Avaliação da Automatabilidade como Estratégia de Garantia de Qualidade no Front-end do Projeto SobreVidas - "Câncer de Boca"

Tallya J. S. Barbosa[1], Gabriel F. dos Reis[1], Leandro Pedrosa[2], Luiza de O. Costa[1], Renata D. Braga[1], Rejane F. Ribeiro-Rotta[3]

[1]Instituto de Informática – UFG – Goiânia – GO – Brasil

[2]Programa de Pós-Graduação em Ciências da Saúde
Faculdade de Medicina - UFG - Goiânia - GO - Brasil

[3]Faculdade de Odontologia – UFG - Goiânia - GO - Brasil

{tallyabarbosa,freitas.gabriel,leandropedrosa,luizaluiza}
@discente.ufg.br, {renatadbraga,rejanefrr}@ufg.br,
oliveiraluiza2012ufg@gmail.com

Abstract. *Testability has gained relevance as a criterion for evaluating software quality. In critical systems, such as healthcare systems, ensuring that software is comprehensively testable and effectively tested is essential. In agile environments, automatability emerges as a key factor for software testability, enabling fast, repeatable, and reliable test execution. This study evaluates the automatability of the Web interface of the SobreVidas - "Oral Cancer" platform, identifying deficiencies and proposing improvements aimed at strengthening software quality assurance and enhancing system reliability.*

Resumo. *A testabilidade tem ganhado grande relevância como critério de avaliação da qualidade de software. Em sistemas críticos, como sistemas para a área da saúde, é essencial que o software seja amplamente testável e efetivamente testado. Em ambientes ágeis, a automatabilidade surge como um fator central para a testabilidade, possibilitando a execução rápida, repetível e confiável de testes. Este estudo avalia a automatabilidade da interface Web da plataforma SobreVidas – "Câncer de Boca", identificando deficiências e propondo melhorias voltadas ao fortalecimento da garantia de qualidade e ao aumento da confiabilidade do sistema.*

1. Introdução

A diversidade de dispositivos, o uso de *frameworks* modernos e a adoção de padrões de arquitetura aumentaram, significativamente, a complexidade das interfaces Web. Essa crescente complexidade, aliada ao fato de que a camada de *front-end* é o ponto primário de contato do usuário com o software, torna fundamental que práticas de Garantia de Qualidade de Software (*Software Quality Assurance* - SQA) sejam aplicadas ao desenvolvimento do *front-end*, assegurando a confiabilidade do *software* e uma experiência satisfatória ao usuário final [Ronchieri and Canaparo 2023].

O SobreVidas - "Câncer de Boca" é uma aplicação multiplataforma e multiusuário projetada com o intuito de rastrear e monitorar a população de risco na Atenção Primária

à Saúde, e apresenta diversas interfaces gráficas interativas. Em *softwares* voltados ao domínio da saúde, a robustez é ainda mais crítica, visto que falhas podem impactar diretamente na eficiência do atendimento e na segurança dos pacientes. Dessa forma, a qualidade do *software* transcende a simples usabilidade, configurando-se como uma questão de segurança em saúde pública [Alotaibi and Federico 2017].

A testabilidade é um atributo externo ao *software* que avalia a complexidade e o esforço necessário para testagem do *software* [Ratnani and Suri 2015]. A medida da testabilidade de um *software* é um resultado do conjunto formado pelo *software* a ser testado, os objetivos de teste, os métodos de teste e os recursos de teste [Binder 1994]. Em outras palavras, a testabilidade pode ser analisada sob múltiplas perspectivas e é altamente dependente do contexto de cada sistema ou componente, sendo determinante para a adoção de estratégias eficazes de validação [Obigbesan et al. 2024].

Em ambientes de desenvolvimento ágil, como é o caso do Projeto SobreVidas, em que os períodos de desenvolvimento são curtos e há a necessidade de entregas contínuas, o tempo para a realização minuciosa de testes manuais é reduzido. Nesse cenário, os testes automatizados surgem como aliados fundamentais, permitindo execuções rápidas, repetíveis e consistentes. Além de acelerar o processo de validação, a automação possibilita a detecção precoce de defeitos e libera a equipe para atividades mais complexas que elevam a qualidade do produto final [Basit et al. 2018].

Dessa forma, este estudo buscou avaliar a automatabilidade da interface Web da Plataforma SobreVidas, identificando lacunas que possam dificultar ou impedir a implementação de testes automatizados, além de propor práticas para mitigar essas limitações. O objetivo foi viabilizar uma automação de testes eficiente e sustentável, agregando valor à estratégia de SQA e contribuindo para que o sistema cumpra plenamente seu propósito de suporte ao rastreamento e monitoramento do câncer de boca.

As demais seções deste artigo estão organizadas da seguinte forma: a Seção 2 detalha o método empregado, que inclui uma análise documental e a implementação de uma Prova de Conceito (PoC) para uma estrutura de testes automatizados; a Seção 3 apresenta os resultados obtidos, destacando os principais desafios de automatabilidade encontrados; a Seção 4 discute as implicações desses achados para a manutenção e confiabilidade dos testes; e, por fim, a Seção 5 apresenta as conclusões do estudo e recomendações de trabalhos futuros.

2. Método

Inicialmente, foi realizada uma análise documental dos artefatos relacionados ao projeto, incluindo planos de *sprint*, *backlog* do produto, documentos de requisitos, definições de *design* e metas estabelecidas. O objetivo dessa etapa foi obter uma compreensão detalhada do produto, suas características e necessidades, bem como mapear as práticas de testagem aplicadas na camada de *front-end*.

Com base nessa análise, foi desenvolvida uma PoC (*Proof of Concept*/Prova de Conceito) para uma infraestrutura de testes automatizados no *front-end*. Para essa PoC, foram selecionadas duas abordagens principais:

- Testes e2e (*end-to-end*) baseados em *script*, que simulam a interação de um usuário com a aplicação em diferentes fluxos;

- Testagem aleatória (*monkey testing*), que gera interações não determinísticas na interface, de modo a explorar caminhos alternativos de uso não previstos nos casos de teste formais.

A implementação utilizou o *framework* Cypress [Cypress.io 2025] para os testes e2e, escolhido pela facilidade de instalação, integração e aceitação na comunidade de desenvolvedores. Para a testagem aleatória, foi adotada a biblioteca *Gremlins.js*, selecionada por sua capacidade de customização e integração com o Cypress. Ambos os recursos foram escolhidos por serem *open source*, favorecendo a utilização de tecnologias acessíveis e bem consolidadas, além de reduzir riscos de instabilidade ou descontinuidade.

3. Resultados

Na fase de análise, foi identificado que, ao longo de aproximadamente duas *sprints* do SobreVidas, nas quais a equipe empregou um esforço maior que o habitual para a realização de testes manuais, foram identificados e catalogados 45 *bugs*. Desses, 35 eram relacionados a *front-end*, e pelo menos 15 eram erros de funcionalidade que poderiam ter sido identificados por meio de testes automatizados.

Durante a implementação da PoC para os testes automatizados, foram identificados dois desafios principais: a ausência de identificadores consistentes e únicos nos elementos da interface; e a dificuldade de navegação em alguns componentes dinâmicos por meio de seletores CSS ou XPath.

3.1. Identificadores

A Figura 1 apresenta um exemplo de *suite* de testes automatizados criada para validação do fluxo de cadastro e consulta de pacientes. Nesse fluxo, observou-se que a ausência de identificadores bem definidos nos elementos da UI levou à criação de seletores frágeis, um tipo de "*test smell*" [Bicalho et al. 2024]. Esses seletores dependem excessivamente da estrutura do DOM (*Document Object Model*) ou de classes CSS genéricas, tornando os testes suscetíveis a falhas mesmo após pequenas alterações na interface.

Para ilustrar o problema, analisou-se a interface de aceite do Termo de Consentimento Livre e Esclarecido (TCLE) (Figura 2). Nela, o botão de confirmação do diálogo ("Confirmar") é renderizado sem atributos únicos, como *id* ou *label* (Figura 3).

Como evidenciado na Figura 3, o botão não possui um seletor único que possa ser utilizado para identificação. Assim, a seleção do botão foi feita pelo conteúdo textual via XPath `//button[text=' Confirmar ']`. Esse tipo de seletor é considerado frágil, pois depende diretamente do texto exibido e pode gerar testes não robustos, de alta manutenção e com risco de falsos-negativos [Bernardo 2011].

A solução para melhorar a automatabilidade, portanto, é a refatoração de código visando a implementação de boas práticas de testabilidade no desenvolvimento, como a atribuição de IDs únicos e estáveis aos principais componentes interativos [Bernardo 2011]. Alternativamente, recomenda-se o uso de atributos dedicados, como *data-testid* (Figura 4), que tornam os seletores mais concisos e resistentes a mudanças visuais, aumentando a confiabilidade dos testes.

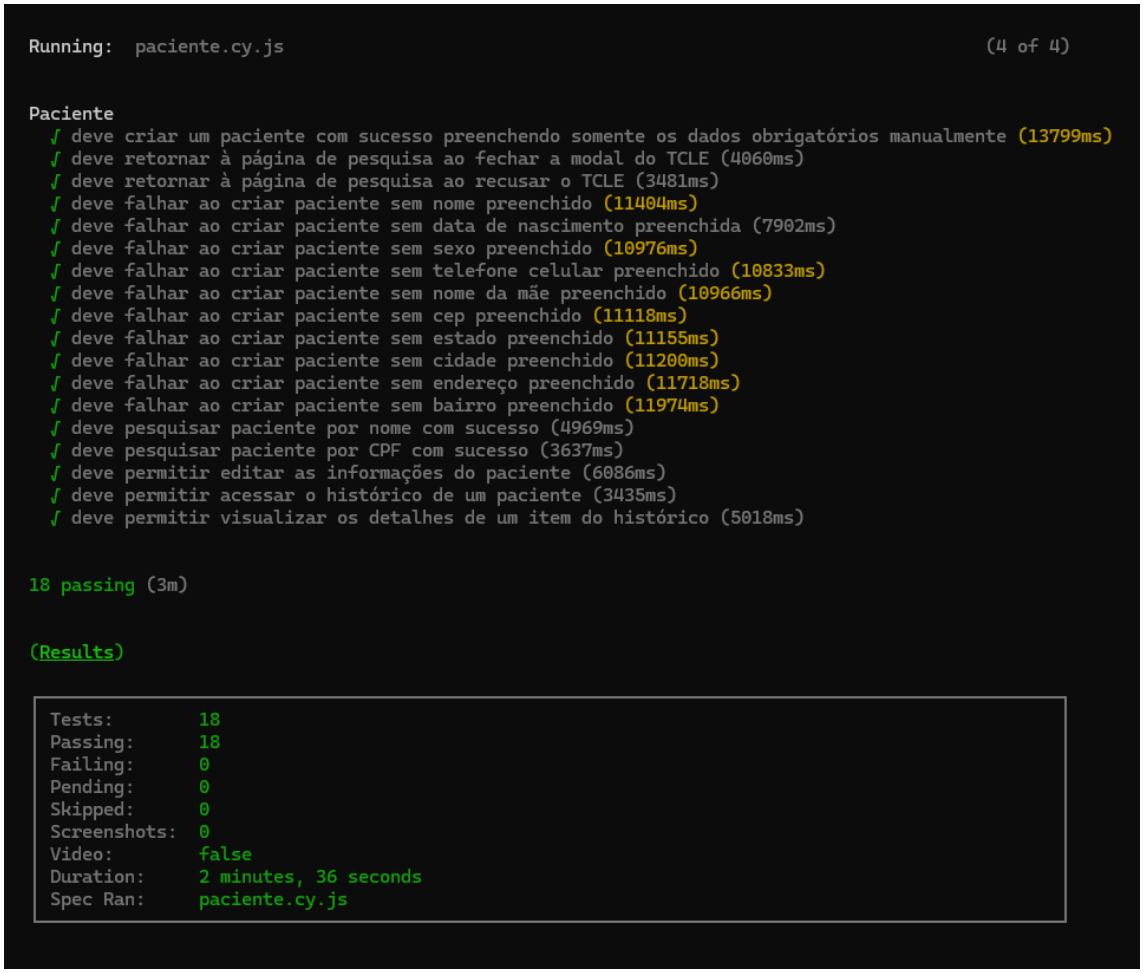

Figura 1. *Suite* de testes automatizados criada para validação do fluxo de cadastro e consulta de pacientes

3.2. Bibliotecas de componentes

Outro desafio identificado refere-se ao uso de bibliotecas de componentes com geração dinâmica de HTML. O *front-end* da Plataforma SobreVidas foi desenvolvido em Angular com a biblioteca *PrimeNG*, além do *FullCalendar* para a gestão de agendas.

A Figura 5 apresenta a tela de agenda semanal, onde os dias são divididos em intervalos de 30 minutos, enquanto a Figura 6 exibe parte do HTML gerado para esse componente.

Verificou-se que as linhas do calendário utilizam apenas classes genéricas e o atributo *data-time*, que corresponde à semana inteira, dificultando a diferenciação entre slots de dia e horário. Além disso, a disponibilidade de agendamento é representada apenas por cores (branco = disponível; cinza = indisponível), o que inviabiliza a automação por seletores.

Nesse cenário, pode ser possível realizar refatorações para que o componente se torne melhor navegável via *scripts* de testes automatizados. No caso da biblioteca *FullCalendar*, por exemplo, são oferecidos *hooks* que permitem interagir com o componente em diferentes níveis e adicionar atributos aos elementos. Caso esse recurso não seja suficiente

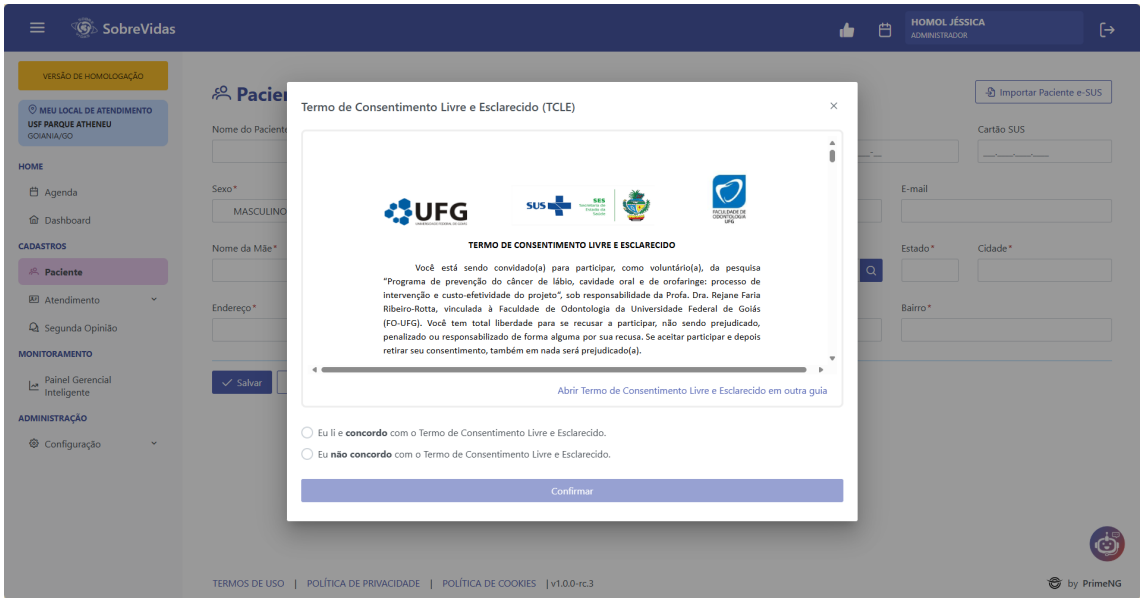

Figura 2. Tela de aceite do TCLE

```
<div _ngcontent-ng-c589903851 class="flex flex-column ng-tns-c1946505645-42"> flex
  <div _ngcontent-ng-c589903851 class="flex align-items-center justify-content-center ng-tns-c589903851-41">
    flex
    <button _ngcontent-ng-c589903851 pbutton class="p-element p-button-primary w-full text-center ng-tns-c589
    903851-41 p-button p-component" disabled style="display: block;"> Confirmar </button>
  </div>
</div>
```

Figura 3. HTML renderizado do botão "Confirmar"

e a biblioteca não ofereça outros recursos úteis, a equipe deverá avaliar a viabilidade da substituição da biblioteca por outra que ofereça esse suporte. Por isso, é importante levar em consideração a testabilidade e a automatabilidade desde a etapa de *design* do *software* [Ratnani and Suri 2015].

3.3. Testagem aleatória (*Monkey testing*)

A testagem aleatória foi realizada em três etapas diferentes de utilização da plataforma: após o login; após a seleção da unidade de atendimento (página de agenda); e na página de paciente (pesquisa e cadastro). A Figura 7 apresenta a suíte de testes desenvolvida nessa estratégia.

A biblioteca *Gremlins.js* foi utilizada para simular ações aleatórias como cliques, digitação, rolagem de tela e monitoramento de logs de erro. Até o momento, não foram identificados erros relevantes a partir dessa estratégia, mas ela demonstrou potencial como complemento aos testes e2e.

4. Discussão

A implementação da PoC para testes automatizados no *front-end* do Projeto SobreVidas expôs as barreiras de automatabilidade presentes na aplicação. Entre os principais desafios, destaca-se a ausência de identificadores únicos e estáveis nos elementos da interface de usuário (UI). A dependência de seletores frágeis, baseados apenas na estrutura

```
▼ <div _ngcontent-ng-c589903851 class="flex flex-column ng-tns-c1946505645-42"> flex
  ▼ <div _ngcontent-ng-c589903851 class="flex align-items-center justify-content-center ng-tns-c589903851-41">
    flex
      <button _ngcontent-ng-c589903851 pbutton class="p-element p-button-primary w-full text-center ng-tns-c589
      903851-41 p-button p-component" disabled style="display: block;" data-testid="confirmar-tcle"> Confirmar
      </button>
    </div>
  </div>
```

Figura 4. Código da Figura 3 modificado para maior automatabilidade

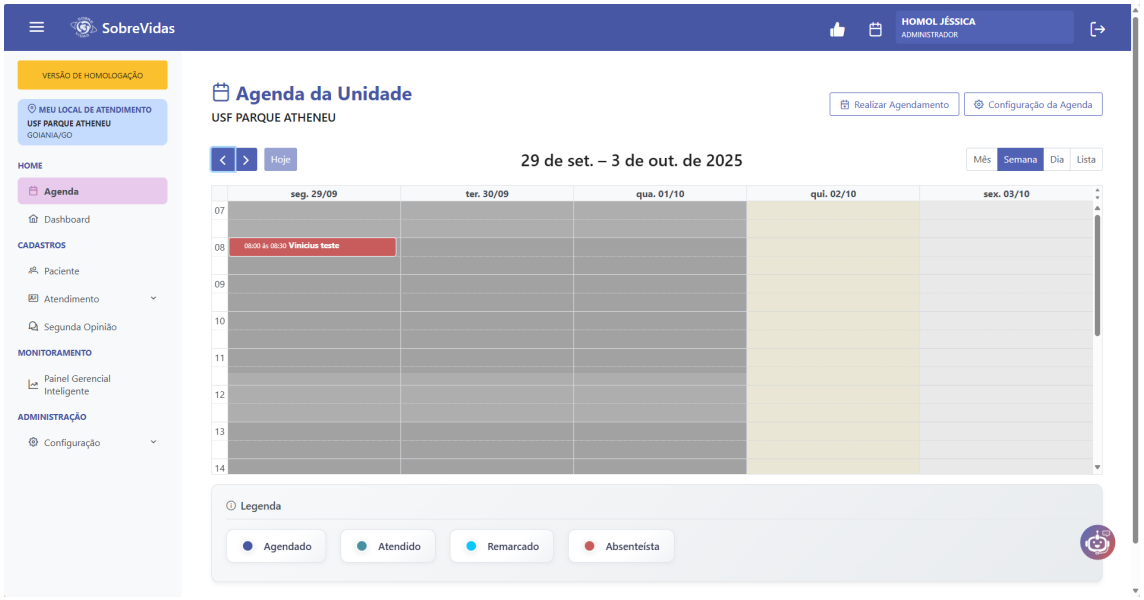

Figura 5. Tela de agenda da unidade com exibição do calendário

do DOM ou em conteúdo textual, aumenta a complexidade e o custo de manutenção dos *scripts* de teste, além de comprometer a confiabilidade dos resultados, consumindo tempo da equipe e reduzindo a efetividade do processo de automação [Parry et al. 2021].

A adoção de atributos dedicados para testes representa uma solução robusta para esse problema. Essa prática dissocia os testes automatizados da implementação visual e estrutural, acoplando-os à implementação funcional, que é o alvo da validação. Dessa forma, alterações visuais podem ser realizadas sem impactar a suíte de testes. No entanto, a implementação dessa abordagem requer comprometimento da equipe de desenvolvimento, que deve tratar a testabilidade e a automatabilidade como requisitos do *software* desde as etapas iniciais do ciclo de vida, e não como uma atividade secundária [Xu et al. 2025].

Além disso, a análise revelou que a utilização de bibliotecas de componentes de terceiros, como a *FullCalendar*, embora acelere o desenvolvimento, pode introduzir desafios de automatabilidade. A dificuldade em simular, por meio de *scripts*, ações básicas de usuários, como o agendamento de consultas em horários disponíveis, transforma funcionalidades-chave em "caixas-pretas" para os testes automatizados. Esse cenário reduz os benefícios da automação e obriga a manutenção de testes manuais em fluxos essenciais, o que contraria os objetivos de agilidade e entrega contínua característicos de ambientes ágeis [De Luca et al. 2024].

Figura 6. HTML renderizado para alguns horários do calendário

Nesse contexto, é fundamental que sejam avaliadas alternativas como a customização da biblioteca existente ou, quando necessário, a substituição por ferramentas mais adequadas à automação. Essa decisão deve considerar não apenas os custos imediatos de adaptação ou troca, mas também os custos decorrentes da manutenção de testes manuais e os riscos associados à introdução de falhas em produção. As dificuldades observadas reforçam a necessidade de considerar a automatabilidade como critério estratégico desde a concepção do sistema, incluindo a seleção de *frameworks* e bibliotecas externas.

5. Conclusões

Este estudo avaliou a automatabilidade da interface web do Projeto SobreVidas - "Câncer de Boca" e identificou deficiências que impedem a implementação eficaz de uma estratégia de testes automatizados. Entre os principais obstáculos destacam-se a ausência de identificadores únicos nos elementos da interface e a baixa acessibilidade de componentes gerados por bibliotecas externas, como os calendários interativos. Essas barreiras resultam na criação de testes frágeis, de alta manutenção e com baixa confiabilidade.

Os achados reforçam que, em sistemas críticos na área da saúde, a confiabilidade é um pilar de segurança para os usuários, e, portanto, a garantia de qualidade de *software* não pode ser negligenciada. A automação de testes no *front-end* configura-se como ferramenta essencial para elevar a robustez do sistema, permitindo execuções rápidas, repetíveis e consistentes, além de viabilizar a detecção precoce de defeitos que poderiam impactar diretamente o atendimento a pacientes.

Conclui-se que é imperativo para a equipe do Projeto SobreVidas adotar práticas de desenvolvimento que promovam a testabilidade desde a concepção do sistema. A implementação das melhorias propostas neste estudo tornará a automação de testes viável

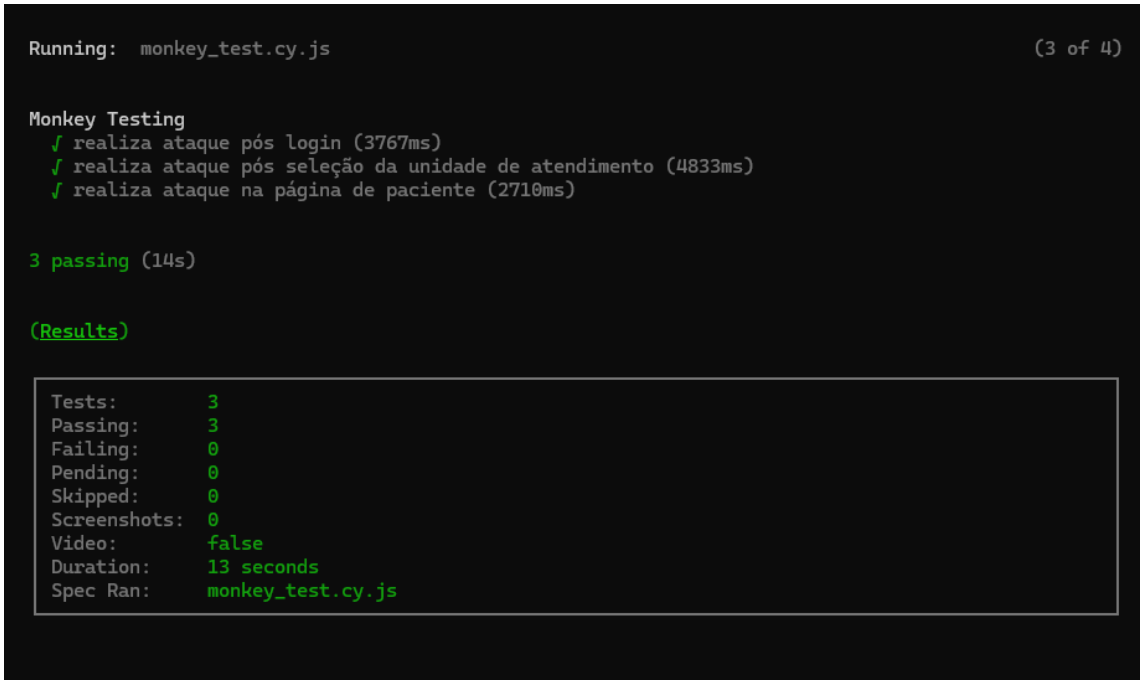

Figura 7. Suíte de testagem aleatória

e sustentável, fortalecerá as práticas de SQA do projeto e ampliará a confiabilidade e a qualidade do *software* entregue aos profissionais de saúde e à população atendida.

Recomenda-se, para trabalhos futuros, a implantação das melhorias sugeridas neste artigo e de uma estrutura robusta de testes automatizados na plataforma SobreVidas, com alta cobertura de código e execução automatizada *via* esteira de CI/CD. É recomendável também o estudo da implementação de bibliotecas e *frameworks* com geração dinâmica de HTML, a fim de entender suas particularidades de implementação e identificar oportunidades de melhoria quanto à automatabilidade.

6. Agradecimentos

Agradecemos ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo financiamento desta pesquisa, concedido através da Chamada nº 21/2023 – Estudos Transdisciplinares em Saúde Coletiva.

Referências

Aghazadeh, S., Pirnejad, H., Aliyev, A., and Moradkhani, A. (2015). Evaluating the effect of software quality characteristics on health care quality indicators. *Journal of Health Management*, 2:67–73.

Aho, P. and Vos, T. (2018). Challenges in automated testing through graphical user interface. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 118–121.

Alotaibi, Y. K. and Federico, F. (2017). The impact of health information technology on patient safety. *Saudi Medical Journal*, 38(12):1173–1180.

Basit, M., Baldwin, K., Kannan, V., Flahaven, E., Parks, C., Ott, J., and Willett, D. (2018). Agile acceptance test–driven development of clinical decision support advisories: Feasibility of using open source software. *JMIR Medical Informatics*, 6:e23.

Bernardo, P. C. (2011). Padrões de testes automatizados. Dissertação (mestrado em ciência da computação), Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

Bicalho, L., Montandon, J., and Valente, M. (2024). Identificação de smells em testes fim-a-fim implementados em cypress. In *Anais do XII Workshop de Visualização, Evolução e Manutenção de Software*, pages 1–12, Porto Alegre, RS, Brasil. SBC.

Binder, R. V. (1994). Design for testability in object-oriented systems. *Commun. ACM*, 37(9):87–101.

Cypress.io (2025). Why cypress? `https://docs.cypress.io/app/get-started/why-cypress` [Accessed: 2025-09-05].

De Luca, M., Fasolino, A. R., and Tramontana, P. (2024). Investigating the robustness of locators in template-based web application testing using a gui change classification model. *Journal of Systems and Software*, 210:111932.

IEEE (2014). IEEE Standard for Software Quality Assurance Processes. *IEEE Std 730-2014 (Revision of IEEE Std 730-2002)*, pages 1–138.

International Organization for Standardization (2011). ISO/IEC 25010: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

Kaur, G. and Tiwari, R. G. (2023). Comparison and analysis of popular frontend frameworks and libraries: An evaluation of parameters for frontend web development. In *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 1067–1073.

Kulesza, R., Sousa, M., Lima, M., Araujo, C., and Filho, A. (2018). *Evolução das Arquiteturas de Software Rumo à Web 3.0*, pages 1–38.

Lalband, N. and Dwaram, K. (2019). Software engineering for smart healthcare. *International Journal of Innovative Technology and Exploring Engineering*, 8:325–331.

Obigbesan, O., Graham, K., and Benzies, K. M. (2024). Software testing of ehealth interventions: Existing practices and the future of an iterative strategy. *JMIR Nursing*, 7:e56585.

Parry, O., Kapfhammer, G. M., Hilton, M., and McMinn, P. (2021). A survey of flaky tests. *ACM Trans. Softw. Eng. Methodol.*, 31(1).

Pathak, K., Ninoria, S., and Bharadwaj, S. (2022). Scope of agile approach for software testing process. In *2022 11th International Conference on System Modeling Advancement in Research Trends (SMART)*, pages 1079–1083.

Ratnani, H. and Suri, P. (2015). Object oriented software testability survey at designing and implementation phase. *International Journal of Science and Research (IJSR)*, 438.

Richardson, I., Reid, L., and O'Leary, P. (2016). Healthcare systems quality: development and use. pages 50–53.

Ronchieri, E. and Canaparo, M. (2023). Assessing the impact of software quality models in healthcare software systems. *Health Systems*, 12:1–13.

Smith, A. and Jones, B. (1999). On the complexity of computing. In *Advances in Computer Science*, pages 555–566. Publishing Press.

Xu, Z., Li, Q., and Tan, S. H. (2025). Guiding ChatGPT to Fix Web UI Tests via Explanation-Consistency Checking.