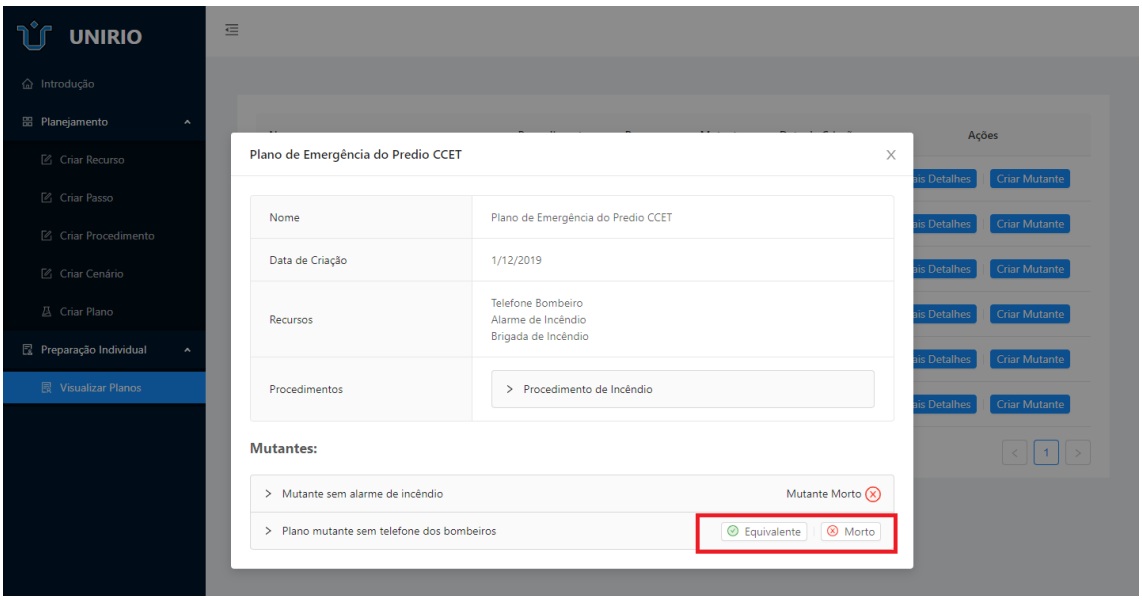


Figura 39: Botões para marcar o estado de um mutante como equivalente ou morto.

6 Conclusão

6.1 Considerações Finais

Foi observado ao longo deste trabalho que uma das principais formas de se avaliar planos antes que a emergência ocorra é por meio de simulação. Entretanto, apesar de seu potencial, ela pode ser muito cara e demandar a construção de ambientes e ferramentas complexas. Com isso, antes de enviar um plano de emergência para avaliação, todos os esforços devem ser tomados para reduzir erros básicos ou omissões no plano.

Com base nesse estudo sobre planos de emergência, foi desenvolvido um sistema web, composto de um cliente (Front-End), um servidor (Back-End) e um sistema de banco de dados não relacional (NoSQL) para servir de apoio na avaliação de planos de emergência. Para construir a aplicação, foram utilizados a biblioteca React, que permite a criação de aplicações front-end a partir de código JavaScript, HTML 5 e CSS 3 (LESS), além do ambiente de execução JavaScript, Node.Js, para criação do servidor e MongoDB para o banco de dados não relacional.

Assim espera-se que a aplicação possa apoiar as duas primeiras fases do ciclo de gestão de emergência (mitigação e preparação), uma vez que oferece suporte às três primeiras fases da abordagem para avaliação de planos de emergência utilizando as técnicas de FTR e Teste de Mutação (planejamento, orientação e preparação individual). De tal forma, espera-se que o aplicativo possa ser utilizado como base para uma equipe que precisa desenvolver seu plano de emergência específico.

Os benefícios de utilizar técnicas de teste de software é garantir que não há elementos faltando ao avaliar um procedimento lado a lado com um cenário específico, e, a partir dos mutantes, também verificar que não há etapas desnecessárias. Uma das limitações da aplicação é que a verificação dos mutantes é feita manualmente, onde o

usuário precisa avaliar um mutante criado contra um plano e um cenário e verificar se eles podem ser aceitos como equivalente ou descartados como mortos.

Com essas técnicas apoiando a avaliação do plano de emergência, temos maior confiança de que o plano de emergência esteja pronto para testes de simulação ou etapas posteriores, diminuindo custos caso algum problema tenha sido resolvido com apoio do sistema que de outra forma só teria sido percebido durante a simulação ou durante a emergência.

6.2 Trabalhos Futuros

O sistema implementa até a fase 2 (preparação individual) da abordagem de avaliação de planos de emergência utilizando FTR e Teste de Mutação, apresentado na Figura 28. Ele serve como uma ferramenta de apoio à decisão, onde os dados são criados e organizados para serem manualmente revisados e avaliados pela equipe.

A ideia de continuação da implementação do sistema se faz necessária para que (a) o número de funções que auxiliam na avaliação e monitoramento dos planos de emergência sejam aumentados e melhorados e (b) a parte de validação e revisão do plano de emergência deixe de ser somente um simples apoio à decisão e se torne uma ferramenta complexa, que gere e avalie os mutantes de forma automática e forneça dados estatísticos conforme necessário.

As funcionalidades previstas anteriormente que ficaram pendentes e podem ser implementados em trabalhos futuros estão listadas na Tabela 2.

Tabela 2: Status das funcionalidades para suporte tecnológico à abordagem.

Fase da abordagem	Funcionalidade	Implementada
Planejamento	Desenvolvimento e/ou envio de planos de emergência que precisam ser avaliados: Os autores podem desenhar um novo plano de emergência, enviar um existente ou modificar seus planos utilizando a ferramenta.	Sim
	Formatar o plano de emergência sob os elementos estáticos e variáveis: A ferramenta pode induzir a criação de planos de emergência utilizando elementos estáticos e variáveis. Se o plano de emergência não foi criado utilizando a ferramenta, deveria ser possível formatar o plano de acordo com	Sim

	estes elementos.	
	Geração de cenários para o plano de emergência sob avaliação: O time de revisão pode gerar cenários que serão utilizados nos planos de emergência. Estes cenários podem ser previamente armazenados em um banco de cenários e serem recuperados de acordo com o plano de emergência no escopo.	Sim
	Simulação de diferentes cenários sobre plano de emergência em avaliação: O time de revisão pode retornar um conjunto de cenários e executá-los contra o plano de emergência. Através dessa ação, é possível analisar o comportamento do plano de emergência, verificando se ele parece estar correto.	Sim
Preparação Individual	Geração de planos mutantes: Os revisores podem remover partes do plano de emergência sob avaliação, gerando um conjunto de planos mutantes. Estes planos mutantes devem ser compartilhados entre aqueles que estão envolvidos com a avaliação do plano de emergência, com o objetivo de evitar duplicidade dos planos mutantes.	Sim
	Apoio à revisão de planos mutantes: A ferramenta pode apoiar as atividades do revisor e do time de revisão, incluindo a análise dos planos mutantes e registro do status deles (morto, requer avaliação durante reunião de revisão, equivalente ao plano original, requer novos cenários etc).	Sim
Reunião de revisão	Registro dos problemas do plano de emergência: A ferramenta precisa registrar os problemas identificados no plano de emergência durante o processo de avaliação e permitir a recuperação pelos autores durante os ajustes no plano de emergência.	Sim
Retrabalho	Monitoramento da evolução do plano de emergência: A ferramenta pode rastrear a evolução do plano de emergência, permitindo o registro de mudanças feitas ao plano de emergência original (histórico de mudanças) e o registro de como e se os problemas identificados no plano de emergência foram resolvidos.	Não

Verificação	Geração de estatísticas: A ferramenta poderia suportar a geração do recorde de mutação, que pode ser utilizado como uma métrica para avaliar o nível de qualidade do plano de emergência e consequentemente a confiança nele.	Não
Todas	Notificação de participantes: A ferramenta precisa suportar a notificação de quem está envolvido no processo de avaliação do plano de emergência. Essas pessoas deveriam ser informadas quando um plano de emergência precisa ser avaliado, quando planos mutantes podem ser gerados, quando um plano de emergência precisa ser modificado, etc.	Não

Referências Bibliográficas

1. ÁVILA, S., Cerqueira, I., Junior, V., Nascimento, J., Peres, C., Ahumada, C. B., & Arezes, P. (2020) Integrated Management in Disaster: A Discussion of Competences in a Real Simulation. In: Arezes P. (eds) Advances in Safety Management and Human Factors. AHFE 2019. Advances in Intelligent Systems and Computing, vol 969. Springer, Cham.
2. DIIRR, B. & BORGES, M. (2013) Applying software engineering testing techniques to evaluate emergency plans. In Proceedings of the 10th International ISCRAM Conference, Baden-Baden, Germany, p. 758-763.
3. COPPOLA, D. P. (2006) Introduction to International Disaster Management, Elsevier Inc.
4. JOHNSON, P. (1994) An Instrumented Approach to Improving Software Quality through Formal Technical Review. In Proceedings of the 16th International Conference on Software Engineering (ICSE), Sorrento, Italy, p. 113-122.
5. FERREIRA, S. M. A. (2019) "Mutation-based Web Test Case Generation".
6. JIA Y., HARMAN M. (2011), "An analysis and survey of the development of mutation testing", IEEE Trans. Softw. Eng., vol. 37, no. 5, pp. 649-678.
7. VUSSC (2008) Disaster Management. Course Manual. Available at: <http://www.col.org/SiteCollectionDocuments/Disaster_Management_version_1.0.pdf>. Accessed: 14 nov. 2016
8. LEY, B.; LUDWIG, T.; PIPEK, V.; RANDALL, D.; REUTER, C.; WIEDENHOEFER, T. (2014) Information and expertise sharing in inter-organizational crisis management. Computer Supported Cooperative Work, Dordrecht, v. 23, n. 4, p. 347-387.
9. Site oficial do React. Disponível em: <https://pt-br.reactjs.org/> Acesso em: 08 de dezembo de 2019
10. Site oficial do NodeJS. Disponível em: <https://nodejs.org/pt-br/about/> Acesso em: 08 de dezembo de 2019
11. ”What is V8?”. Disponível em: <https://v8.dev/> Acesso em: 08 de dezembro de 2019

12. “Standard ECMA-262: ECMAScript”. Disponível em: <https://www.ecma-international.org/publications/standards/Ecma-262.htm> Acesso em: 08 de dezembro de 2019
13. Site oficial do Express. Disponível em <https://expressjs.com/pt-br/> Acesso em: 08/12/2019
14. GOGÓ, H. (2017) Introdução React.js. Disponível em <https://www.slideshare.net/henriquegogo/introduo-reactjs> Acesso em: 09/12/2019
15. Site oficial do NPM. Disponível em <https://www.npmjs.com/> Acesso em: 09/12/2019
16. Site oficial do MongoDB. Disponível em <https://www.mongodb.com/> Acesso em: 14/12/2019
17. Site oficial do Mongoose. Disponível em <https://mongoosejs.com/> Acesso em: 14/12/2019
18. Site oficial do MongoDB Atlas. Disponível em <https://www.mongodb.com/cloud/atlas> Acesso em: 14/12/2019
19. Site oficial do Lerna. Disponível em <https://lerna.js.org/> Acesso em: 15/12/2019
20. Site oficial do Atlassian. Disponível em <https://www.atlassian.com/git/tutorials/monorepos> Acesso em: 15/12/2019
21. AINAPURE, B. S. (2014) Software Testing & Quality Assurance , Technical Publications.