UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

Gabriel Donadel Dall'Agnol

Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica

Florianópolis - SC
2022

Gabriel Donadel Dall'Agnol

Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica

Trabalho de Conclusão do Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Jean Everson Martina, Dr.

Florianópolis - SC
2022

Dedico este trabalho a toda a comunidade entusiasta das criptomoedas, em especial àqueles que têm desenvolvido ferramentas para facilitar seu uso em nossa sociedade.

AGRADECIMENTOS

Em primeiro lugar aos meus pais que me motivaram e me deram forças para concluir este trabalho, sempre me incentivando a batalhar por meus objetivos e me apoiando nos momentos difíceis.

À minha namorada Juliana por seu amor e carinho, estando ao meu lado durante toda graduação e sendo obrigada a ouvir minhas explicações semanais sobre os temas aprendidos em sala de aula.

Agradecer aos meus colegas e amigos de faculdade, em especial ao José Augusto, Peter e Samuel que estiveram presentes fazendo trabalhos em inúmeras madrugadas nestes últimos quatro anos e acompanharam de perto minha jornada na graduação de sistemas de informação. Amizades que vou levar para a vida toda.

À Universidade Federal de Santa Catarina e todo o seu corpo docente, essencial no meu processo de formação profissional, e por tudo o que aprendi ao longo dos anos do curso.

E a todos que de alguma forma contribuíram com meu trabalho e com minha formação.

“A raiz do problema com as moedas convencionais
é toda a confiança necessária
para fazê-las funcionarem.“
(NAKAMOTO, 2009)

RESUMO

O mercado vem experienciando uma alta demanda por mais plataformas de desenvolvimento e implantação de software baseadas em interface gráfica, focadas na usabilidade e acima de tudo experiência do usuário. Em paralelo, a tecnologia *blockchain* tem se tornado cada vez mais popular, especialmente no que se refere a plataforma Ethereum, que tem como diferencial a criação de contratos inteligentes e o desenvolvimento de aplicações descentralizadas. Porém, a falta de uma interface amigável e o requerimento de habilidades técnicas especificas criam uma alta barreira de entrada para o público geral, fazendo com que poucos estejam aptos a extrair o potencial máximo da tecnologia e de fato utilizem contratos inteligentes. O objetivo deste trabalho é desenvolver uma plataforma web *low-code* com enfoque em usabilidade que possibilite a criação e implantação de contratos inteligentes na rede Ethereum.

Palavras-chave: *Smart Contracts*. *Blockchain*. *Low-code*. Ethereum.

ABSTRACT

The business market has been experiencing a high demand for more development and deployment platforms based on graphical user interfaces that are focused on usability and above all, user experience. In parallel, the blockchain technology has become increasingly popular, especially the Ethereum platform, which has the differential of creating smart contracts and the development of decentralized applications. However, the lack of a user-friendly interface and the requirement of specific technical skills create a high barrier to entry for the general public, making few able to extract the full potential of the technology and actually use smart contracts. This work aims to develop a low-code web platform with a focus on usability that enables the creation and implementation of smart contracts on the Ethereum network.

Keywords: Smart Contracts. Blockchain. Low-code. Ethereum.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABI	*Application Binary Interface*
API	*Application Programming Interface*
BaaS	*Backend as a Service*
CLI	*Command-line interface*
CSS	*Cascading Style Sheets*
DApps	*Decentralized applications*
EVM	*Ethereum virtual machine*
HTML	*HyperText Markup Language*
JSON	*JavaScript Object Notation*
RF	Requisitos Funcionais
RNF	Requisitos Não-Funcionais
SDK	*Software Development Kit*
STEM	*Science, Technology, Engineering e Mathematics*
VM	*Virtual machine*

SUMÁRIO

1 INTRODUÇÃO

A tecnologia *blockchain* vem ganhando muita força nos últimos anos, estando presente nos mais diversos setores, desde a área da agricultura, saúde, finanças, logística, setores governamentais e na indústria em geral (VADGAMA; TASCA, 2021). Entre as diversas *blockchains* existentes no mercado destaca-se a utilizada pela moeda digital Ether, a plataforma Ethereum, que tem como principal diferencial a possibilidade da criação de contratos inteligentes e o desenvolvimento de aplicações descentralizadas (DESTEFANIS, 2021).

No entanto, esta tecnologia ainda é relativamente nova, em especial quando falamos de contratos inteligentes, que foram apenas introduzidos em 2015 com o lançamento oficial da rede Ethereum. Atualmente, a falta de uma interface amigável para o desenvolvimento de contratos inteligentes e o requerimento de habilidades técnicas criam uma alta barreira de entrada para o público geral, fazendo com que poucos estejam aptos a extrair o potencial máximo da tecnologia e de fato utilizem contratos inteligentes, os quais possibilitam negociações confiáveis sem a necessidade de uma entidade mediadora (TSANKOV *et al.*, 2018).

Ao mesmo tempo observa-se uma alta demanda no mercado por plataformas *low-code*, plataformas de desenvolvimento e implantação de software baseadas em interface gráfica com abstrações visuais, de forma a exigir pouco ou nenhum desenvolvimento de código. (SAHAY *et al.*, 2020)

É neste contexto que o trabalho será elaborado, visando compreender as problemáticas no cenário dos contratos inteligentes frente à análise da usabilidade de plataformas baseadas em interface gráfica e desenvolver uma plataforma web *low-code* que possibilite o desenvolvimento e a implantação de contratos inteligentes na rede Ethereum.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem por objetivo desenvolver uma plataforma web *low-code* que possibilite a criação e implantação de contratos inteligentes na rede Ethereum, por meio de uma interface amigável e intuitiva, sem a necessidade de conhecimento prévio em programação.

1.1.2 Objetivos Específicos

- Reconhecer na literatura especializada o estado da arte de plataformas *low-code*.
- Desenvolver uma plataforma *low-code* para a criação de contratos inteligentes.

- Integrar a plataforma desenvolvida para possibilitar a implantação de contratos inteligentes na rede.
- Avaliar a usabilidade da plataforma desenvolvida.

2 MÉTODO DE PESQUISA

Inicialmente o trabalho terá um caráter de pesquisa exploratória, fase onde serão realizados estudos sobre a fundamentação teórica e revisão do estado da arte quanto a contratos inteligentes e plataformas *low-code*.

A metodologia dá o viés durante a pesquisa, mostrando os passos a serem seguidos pelo acadêmico para que possa atingir os objetivos traçados no estudo.

A pesquisa exploratória é o ponto de partida para todo trabalho científico, sobretudo quando bibliográfica, proporcionando maiores informações sobre determinado tema, facilitando a delimitação de determinado assunto, definindo os objetivos e descobrindo novo tipo de enfoque para o trabalho que se tem em mente (ANDRADE, 2010).

Para Gil (2002, p. 41), a “pesquisa exploratória tem como objetivo principal o aprimoramento de ideias ou a descoberta de intuições“. Será utilizado de mesmo modo a pesquisa bibliográfica, a qual contará com a análise da produção bibliográfica de diversos autores referência, que geram conhecimento de ponta a respeito dos assuntos abordados na pesquisa. Motivo pelo qual este tipo de pesquisa continua sendo uma das mais importantes para elaborações de trabalhos técnicos, os quais exigem maior profundidade teórica.

Quanto a abordagem, o estudo será qualitativo. A abordagem qualitativa de acordo com Oliveira (2007, p. 37), “[...] é um processo de reflexão e análise da realidade através da utilização de métodos e técnicas para a compreensão detalhada do objetivo de estudo em seu contexto histórico e/ ou segundo sua estruturação“.

Após esta análise serão levantados os requisitos que visam identificar as principais dores em relação a usabilidade no desenvolvimento de contratos inteligentes.

Em seguida se dará inicio a prototipação e ao desenvolvimento da plataforma, onde será avaliada a usabilidade a partir de uma abordagem qualitativa por meio de pesquisa aplicada de caráter experimental.

3 FUNDAMENTAÇÃO TEÓRICA

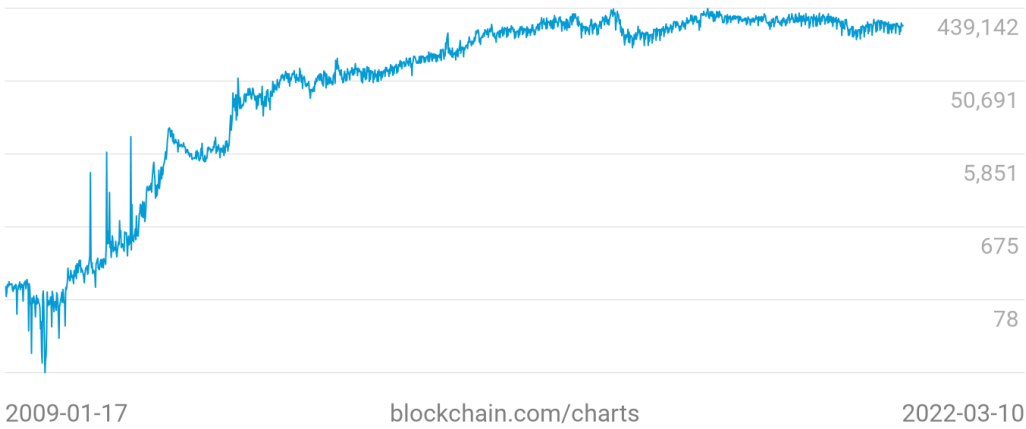
3.1 BLOCKCHAIN

O primeiro conceito de *blockchain* surgiu em 2008 no whitepaper de Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System* (NAKAMOTO, 2008). Em sua publicação Satoshi propõe uma criptomoeda descentralizada chamada Bitcoin que utilizaria um sistema de lista de blocos de transações facilmente verificáveis. Essa tecnologia tempo depois viria a ser conhecida como *blockchain* e permitiria pela primeira vez a validação de transações sem a existência de unidade central verificadora.

A principal característica da *blockchain* é o forte acoplamento entre seus blocos, onde cada bloco, com exceção do Gênesis, primeiro bloco de uma *blockchain*, armazena em seu cabeçalho o *hash* SHA-256 do bloco anterior, tornando assim a sequência de blocos robusta e prevenindo que informações passadas possam ser sobrescritas por meio de métodos tradicionais.

Como é possível observar na Figura 1 a adoção do Bitcoin e o uso da *blockchain* vem ganhando muita força nos últimos anos, sendo que apenas na *blockchain* do Bitcoin ocorrem cerca de dez milhões de transações a cada mês, todas sendo verificadas e armazenadas em sua *blockchain* que atualmente já ultrapassa os 390 GB em tamanho.

Figura 1 – Número de transações de bitcoin por mês em escala logarítmica.

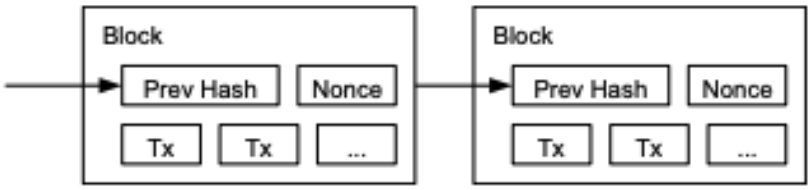


Fonte: Blockchain.com (2022).

Ainda que esta tecnologia tenha surgido com o propósito específico de resolver o gasto duplo em sistemas financeiros descentralizados, seu uso se expandiu para os mais diversos setores, sendo atualmente utilizada na área da saúde, mídia, entretenimento, jogos, manufatura, em setores governamentais e na indústria em geral como aponta o relatório realizado pela comissão europeia (HOW. . . , 2019).

Os supracitados blocos são os elementos que compõem a estrutura básica da *blockchain*, na qual cada bloco é considerado um conjunto de informações. Estes blocos são interligados formando uma cadeia de blocos encadeados, onde cada bloco contém uma referência ao bloco anterior, que no caso do Bitcoin, utiliza um *hash*. Pode-se ter uma representação visual a partir da ilustração na Figura 2.

Figura 2 – Blocos encadeados.



Fonte: Extraído de Nakamoto (2008, p. 3)

Além da referência ao bloco anterior, que é uma condição de existência do bloco, outras informações são armazenadas e, portanto, o tamanho final deste conjunto de dados (blocos) pode variar de solução para solução. No caso do Bitcoin, o bloco tem o tamanho de 1 megabyte, o que é suficiente para armazenar até 4000 transações, além das informações adicionais como data e hora, versão, *nonce* entre outras. Importante destacar que o *nonce* nada mais é do que um número que deve ser descoberto no processo de mineração. O conteúdo de cada bloco depende do propósito da criação da *blockchain*, que apesar de possuir uma forte ligação com bitcoin, atualmente seu conceito transcende este modelo de aplicação, sendo hoje utilizado em diversos contextos.

Com tamanha quantidade de dados transitando através dos blocos que formam a rede, é evidente a necessidade da existência de um mecanismo de validação das informações inseridas. A solução proposta por Satoshi Nakamoto para garantir a integridade dos dados foi descentralizar esta verificação de modo que qualquer computador possa tornar-se um nodo de validação e resolver um bloco na rede distribuída.

Por se tratar de um sistema distribuído é necessário que haja um consenso entre a maioria dos nodos dessa rede, e é nesse momento que surgem os algoritmos de validação. Atualmente existem diversos algoritmos de validação como: *Proof-of-Work* (NAKAMOTO, 2008), *Proof-of-Stake* (KING; NADAL, 2012), *Proof-of-Authority* (DE ANGELIS *et al.*, 2018), etc.

No caso do Bitcoin nodos de validação são incentivados a resolverem blocos através da inserção de novas moedas na rede, desta forma diversas pessoas investem em equipamentos para “minerar“ blocos, onde cada nodo disponibiliza uma determinada quantidade de recurso computacional que será destinado a resolver cálculos

complexos que certificam a veracidade do conteúdo do bloco e os propagam, ampliando a rede de blocos verificados.

Manter a consistência entre o histórico de dados que compõem a *blockchain* é um dos pilares desta inovação, afinal todo conteúdo se resume “a zeros e uns“ que transitam por cabos e fibras pelo mundo. A consistência é dada pelo algoritmo de equalização escolhido pela tecnologia em questão, que neste caso é determinado pelo galho mais longo da árvore.

O *Proof-of-Work* é o mais famoso destes algoritmos pois é o utilizado para a validação de blocos Bitcoin e diversas outras criptomoedas. Através desse mecanismo, qualquer nó pode provar que realizou um gasto de poder computacional para realizar um desafio matemático. No caso do Bitcoin, o desafio consiste em entrar um valor (nonce) que quando aplicado o *hash* SHA-256, junto ao *hash* do bloco anterior e os dados do bloco que está sendo validado, resultará em um novo *hash* com uma quantidade de bits zero pré determinada, sendo assim, o aumento da quantidade de bits zero eleva a dificuldade do desafio de forma exponencial.

Uma vez encontrado o *nonce* resultante no *hash* que resolve o bloco, seu valor é divulgado aos outros nodos da rede, processando o bloco e dando direito ao minerador de receber a recompensa pelo descobrimento do *nonce*.

Assim que a publicação do bloco acontece, os outros nodos participantes podem então verificar a validade do bloco de mineração aplicando a mesma metodologia, porém, como o nonce foi publicado junto ao bloco, basta uma simples operação para assegurar sua legitimidade tendo em vista que os valores a serem testados já são conhecidos.

Este algoritmo define que cada *nonce* pode ser utilizado uma vez, porém, não garante a não existência de um outro número que quando aplicado a mesma metodologia de verificação resulte em um *hash* com a mesma quantidade de bits pré definida, desta maneira existiriam duas validações diferentes para um mesmo bloco causando uma bifurcação. Neste tipo de situação os nodos costumam esperar o processamento de novos blocos e consideram a cadeia de blocos que se tornar mais longa, pois como a validação é baseada no *hash* do bloco anterior, quanto maior a cadeia de blocos mais robusto histórico da rede se torna, pois a alteração de um determinado bloco resulta na reescrita de todos os blocos posteriores a ele, já que o *hash* resultante da validação será alterado também.

Outra característica importante do *Proof-of-Work* é one-CPU-one-vote, onde o consenso é baseado em ter-se a maioria da rede, ou seja, 51% do poder computacional em acordo com os dados validados, sendo assim para controlar o fluxo das informações seria necessário ter um poder computacional superior ao poder total da rede.

3.2 ETHEREUM

O Ethereum é uma plataforma descentralizada baseada em *blockchain* que possibilita a construção e execução de contratos inteligentes e de aplicativos distribuídos chamados *Decentralized applications* (DApps), fazendo uso de uma criptomoeda própria chamada Ether (DESTEFANIS, 2021).

A principal diferença do Ethereum quando comparado a *blockchain* tradicionais como a do Bitcoin, é que a plataforma Ethereum fornece uma linguagem de programação Turing-completa, que permite a criação e execução de programas na *blockchain*.

De maneira geral o Ethereum opera usando contas e seus saldos que podem ser alterados por meio de transições de estado. O estado denota os saldos atuais de todas as contas e pode armazenar outros dados adicionais. Vale ressaltar que o estado não é armazenado diretamente na *blockchain*, sendo codificado e mantido por contas em uma estrutura de dados separada, esta organizada em árvores de Merkle. De forma similar a outras blockchains descentralizadas, as contas na rede Ethereum são pseudônimas e estão vinculadas a um ou mais endereços, assim garantindo o anonimato (FERRETTI; D'ANGELO, 2020).

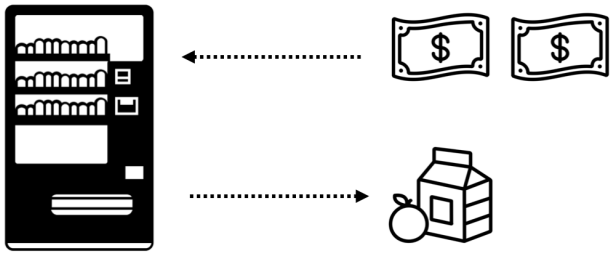
Existem dois tipos de contas na rede, contas de propriedade externa e contas de contratos. As contas de propriedade externa são controladas por pessoas, de maneira semelhante ao Bitcoin, onde cada pessoa possui sua própria chave privada que é usada para realizar transações na *blockchain*. No entanto as contas de contrato são controladas por algum código de contrato inteligente, pode-se dizer que tais contas são uma espécie de entidades cibernéticas, que possuem saldo próprio e podem ser acionadas por meio de transações, sendo estas provenientes de uma conta externa ou ainda de outros contratos. Uma vez acionado, o código especificado no contrato é executado e pode gerar ainda outras transações. É justamente a presença desses contratos inteligentes que permite que desenvolvedores utilizem o Ethereum como um *framework* de propósito geral para criar DApps.

Já o Ether é a criptomoeda empregada na *blockchain* Ethereum, de certa forma o Ether é o combustível utilizado para operar os DApps na rede. Por meio desta criptomoeda é possível efetuar pagamentos para outras contas ou para os endereços de contratos, que executam alguma operação solicitada, permitindo assim, executar DApps, habilitar contratos inteligentes, gerar novos tokens e também para fazer pagamentos P2P padrão.

3.3 CONTRATOS INTELIGENTES

O conceito de contratos inteligentes foi primeiramente proposto em 1994 por Nick Szabo, jurista e cientista da computação que definiu o termo *Smart Contracts* como “[...] um protocolo de transação computadorizado que executa os termos de um contrato“ (SZABO, 1994). Para Szabo, os objetivos gerais de um contrato inteligente seriam satisfazer as condições contratuais de todas as partes, desde termos de pagamento, garantias, confidencialidade, além de minimizar a necessidade de intermediários confiáveis.

Figura 3 – Interação com uma maquina de vendas.



Fonte: Matt Stokes, 2021

Uma analogia que representa adequadamente esta situação é a interação entre uma pessoa e uma máquina de vendas automática, representada na Figura 3, onde mesmo sem a existência de um atendente, o contrato de venda de uma bebida a um preço anunciado ao cliente é executado assim que o cliente insere quantia suficiente de dinheiro na máquina.

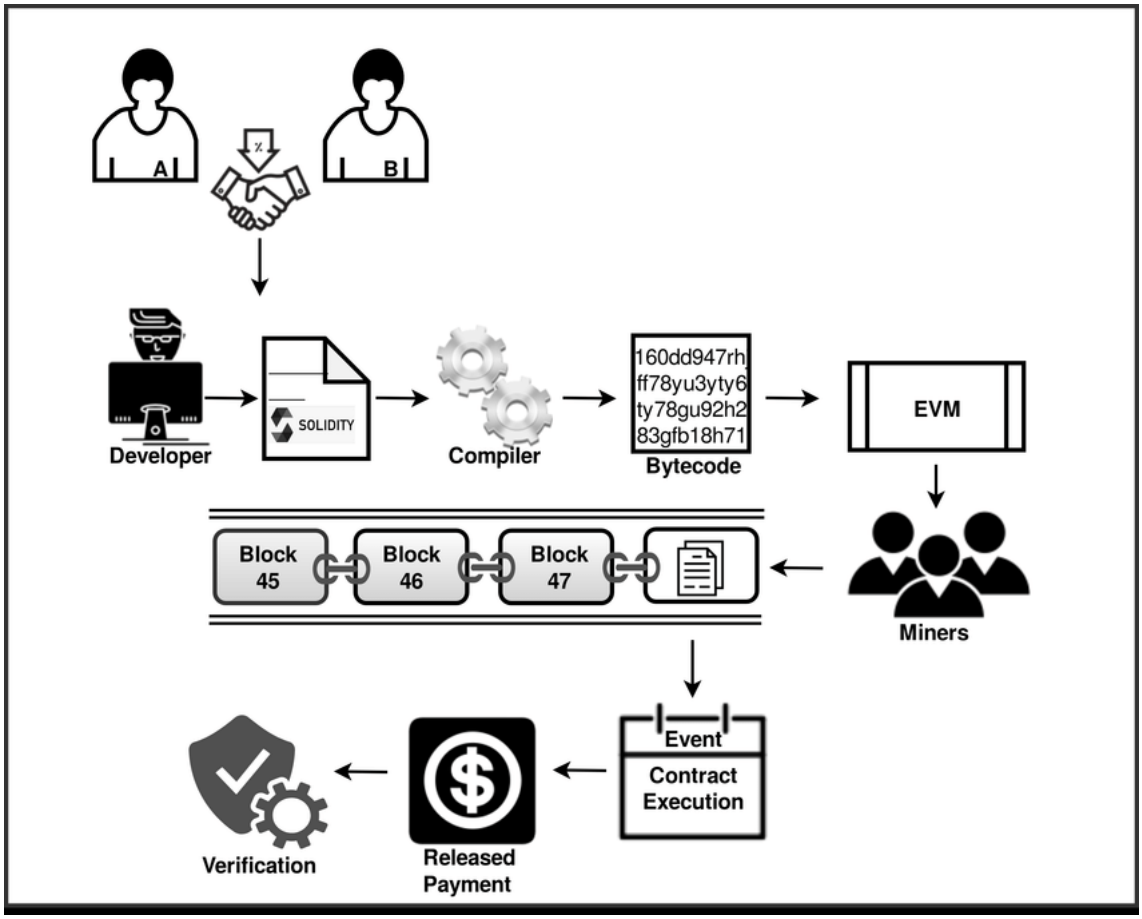
A ideia proposta por Szabo era converter as cláusulas de um contrato tradicional em código que seria incorporado ao hardware ou software fazendo cumprir os termos desse contrato, eliminando assim a necessidade de entidades terceirizadas e agências de certificadoras, porém, na época os contratos inteligentes eram apenas uma ideia em potencial e seriam extremamente difíceis de implementar.

A popularização dos contratos inteligentes só veio acontecer mais de 20 anos depois, com o lançamento da plataforma Ethereum em 2015, possibilitando que desenvolvedores escrevam contratos inteligentes em linguagens de alto nível, e os compilem para *bytecode* da *Ethereum virtual machine* (EVM), uma *Virtual machine* (VM) baseada em pilha operando em palavras de 256 bits, por intermédio da Solidity, uma das linguagem de programação mais utilizadas atualmente para este intuito.

Quando comparada a VMs de uso geral, como a máquina virtual Java, a EVM é relativamente simples, executa deterministicamente e suporta nativamente certas

primitivas criptográficas (BUTERIN, 2017).

Figura 4 – Ciclo de execução de um contrato inteligente



Fonte: (SAYEED *et al.*, 2020)

Na rede Ethereum um contrato é implantado transmitindo uma transação contendo seu *bytecode* e os parâmetros de inicialização. Assim que os mineradores da rede o incluem em um bloco, o contrato é armazenando permanentemente e recebe um endereço *blockchain* exclusivo.

Com este endereço em mãos os usuários podem interagir com o contrato transmitindo transações que contêm o endereço do contrato, a função a qual deve ser executada e seus argumentos, em alguns casos, o contrato pode chamar outros contratos e enviar unidades de ether, criptomoeda nativa do Ethereum, para usuários ou ainda outros contratos.

Para tornar o spam da rede oneroso, o protocolo Ethereum especifica um custo para cada operação na EVM, expressado em unidades de *gas*, em alusão ao combustível que permite que a rede opere. Desta forma um usuário deve pagar antecipadamente pela quantidade esperada de *gas* que a computação consumirá e após uma execução bem-sucedida receberá um reembolso parcial, caso necessário. Caso ocorra

uma exceção, todas as alterações de estado serão revertidas, mas o *gas* não poderá ser reembolsado. Existe ainda a possibilidade de uma transação falhar por um erro de *"Out of Gas"*, situação em que o montante de *gas* disponibilizado para a execução das operações não é suficiente e, portanto, a transação não se concretiza.

Desta forma o Ethereum permite com que pessoas e empresas ao redor do mundo codifiquem e garantam programaticamente a execução de acordos financeiros complexos sem a necessidade de uma entidade mediadora, abrindo um leque de possibilidades de novos modelos de negócios e mudando drasticamente a constituição da economia digital (TIKHOMIROV *et al.*, 2018).

3.4 PLATAFORMAS *LOW-CODE*

As plataformas de desenvolvimento *low-code* são plataformas baseadas em interface gráfica que possibilitam o desenvolvimento de software sem exigir amplo conhecimento de codificação, em contraposição às técnicas tradicionais de programação. Estas plataformas permitem com que o programador gaste menos tempo pensando na sintaxe do código e coloque mais ênfase no design da estética e funcionalidade do aplicativo, reduzindo assim o tempo gasto na implementação e solução de problemas (WASZKOWSKI, 2019).

Nos primórdios da programação, era comum programar em linguagens *assembly*, escrevendo instruções de código de máquina projetadas para arquiteturas de computador específicas, tais linguagens eram frequentemente chamadas de “baixo nível“ e costumavam ser um incômodo aos desenvolvedores, pois um determinado programa desenvolvido para uma máquina específica só seria executado em uma máquina do mesmo modelo.

Essas dificuldades levaram ao surgimentos das linguagens de programação de “alto nível“, como FORTRAN e, posteriormente, COBOL, que tem por objetivo abstrair os detalhes do hardware da máquina, usando uma sintaxe mais próxima da linguagem humana. Inicialmente, a programação de “alto nível“ foi recebida com ceticismo por causa de seus bugs, no entanto, com o passar do tempo, o paradigma de programação mudou, e a utilização de linguagens de programação de “alto nível“ culminou no aumento da produtividade dos desenvolvedores de software e facilitação da codificação (CRUZ *et al.*, 2021).

A tendência de simplificar o desenvolvimento de software continua até os dias atuais, o que tem feito o desenvolvimento de software *low-code* crescer em popularidade. As gigantes da tecnologia como Google, Microsoft, Oracle, Salesforce e Alibaba têm suas próprias plataformas de desenvolvimento de software *low-code* e de acordo com o Gartner, uma empresa de pesquisa global, plataformas *low-code* serão usadas em 65% do trabalho de desenvolvimento de aplicativos até 2024 (DUFFY, 2019).

3.5 TRABALHOS CORRELATOS

A tecnologia de *Smart Contracts* baseados em *blockchain* ainda é muito nova e isto ficou evidente durante a realização da revisão bibliográfica sistemática. Quando fala-se de plataformas de desenvolvimento de *Smart Contracts* através de interfaces gráficas são poucos os artigos que abordam esta temática de forma completa, por estes motivos optou-se por analisar artigos que abordassem cada uma das temáticas de forma separada, representando o real estado da arte.

Ao todo foram produzidos resumos de dois artigos, *Toward A Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD Styles* (LIAO *et al.*, 2017) e *Design Patterns for Gas Optimization in Ethereum* (MARCHESI *et al.*, 2020), além de uma dissertação de mestrado, Desenvolvimento De Um Ambiente De Apoio Ao Ensino De Algoritmos E Programação: Usando Blockly (ELI, 2017) , que foram peças centrais para produção deste trabalho.

3.5.1 Toward A Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD Styles

Produzido em Taiwan no ano de 2017, este artigo propõe uma plataforma de desenvolvimento de contratos inteligentes que suporta o Desenvolvimento Orientado a Comportamento, (Behavior-driven development) ou BDD, testes e implantação de contratos em blockchains baseadas em Ethereum. No artigo em questão os autores partem da premissa de que no cenário atual não existem ferramentas que possibilitem a verificação sistemática e integrada, além de não existirem mecanismos de teste que garantam a precisão dos contratos inteligentes desenvolvidos.

O trabalho começa por uma revisão histórica das tecnologias que compõem os contratos inteligentes, explica de forma breve a origem das *blockchain* e alerta que atualmente a maioria das tecnologias de *blockchain* ainda estão sob constante desenvolvimento e muitos dos problemas de engenharia de software de aplicativos de *blockchain* ainda não foram bem explorados e, portanto, construir um aplicativo baseado em *blockchain* que possua qualidade comercial normalmente é muito mais desafiador do que um aplicativo corporativo tradicional.

O artigo segue explicando que a sintaxe utilizada para a criação de contratos inteligentes varia dependendo do tipo de *blockchain* ao qual o contrato está atrelado e que consequentemente, em linguagens como *Solidity*, que possibilitam a criação de contratos mais complexos e sofisticados, a complexidade lógica por trás dos contratos acaba se tornando um desafio para os desenvolvedores, sendo que o problema mais crítico é o caso de um destes contratos não refletir de forma precisa e abrangente os requisitos de negócios.

Os autores abordam também o cenário das ferramentas para testes de contratos

inteligentes, as quais em sua maioria dependem de outras linguagens de programação mais populares e que se comunicam via *Application Programming Interface* (API) com os contratos. Um dos exemplos citados é a *Solidity*, onde o método de teste de unidade que é comumente usado é o Mocha em combinação com Web3.js e TestRPC.

Diante deste cenário, o artigo propõe uma plataforma de teste de integração automática, que teria suporte a contratos inteligentes escritos em *Solidity*, e toma como exemplo a troca de pontos de fidelidade para verificar a viabilidade do sistema desenvolvido. A plataforma forneceria serviços de teste de integração automatizados na forma web, integrando diversas ferramentas como Cucumber.js, Web3.js e Mocha, permitindo a redução da complexidade e dificuldade ao projetar, testar e melhorar a qualidade do contrato inteligente.

3.5.2 Design Patterns for Gas Optimization in Ethereum

Este artigo apresenta um conjunto de padrões de design e desenvolvimento para contratos inteligentes para a rede Ethereum desenvolvidos em *Solidity* com enfoque na economia de *gas*, unidade que mede a quantidade de esforço computacional necessário para executar operações específicas na rede Ethereum, desta forma tornando a execução do contrato mais rápida e barata.

Segundo os autores, até o momento poucos esforços foram feitos pela comunidade na coleta e classificação de padrões relacionados a esse assunto, e em sua maioria os materiais existentes estão publicados em blogs ou listas de discussão. Devido a estes motivos os autores coletaram 24 padrões de projetos os quais foram classificados em 5 grandes categorias que são provenientes da análise dos padrões coletados e da observação do contexto no qual eles são utilizados.

O artigo foi estruturado em 4 principais seções, sendo elas: Introdução, onde a problemática é apresentada de forma breve e sucinta, ambientando o leitor quanto ao atual estado dos *Smart Contracts* na *blockchain* Ethereum, Background, que fornece um breve histórico da *blockchain* Ethereum e o mecanismo de *gas* por meio do qual a execução de *Smart Contracts* é gerenciada, Padrões de Projeto de Economia de *Gas*, onde de fato são apresentados os padrões e dicas de projeto que são objetivo de estudo deste artigo e por fim a Conclusão, onde os autores ressaltam que a maioria dos princípios gerais de programação e otimização também se aplicam a *Solidity*, mas existem algumas peculiaridades nesta linguagem que a tornam mais desafiadora em termos de otimização de código, e que ao aplicar os padrões e conselhos apresentados os desenvolvedores podem mitigar esses problemas típicos do desenvolvimento de *blockchain*.

3.5.3 Desenvolvimento De Um Ambiente De Apoio Ao Ensino De Algoritmos E Programação: Usando Blockly

Esta dissertação de mestrado aborda a construção de um ambiente de ensino baseado nas Tecnologias da Informação e Comunicação, e no framework Blockly, de forma a permitir que os alunos do ensino superior tenham uma melhor experiência no desenvolvimento dos conceitos de lógica de programação e algoritmos.

Segundo o autor, o trabalho foi elaborado visando a análise das principais dificuldades encontradas pelos alunos na aprendizagem dos conceitos atrelados ao desenvolvimento de programas em instituições de ensino superior, ressaltando-se o fato de que os alunos que constituem o público-alvo do projeto são os alunos de áreas que possuem contato direto ou indireto com as competências de lógica e programação.

A dissertação foi estruturada em 5 capítulos, onde no primeiro capítulo são apresentados os conceitos basilares utilizados para a confecção do projeto. Já o segundo capítulo retrata os detalhes do contexto da pesquisa, com conceitos relacionados a *Science, Technology, Engineering e Mathematics* (STEM), lógica, índices de evasão e dificuldades evidenciados no estudo das competências do projeto. O terceiro capítulo apresenta a metodologia de construção do projeto, além de discutir questões técnicas e justificar os motivos de escolha das ferramentas utilizadas. Ao longo do quarto capítulo são demonstrados os resultados da pesquisa realizada, juntamente com a análise dos dados coletados, a fim de verificar se os objetivos traçados foram de fato atendidos, assim como expor os percentuais de receptividade da ferramenta. O último capítulo foi destinado para as considerações finais, onde o autor apresenta suas conclusões e recomendações para trabalhos futuros.

4 DESENVOLVIMENTO

4.1 ANÁLISE DE REQUISITOS

De acordo com Larman requisitos são uma descrição das necessidades ou dos desejos para um produto, onde o objetivo básico é identificar e documentar o que é realmente necessário, de forma a comunicar claramente essa informação tanto ao cliente quanto aos membros da equipe de desenvolvimento, sendo de suma importância a definição de requisitos de maneira não-ambígua, para que assim os riscos sejam identificados e não aconteçam surpresas durante o desenvolvimento do produto (LARMAN, 2000).

Os requisitos de um software podem ser classificados em Requisitos Funcionais (RF) ou Requisitos Não-Funcionais (RNF). Os Requisitos Funcionais são os responsáveis por definir funções e comportamentos do software, já os Requisitos Não-Funcionais dizem respeito à restrições de desenvolvimento, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade e padrões a serem seguidos.

4.1.1 Requisitos Funcionais

Para o levantamento dos RF foram levados em consideração os recursos mínimos necessários para o funcionamento da aplicação e as principais funcionalidades encontradas em outras plataformas *low-code*.

RF	Descrição
RF-001	O sistema deve permitir o usuário se autenticar com o Google, Github ou email.
RF-002	O sistema deve permitir o usuário se cadastrar.
RF-003	O sistema deve permitir a criação e edição de contratos inteligentes por meio de blocos.
RF-004	O sistema deve permitir salvar e excluir contratos.
RF-005	O sistema deve permitir a compilação de contratos.
RF-006	O sistema deve permitir implantar na rede Ethereum os contratos criados.
RF-007	O sistema deve permitir o usuário desfazer ações.
RF-008	O sistema deve permitir importar e exportar contratos.
RF-009	O sistema deve permitir o usuário listar exemplos de contratos.

Tabela 1 – Requisitos Funcionais da Plataforma

4.1.2 Requisitos Não-Funcionais

Para o levantamento dos RNF foram levadas em consideração as principais tecnologias utilizadas para o desenvolvimento de plataformas Web3 e o conhecimento prévio do autor.

RNF	**Descrição**
RNF-001	A ferramenta deve suportar os navegadores Google Chrome e Microsoft Edge.
RNF-002	A interface da aplicação deve ser desenvolvida utilizado o framework React JS.
RNF-003	Para a utilização da ferramenta deve ser necessário o acesso à rede mundial de computadores.
RNF-004	Para a implantação dos contratos através da ferramenta deve ser necessário a instalação da extensão de navegador MetaMask.
RNF-005	A plataforma deve ser desenvolvida utilizando como base a biblioteca Blockly.
RNF-006	A aplicação deve compilar contratos por meio de um *Web Worker*.

Tabela 2 – Requisitos Não-Funcionais da Plataforma

4.2 PROTOTIPAÇÃO

No campo de desenvolvimento de sistemas, protótipos são representações limitadas de um design, podendo ser usados como ferramenta de comunicação entre membros de uma equipe de desenvolvimento ou ainda puramente como meio para explorar ideias (ROSEMBERG *et al.*, 2008).

De maneira geral protótipos podem ser classificados em algumas categorias, sendo elas: protótipos de baixa fidelidade, que tem como foco a interação do usuário com o sistema, os componentes da interface e estrutura geral; protótipos de alta fidelidade, úteis para demonstrar padrões e guias de estilo, tendo em vista que representam uma imagem real do sistema; protótipos executáveis, que focam na navegação do sistema, possibilitando testar o uso da interface, normalmente envolvendo código em uma linguagem de programação.

Para este trabalho optou-se apenas pelo uso de protótipos de baixa fidelidade, levando em consideração o fato de poderem ser construídos de maneira ágil e prática, ao mesmo tempo que são capazes de proporcionar uma visão geral do sistema, demonstrando as possibilidades de navegação e quais atividades o sistema comporta.

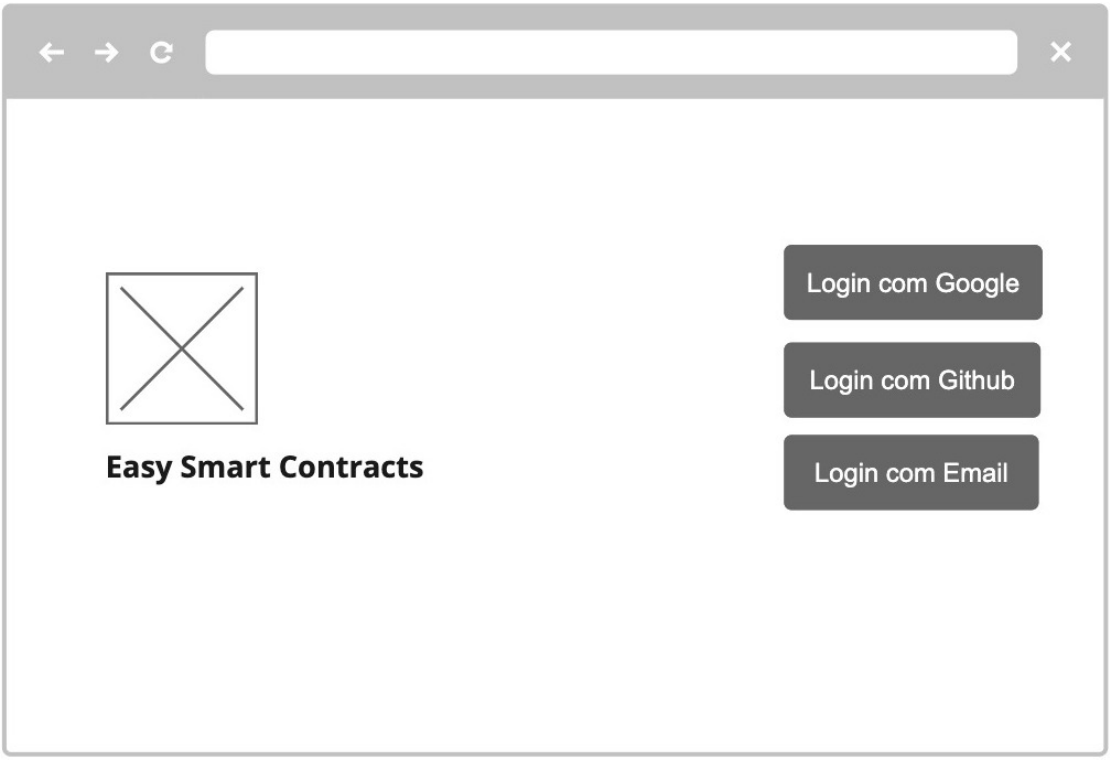
Durante a etapa de prototipação, foi utilizada a ferramenta Miro (2022), uma aplicação web que permite a criação de protótipos de baixa fidelidade através de *wireframes*.

Após uma série de iterações optou-se pelo uso do nome *Easy Smart Contracts* para a plataforma, com o intuito de enfatizar a facilidade com que o usuário pode desenvolver contratos inteligentes através da ferramenta.

4.2.1 Tela de login

Ao acessar o sistema pela primeira vez o usuário é direcionado para a tela de login, Figura 5, constituída por: no lado esquerdo um logo e o nome da aplicação e no lado direito três botões de autenticação.

Figura 5 – Tela de login



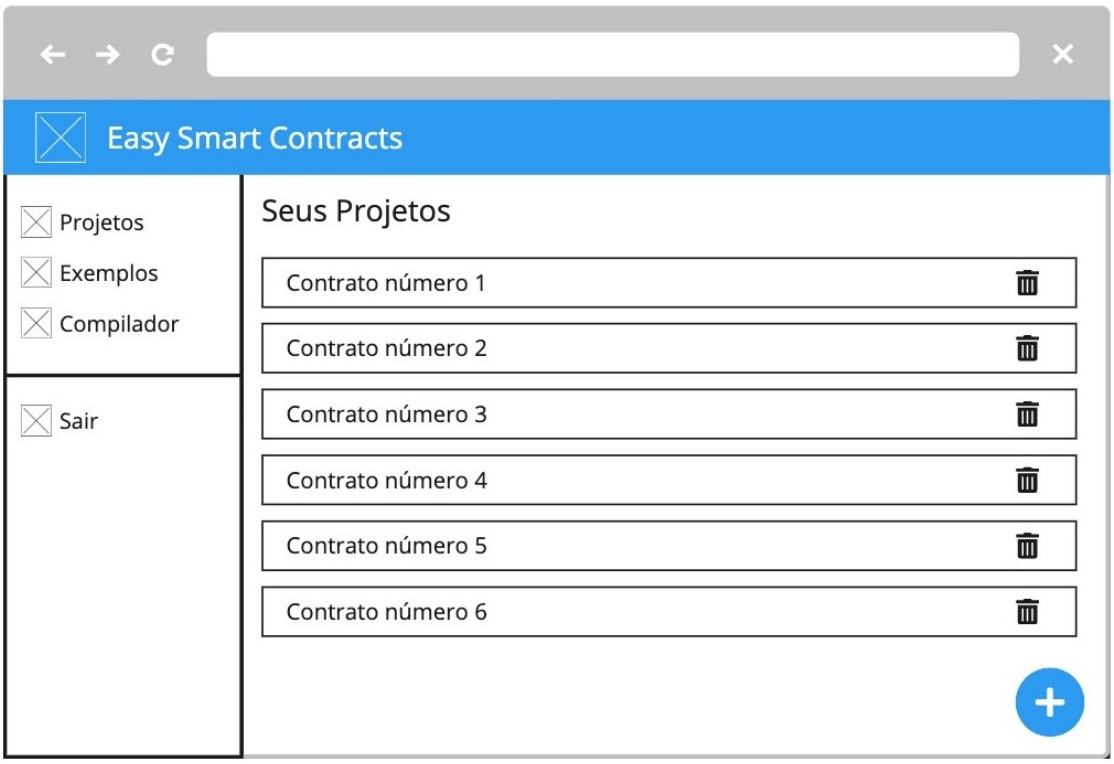
Fonte: do Autor

Interagindo com os botões na lateral direita o usuário pode se autenticar através do Google, Github ou email, cada um deles possuindo um fluxo próprio, lidando com estados de carregamento e eventuais mensagens de erro, uma vez autenticado o usuário é redirecionado a tela de gerenciamento dos projetos 4.2.2. Na eventualidade do usuário ainda não possuir uma conta, esta será criada automaticamente.

4.2.2 Tela de gerenciamento dos projetos

A tela de gerenciamento dos projetos é a rota inicial para usuários autenticados, e é por meio dela que os usuários podem listar, criar, deletar e importar projetos de contratos inteligentes.

Figura 6 – Tela de gerenciamento dos projetos



Fonte: do Autor

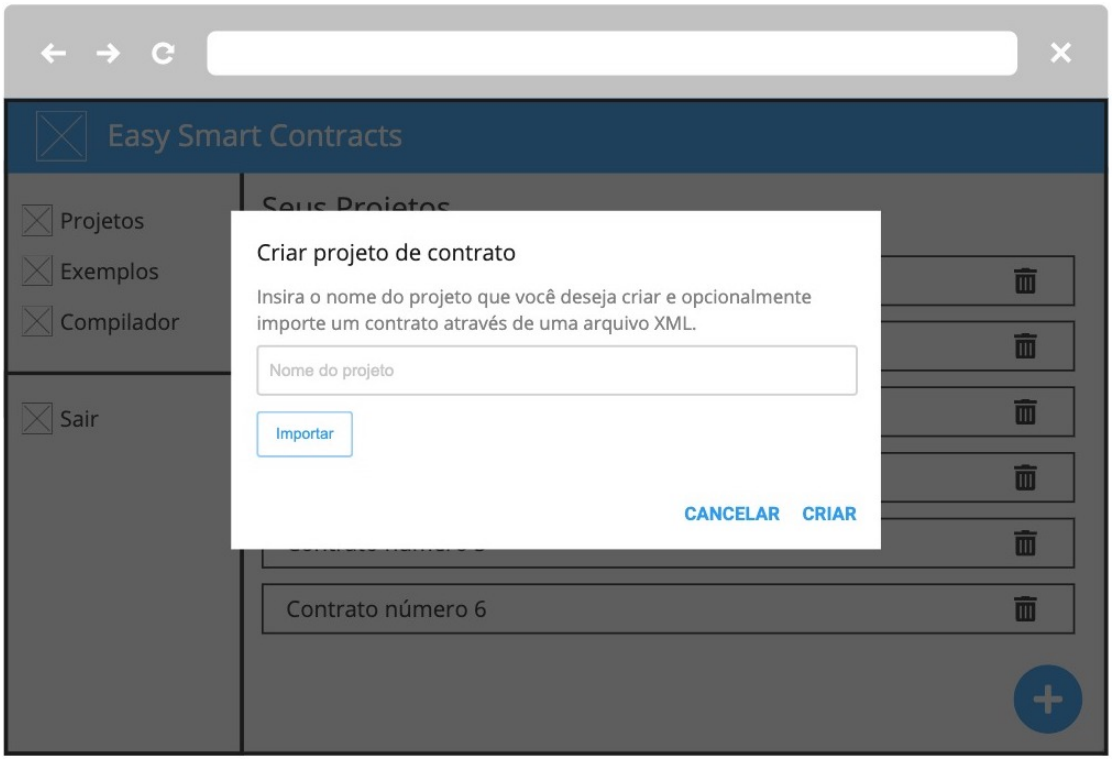
Como pode-se observar na Figura 6, esta tela é composta por três principais seções: um cabeçalho, um menu navegacional na lateral esquerda e a lista dos projetos do usuário ao centro.

O cabeçalho e o menu navegacional são os mesmos apresentados em todas telas, com exceção da tela de login 4.2.1. É através do menu na lateral esquerda que o usuário pode navegar para as outras telas da aplicação, como a tela do compilador 4.2.4, ou a tela de exemplos 4.2.5. Além disso este menu permite que o usuário saia de sua conta, redirecionando-o para a tela de login 4.2.1.

Já o conteúdo principal está localizado ao centro da tela, contendo um título, a lista dos projetos que o usuário possui e um botão para adição de projetos no canto inferior direito.

Cada item da lista conta com o nome do projeto e um ícone de lixeira, ao clicar em um dos itens o usuário é redirecionado para a tela de edição do projeto selecionado 4.2.3 e ao clicar no ícone de lixeira o usuário pode deletar seu projeto.

Figura 7 – Modal de criação de projeto



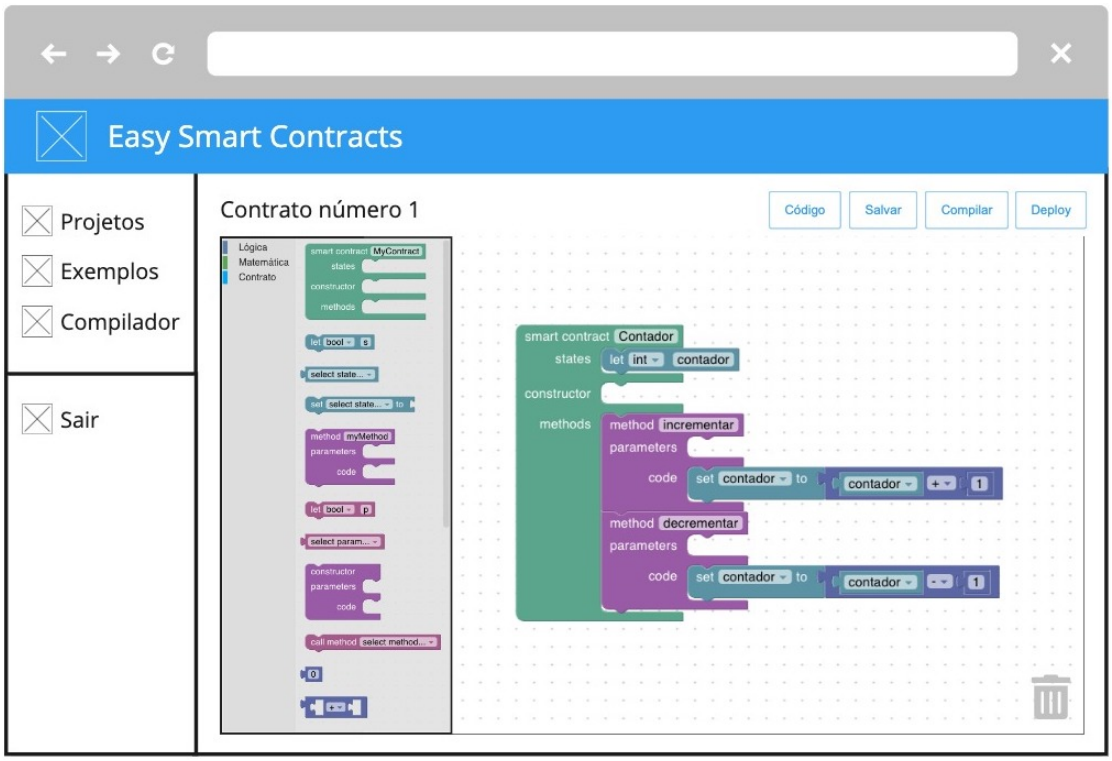
Fonte: do Autor

Clicando no botão no canto inferior direito uma modal é aberta, como pode-se observar pela Figura 7, por onde o usuário pode escolher o nome do projeto que deseja criar e opcionalmente importar um arquivo de um projeto criado anteriormente, ao clicar no botão de criar o projeto é automaticamente adicionado a lista principal.

4.2.3 Tela de edição do projeto

A tela de edição do projeto é a principal tela do sistema, é por meio dela que os usuários podem desenvolver, salvar, compilar e implantar seus contratos inteligentes na rede Ethereum.

Figura 8 – Tela do Editor



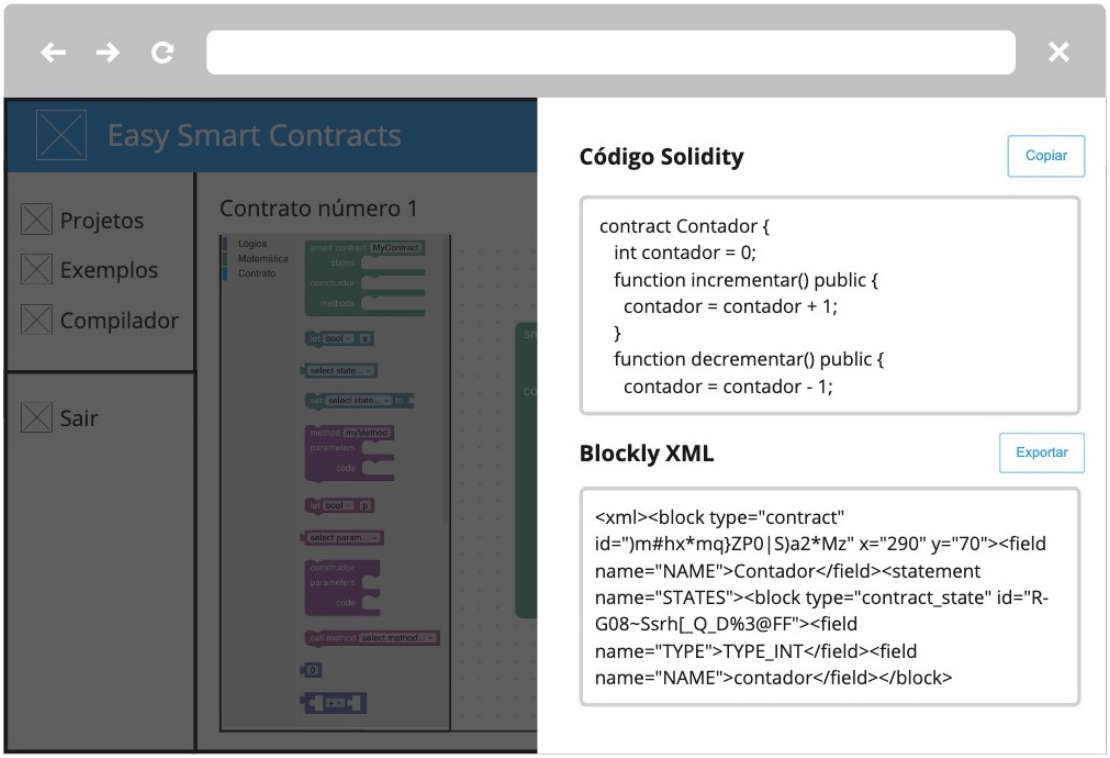
Fonte: do Autor

Como é possível observar na Figura 8, esta tela é composta por duas seções principais: um cabeçalho que mostra o nome do projeto que está sendo editado juntamente com quatro botões de ação e o editor, que é por onde o usuário pode desenvolver seu contrato inteligente utilizando blocos interligados que representam código Solidity.

O Editor possui um menu interno que permite a seleção de diversos blocos de código organizados em categorias, sendo elas, operações lógicas, operações matemáticas e operações relacionadas à entidade do contrato.

Uma vez que o usuário tenha desenvolvido seu contrato ele pode clicar no botão “código“ para revelar uma modal contendo o código Solidity que foi produzido e o XML utilizado pela biblioteca Blockly para renderizar os blocos, como mostra a Figura 9.

Figura 9 – Modal de inspeção de código



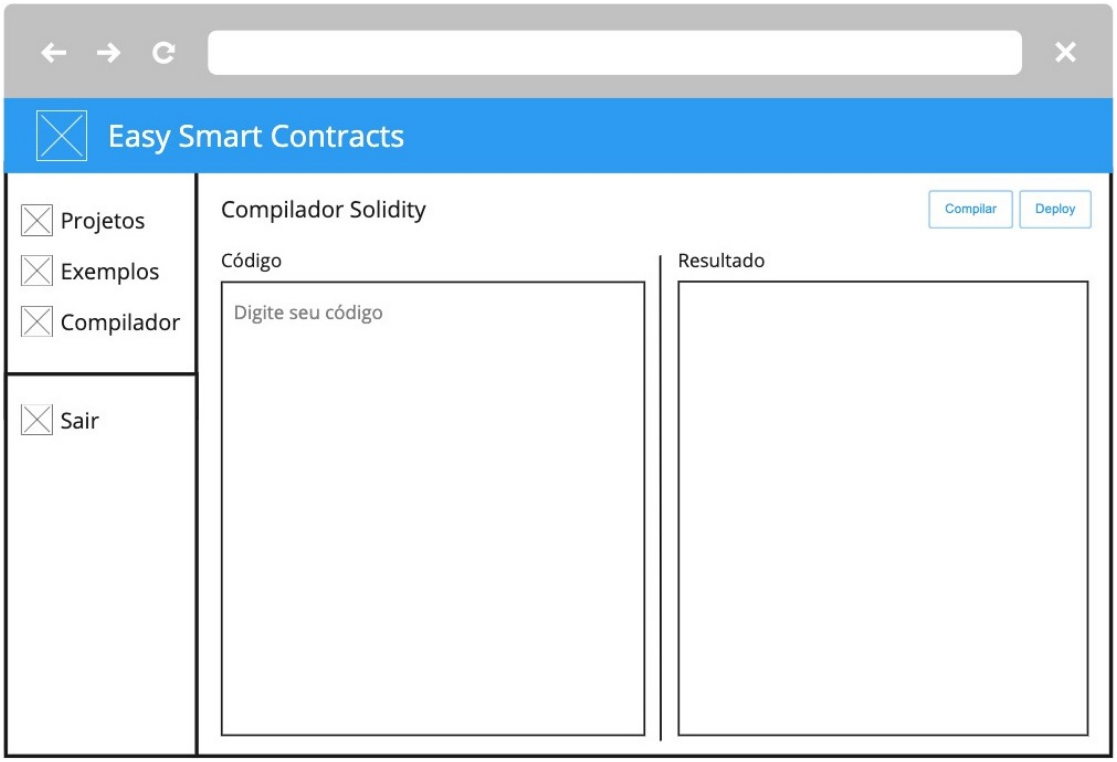
Fonte: do Autor

Por meio dos botões localizados na parte superior do editor o usuário pode também salvar as alterações feitas no projeto, compilar o contrato desenvolvido, verificando eventuais erros que possam ter sido cometidos, assim como implementar o contrato na rede Ethereum através do MetaMask.

4.2.4 Tela do compilador

A Tela do compilador foca em usuários com um perfil mais avançado, que já tenham algum conhecimento a respeito de programação orientada a objetos e que desejam ter um maior controle sobre as linhas de código Solidity utilizados em seus contratos inteligentes.

Figura 10 – Tela do compilador



Fonte: do Autor

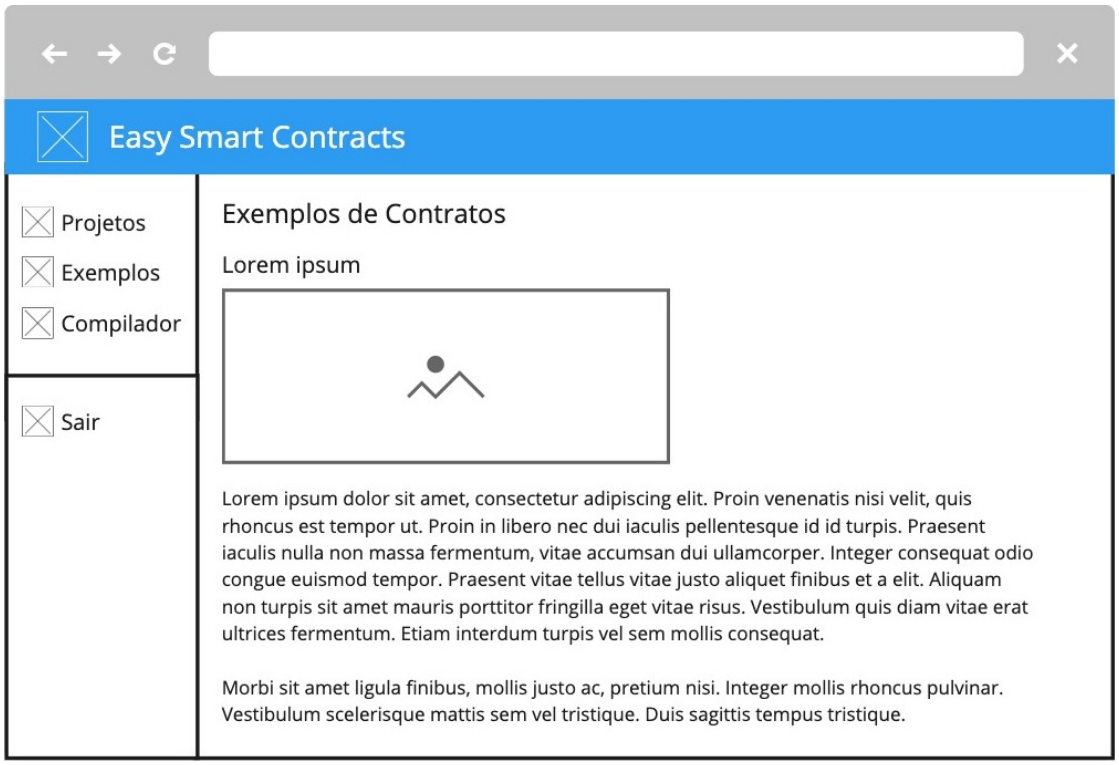
A interface desta tela é constituída por duas principais partes: um campo de texto a esquerda, onde o usuário pode escrever o código do contrato inteligente a ser criado e um bloco não editável a direita, no qual são mostrados os resultados detalhados da compilação dos contratos, como mostra a Figura 10.

Esta tela conta também com dois botões localizados no canto superior direito, que permitem ao usuário compilar e implantar contratos inteligentes na rede Ethereum. A principal vantagem do uso desta interface é a possibilidade do desenvolvimento de contratos que utilizem diretivas mais avançadas não suportadas pelo Editor de blocos. Além disso, a interface com acesso direto ao compilador serve como porta de entrada para o momento em que o usuário queira começar a escrever seus contratos diretamente em Solidity.

4.2.5 Tela de exemplos

A tela de exemplos tem um propósito puramente educativo e é direcionada a usuários que nunca tiveram o contato com contratos inteligentes, permitindo-os visualizar exemplos de contratos desenvolvidos dentro da plataforma e aprender sobre o desenvolvimento de código Solidity através de blocos interligados.

Figura 11 – Tela de exemplos



Fonte: do Autor

A interface demonstrada na Figura 11 permite também que o usuário crie projetos de contratos inteligentes com o código demonstrado em cada um dos exemplos, tornando, desta forma, a experiência mais imersiva.

4.3 TECNOLOGIAS E FERRAMENTAS

Nesta seção são descritas as principais tecnologias e ferramentas que foram julgadas mais adequados para o desenvolvimento da Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica.

4.3.1 Typescript

O TypeScript é uma linguagem de software livre desenvolvida pela Microsoft. Ela é um superconjunto do JavaScript, o que significa que você pode continuar a usar as habilidades de JavaScript que já desenvolveu e adicionar determinados recursos que não estavam disponíveis para você anteriormente.

4.3.2 React

O React é uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces com o usuário, que permite compor interfaces do usuário complexas a partir de pequenos e isolados códigos chamados “componentes”.

4.3.3 Blockly

Blockly é uma biblioteca que adiciona um editor de código visual a aplicativos da Web e móveis. O editor Blockly usa blocos gráficos interligados para representar conceitos de código como variáveis, expressões lógicas, loops e muito mais, permitindo que os usuários apliquem os princípios de programação sem ter que se preocupar com a sintaxe ou a intimidação de um cursor piscando na linha de comando.

4.3.4 Web Workers

Web Workers são mecanismos que permitem que uma operação de um dado script seja executado em uma *thread* diferente da *thread* principal da aplicação Web. Permitindo com que operações computacionalmente pesadas sejam processadas sem que ocorra bloqueio da *thread* principal, geralmente responsável pelos elementos da interface.

4.3.5 Material UI

Material UI é uma biblioteca de código aberto de componentes React que implementa o *Material Design* do Google. Ela inclui uma vasta coleção de componentes pré-construídos com opções de personalização que facilitam a implementação de sistema. Além disso, a acessibilidade é uma das maiores prioridades para biblioteca e por isso, todos seus componentes buscam atender aos mais altos padrões de forma e função.

4.3.6 HTML

HyperText Markup Language (HTML) é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web. Outras tecnologias além do HTML geralmente são usadas para descrever a aparência/apresentação (CSS) ou a funcionalidade/comportamento (JavaScript) de uma página da web.

"Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. Links são um aspecto fundamental da web. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, você se torna um participante ativo na world wide web.

4.3.7 CSS

Cascading Style Sheets (CSS) é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML (incluindo várias linguagens em XML como SVG, MathML ou XHTML). O CSS descreve como elementos são mostrados na tela, no papel, na fala ou em outras mídias.

4.3.8 MetaMask

MetaMask é uma extensão de navegador de código aberto que permite interações e experiências do usuário na Web3. Ela foi criada para atender às necessidades de sites baseados em Ethereum e possibilitar o uso de DApps no navegador. Em particular, ela lida com o gerenciamento de contas e conecta os usuário a *blockchain*.

4.3.9 Firebase

O Firebase é uma plataforma de desenvolvimento do Google composta por um conjunto de serviços que tem por intuito facilitar o desenvolvimento de aplicativos web ou móveis. Dentre os serviços mais relevantes pode-se listar o *Firebase Authentication*, *Cloud Firestore* e o *Firebase Hosting*.

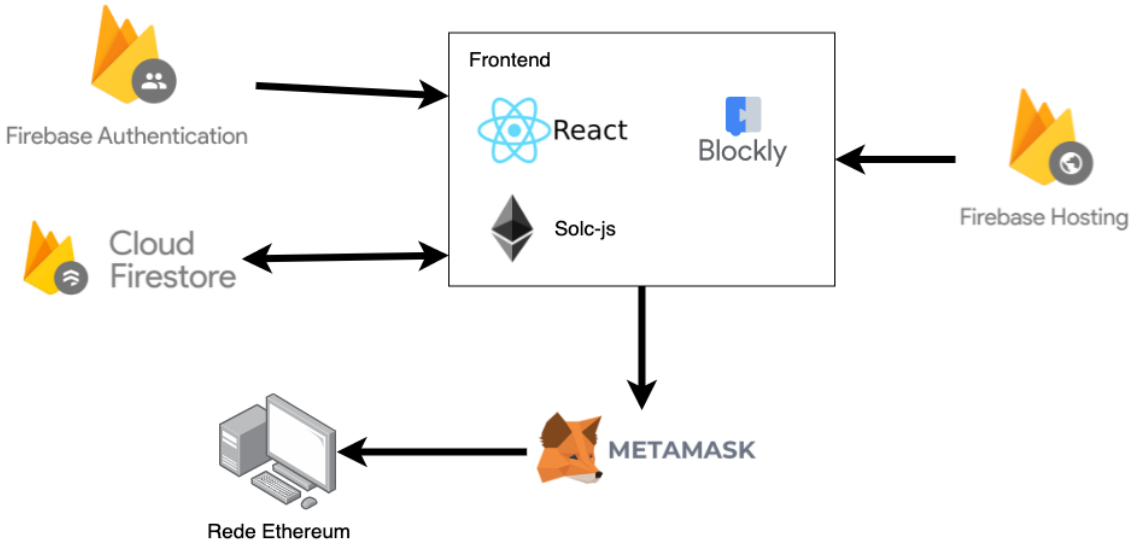
4.3.10 Solc-js

O solc-js é uma biblioteca JavaScript que encapsula o compilador Solidity e possibilita que contratos sejam compilados para *bytecode* da EVM diretamente no browser, eliminando assim a necessidade de um servidor específico para realizar tal operação.

4.4 ARQUITETURA

A arquitetura do sistema foi projetada de forma a dar ênfase ao objeto de estudo do presente trabalho, os contratos inteligentes e as plataformas de desenvolvimento *low-code*. Por este motivo optou-se pelo uso de serviços como o *Firebase Authentication*, *Cloud Firestore* e o *Firebase Hosting*, que facilitam o gerenciamento da autenticação de usuários, o uso de bancos de dados flexíveis e escalonáveis para desenvolvimento focado em dispositivos Web e a hospedagem de conteúdo na Web, respectivamente.

Figura 12 – Arquitetura do sistema



Fonte: do Autor

Em relação ao desenvolvimento da interface da plataforma optou-se pelo uso do TypeScript, por ser uma linguagem de programação amplamente usada no desenvolvimento de sistemas e serviços web, além de possuir uma ótima integração com a biblioteca React. Como base para a plataforma *low-code* foi definida a utilização da biblioteca Blockly, pois esta já inclui as principais mecânicas esperadas de uma plataforma *low-code*, como blocos gráficos interligados que representam conceitos de código, bastando com que seja adicionado o suporte a linguagem de programação Solidity. Além disso, optou-se pelo uso da biblioteca solc-js para a compilação do código Solidity em *bytecode* e da integração com a extensão MetaMask para a implantação dos contratos inteligentes da rede Ethereum.

4.4.1 Modelagem do banco de dados

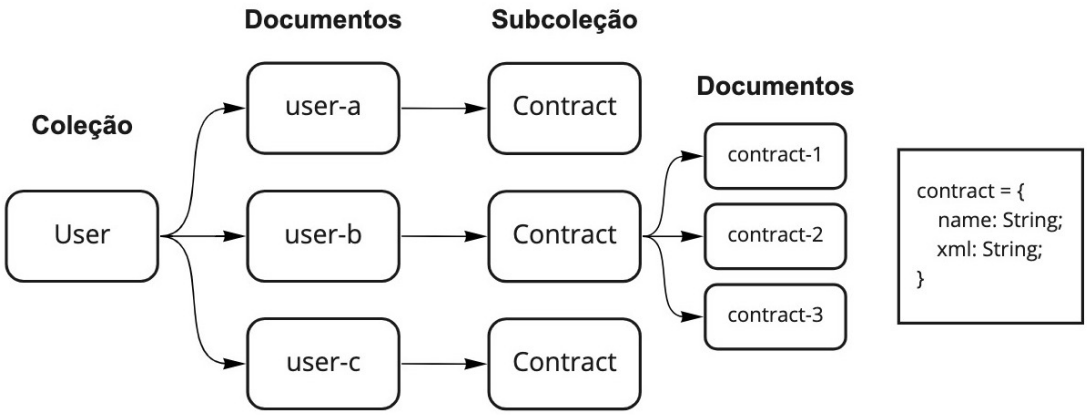
Ao optar-se pelo uso de um banco de dados *NoSQL* baseado em documentos como o *Cloud Firestore*, tem-se algumas vantagens frente a bancos de dados mais tradicionais.

Os bancos de dados baseados em documentos costumam trabalhar com documentos *JavaScript Object Notation* (JSON) ou semelhantes, armazenando dados em pares chave-valor. Em cada um dos documentos as chaves devem ser exclusivas e todo documento contém uma chave de identificação única dentro de uma coleção de documentos, que identifica um documento explicitamente. Desta forma estruturas de dados complexas, como objetos aninhados, podem ser manipuladas de forma mais conveniente (HECHT; JABLONSKI, 2011).

Outra característica de bancos de dados baseados em documentos é a não existência de restrições de esquema, o que torna possível o armazenamento de novos documentos com qualquer tipo de atributo, podendo ser feito tão facilmente quanto adicionar novos atributos a documentos existentes em tempo de execução.

Levando em consideração estes fatores e os RF definidos no começo do capítulo, o banco de dados da aplicação foi projetado de maneira simples, possuindo apenas uma principal coleção, chamada *User*, responsável por armazenar os dados de todos os usuários, como demonstra a Figura 13.

Figura 13 – Diagrama da estrutura dos dados no *Cloud Firestore*



Fonte: do Autor

Cada documento pertencente à coleção *User* representa um usuário cadastrado na plataforma e por sua vez, possui uma subcoleção chamada *Contract*, que armazena quaisquer documentos de projetos de contratos inteligentes que o usuário possua. Todos os documentos dentro da subcoleção *Contract* seguem uma estrutura composta por dois campos: *name*, que representa o nome que o usuário deu ao projeto durante

a etapa de criação e o campo *xml* que guarda em forma de texto a posição e a as interações de cada bloco de código do editor utilizado no projeto.

4.5 IMPLEMENTAÇÃO

Esta seção relata alguns dos pontos que se destacaram durante a etapa de implementação da Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica.

4.5.1 *Cloud Firestore*, *Firebase Authentication* e *Firebase Hosting*

A plataforma de desenvolvimento *Firebase* trabalha com um modelo conhecido como *Backend as a Service* (BaaS) e fornece um conjunto de ferramentas e serviços prontas para o uso, possibilitando ao desenvolvedor criar aplicações completas sem a necessidade de arquitetar sua própria infraestrutura, com servidores, *load balancers*, *proxies*, etc, ou ainda se preocupar com a escalabilidade da aplicação.

Para o uso do *Firebase* é necessário optar entre um de seus dois planos: o Plano Spark ou o Plano Blaze. O Plano Blaze é voltado para aplicações de médio e grande porte, nele o usuário paga um valor baseado nas taxas de consumo de cada um dos serviços disponíveis, como por exemplo o número de leituras de documentos no banco de dados. Já o Plano Spark é totalmente gratuito e é comumente usado por startups e protótipos, tendo em vista que disponibiliza limites generosos de seus serviços sem custo algum.

Para o desenvolvimento da aplicação optou-se pelo uso do Plano Spark, levando em consideração o fato de ser gratuito e de possuir ótimos limites do serviço *Cloud Firestore*, permitindo o armazenamento de até 1 GiB de dados, 20 mil gravações e exclusões ao dia e 50 mil leituras diárias de documentos (FIREBASE, 2022), limite que ultrapassa em muito o consumo previsto no tocante ao número de usuários que farão uso da plataforma neste primeiro momento.

A plataforma desenvolvida no presente trabalho faz o uso de apenas três dos serviços disponíveis: *Cloud Firestore*, um banco de dados *NoSQL* baseado em documentos; *Firebase Authentication*, um serviço de autenticação com interface do usuário localizada, suporte a autenticação através de diversas plataformas como Google, Facebook e Twitter e integração nativa com o *Cloud Firestore* além do *Firebase Hosting*, um serviço de hospedagem de sites dinâmicos e estáticos oferecendo domínios customizados com certificados SSL.

Em relação a implantação do *Cloud Firestore* na plataforma foi necessário definir um conjunto de permissões para restringir o acesso dos usuários a documentos e coleções no banco de dados, tendo em vista que a interface do cliente possui acesso direto ao *Software Development Kit* (SDK) do *Cloud Firestore* e pode executar qualquer

ação caso as permissões não tenham sido definidas.

Figura 14 – Regras de segurança do *Cloud Firestore*

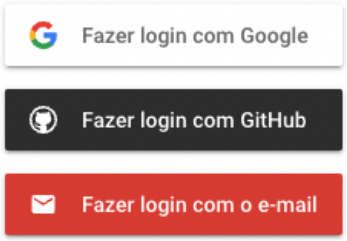
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /user/{userId}/{document=**} {
      allow read,write: if request.auth.uid == userId;
    }
  }
}
```

Fonte: do Autor

Conforme mostra a Figura 14, as regras estabelecidas pelo painel do *Cloud Firestore* permitem que apenas usuários autenticados tenham acesso ao seu próprio diretório, de forma a permitir a leitura e a escrita do documento e subcoleções dentro da coleção *user* que possuir o mesmo identificador que o usuário autenticado.

A plataforma desenvolvida utilizou também o *Firebase Authentication*, pois este permite a fácil integração com provedores de autenticação de terceiros. Foram configurados a autenticação por meio de contas do Google, Github e contras de e-mail como previsto nos RFs definidos em 4.1.1.

Figura 15 – Interface de autenticação



Fonte: do Autor

Em relação a interface de autenticação, foi utilizada a biblioteca *FirebaseUI*[1], disponibilizada pelo próprio *Firebase*, que implementa as melhores práticas recomendadas para autenticação em dispositivos móveis e sites, ajudando na conversão de login e no cadastro de novas contas. A biblioteca lida também com *edge cases*, tais como, a recuperação de contas e a vinculação automática de contas que possuam o mesmo endereço de email.

[1] https://github.com/firebase/firebaseui-web

Para o uso do *Firebase Hosting* foi necessário a instalação da *Command-line interface* (CLI) do *Firebase*, ferramenta que permitiu a implantação de arquivos de diretórios locais nos servidores de *Hosting* do *Firebase*.

Por padrão todos os projetos do *Firebase Hosting* têm direito a utilização de subdomínios nos domínios web.app e firebaseapp.com sem custos financeiros, motivo pelo qual as versões de teste da plataforma desenvolvida foram distribuídas por meio do endereço web `https://easy-smart-contracts.web.app/`.

4.5.2 Suporte da linguagem Solidity a biblioteca Blockly

Blockly é uma biblioteca de código aberto desenvolvida pelo Google que possibilita a adição de um editor de código *low-code* a aplicações web. Através de blocos interligados, o editor Blockly permite a representação de diversos conceitos de código como variáveis, expressões lógicas entre outros, o que possibilita que o usuário construa seu código de maneira intuitiva e visual, sem ter que se preocupar com detalhes de sintaxe.

Do ponto de vista de um desenvolvedor, o Blockly é uma interface de usuário pronta para criar uma linguagem visual, capaz de emitir códigos gerados pelos usuários que sejam sempre sintaticamente corretos. (BLOCKLY, 2022)

Por padrão o Blockly exporta blocos para diversas linguagens de programação como JavaScript, Python, PHP, Lua e Dart, no entanto, a biblioteca não possui suporte nativo a linguagem de programação Solidity e, portanto, esta teve que ser adicionada manualmente.

O Blockly é uma biblioteca muito flexível e permite a extensão do suporte a outras linguagens de programação por meio de “geradores“. Para cada gerador de linguagem que tenha sido criado é necessário também a adição de geradores de código de bloco, que devem ser desenvolvidos para cada bloco que esta nova linguagem suportar.

Um gerador de uma linguagem define as propriedades básicas da linguagem, como o por exemplo o funcionamento da indentação, já os geradores de blocos definem como os blocos individuais são transformados em código e devem ser definidos para cada um dos blocos disponíveis ao usuário por meio do editor.

A linguagem de programação Solidity é uma linguagem de alto nível orientada a objetos utilizada para a implementação de contratos inteligentes, que nada mais são do que programas que regem o comportamento das contas existentes na *blockchain* do Ethereum.

A Solidity é uma linguagem de colchetes projetada para gerar código para a EVM e que foi influenciada e inspirada em várias linguagens de programação conhecidas, sendo mais profundamente influenciada por C++, porém, também emprega conceitos de linguagens como JavaScript, Python e outras (ETHEREUM, 2022).

Figura 16 – Exemplo de código desenvolvido em Solidity

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.11;
3
4  contract Greeting {
5      string public name;
6      string public greetingPrefix = "Hello ";
7
8
9      constructor(string memory initName){
10         name = initName;
11     }
12
13     function setName(string memory newName) public {
14         name = newName;
15     }
16
17     function getGreeting() public view returns (string memory) {
18         return string(abi.encodePacked(greetingPrefix, name));
19     }
20 }
21
```

Fonte: do Autor

A influência da linguagem C++ é nítida, como mostra a Figura 16, podendo ser observada na sintaxe para declarações de variáveis, na utilização de laços, no conceito de sobrecarga de funções, conversões de tipos implícitos e explícitos e em muitos outros detalhes.

Além disso, a Solidity conta com uma ótima documentação em relação ao funcionamento da linguagem. Ela contém explicações a respeito da ordem de precedência de cada um dos seus operadores, detalha todas as variáveis globais suportadas, lista palavras reservadas, enumera seus modificadores, explica como funcionam os especificadores de visibilidade das funções, entre outros. O que foi essencial para a adição do suporte da Solidity ao Blockly.

Da mesma forma a biblioteca Blockly também dispõe de uma boa documentação, onde disponibiliza através do site Blockly Codelabs[2] uma série de tutoriais a respeito de como customizar a biblioteca, dando detalhes da criação de geradores customizados, de como personalizar a *toolbox*, entre outros.

Para adicionar o suporte da linguagem Solidity foi necessário a criação de um gerador de linguagem e diversos geradores de blocos de código. Durante a etapa de desenvolvimento foram utilizados como referência os geradores de linguagens já suportadas, em especial o da linguagem JavaScript.

Isso posto, foram implementados o suporte a algumas diretivas de linguagem Solidity para a biblioteca Blockly, entre elas: a declaração de entidades do tipo *contract*,

[2] https://blocklycodelabs.dev/codelabs/

incluindo o uso de construtores e métodos; estruturas de condição *if then else*; suporte a operações matemáticas de soma, subtração, multiplicação e divisão; suporte aos tipos *bool*, *int* e *uint*, além do suporte a declaração e atualização de variáveis e estados.

4.5.3 Compilador Solidity para Web

Para que se possa implantar um contrato inteligente é necessário primeiramente transformar o código Solidity para *bytecode* da EVM. Esta tarefa é realizada através de um compilador, que no caso da linguagem de programação Solidity se chama Solc.

Solc é originalmente um compilador desenvolvido em C++, porém, no campo das aplicações web ele é mais comumente encontrado em sua versão JavaScript chamada solc-js, uma vez que os navegadores não possuem suporte para código em C++.

O projeto solc-js nada mais é que uma derivação do compilador Solc utilizando Emscripten, desta forma ambos utilizam o mesmo código-fonte do compilador e tem suporte às mesmas funcionalidades. Emscripten é um compilador de código fonte C e C++ para WebAssembly que utiliza LLVM e Binaryen e possibilita que aplicações e bibliotecas escritas em linguagens C e C++ sejam compiladas antecipadamente e executadas de forma eficiente em navegadores da web e ambientes Node.js.

No entanto a tecnologia WebAssembly possui algumas limitações ao rodar na *thread* principal dos navegadores, dado que em navegadores baseados em chromium a diretiva *WebAssembly.Compile* pode apenas ser usada com buffers menores de 4KB, o que efetivamente impede que a plataforma desenvolvida no presente trabalho carregue o solc-js na *thread* principal.

Para contornar este problema foi necessário a utilização de WebWorkers, mecanismos que permitem que um dado script seja executado em uma *thread* diferente da *thread* principal da aplicação.

A *thread* principal dos navegadores e as *threads* dos WebWorkers comunicam-se através de um sistema de mensagens que permite a troca de dados entre as diferentes *threads*, ambos os lados podem enviar mensagens usando o método *postMessage()* e respondem as mensagens através do manipulador de eventos *onmessage* (MDN, 2022). É importante ressaltar que os dados transmitidos entre as *threads* são copiados e não compartilhados, de maneira a evitar o acesso indevido de memória por ambas as partes.

Uma das principais vantagens ao fazer o uso de WebWorkers para a execução do solc-js no sistema desenvolvido consiste no fato de não ocorrer o bloqueio da interface do usuário no sistema durante a compilação de contratos, o que possibilita que estados de carregamento sejam mostrados ao usuário e assegura uma boa usabilidade.

A documentação da linguagem de programação Solidity recomenda que sempre

sejam usadas as versões mais recentes da linguagem para a compilação e implantação de contratos, tendo em vista que a cada nova versão bugs são corrigidos e novos recursos são lançados.

As versões da Solidity seguem as especificações do Versionamento Semântico[3], sendo que as versões de nível de *patch* com versão principal 0, ou seja, 0.x.y, não conterão *breaking changes*, o que significa que um código que compila com a versão 0.x.y sempre poderá ser compilado com 0.x.z, onde z > y.

Levando todos estes fatores em consideração a versão escolhida para o compilador foi a 0.8.6, por ser a versão mais atual no momento da elaboração do projeto desenvolvido no presente trabalho.

4.5.4 Web3 e a implantação de contratos inteligentes

Web3, também conhecida como Web 3.0 ou web descentralizada, foi um termo cunhado por Gavin Wood que propõe o fornecimento de serviços de internet distribuídos, sem a necessidade de agentes terceiros confiáveis, desta maneira oferecendo aos usuários um maior controle sobre seus dados.

Nos últimos anos, o termo Web3 tem feito grande sucesso no mercado de criptomoedas e nas comunidades relacionadas à *blockchain*, tornando-se o termo mais prevalente do setor. (WANG *et al.*, 2022)

O princípio básico por trás de aplicações que utilizam Web3 é que os usuários podem manter o controle total sobre seus dados, ao invés de serem gerenciados por organizações centralizadas, como ocorre na Web1 e Web2. A Web3 move os dados para longe dessas autoridades centrais e da ênfase na descentralização, estabelecendo aplicativos e serviços em torno das blockchains.

Para que um software possa interagir com a blockchain *Ethereum*, seja lendo dados ou enviando transações para a rede, ele deve primeiro se conectar a um nodo *Ethereum*. Para que isso seja possível, todo cliente *Ethereum* implementa uma especificação JSON-RPC[4], fazendo com que exista um conjunto uniforme de métodos padronizados nos quais os aplicativos podem confiar.

O JSON-RPC é um protocolo de chamada de procedimento remoto, leve, sem estado e que define várias estruturas de dados e as regras em torno de seu processamento. O protocolo é agnóstico de transporte, e utiliza JSON como formato de dados. (WACKEROW, 2022)

A plataforma desenvolvida neste trabalho faz o uso da biblioteca web3.js, que é a API *JavaScript* do *Ethereum* que se conecta ao protocolo JSON-RPC. É por meio desta biblioteca que a aplicação interage com o ecossistema *Ethereum* e por onde os contratos inteligentes são implantados.

[3] https://semver.org/lang/pt-BR/

[4] https://www.jsonrpc.org/specification

Tendo em vista que o *Ethereum* é na verdade um protocolo, podem haver várias redes independentes que estejam em conformidade com ele, cada uma contendo sua própria *blockchain*, o que efetivamente permite a existência de vários ambientes, como ambientes de desenvolvimento local, de testes e produção.

A rede chamada *Mainnet* é a principal rede de produção pública do *Ethereum*, sendo essa a *blockchain* que possui real valor monetário. Já as *Testnets* são redes de teste públicas que não tem valor real atrelado e são usadas por desenvolvedores para testar contratos inteligentes antes de serem implantados na *Mainnet*.

Durante os testes da aplicação desenvolvida foi utilizada a rede Rinkeby, uma *Testnet* que usa o conceito de *proof-of-authority* como mecanismo de consenso, onde um pequeno número de nodos é escolhido para validar transações e criar novos blocos. A rede Rinkeby dispõe de sites como o *Rinkeby Faucet*[5] que distribuem de maneira gratuita algumas frações de Ether diariamente, possibilitando que desenvolvedores testem seus contratos inteligentes sem custos.

Para que os contratos possam ser implantados através da plataforma é necessário que o usuário faça a instalação de uma extensão para navegador como a MetaMask[6], uma ferramenta que permite interações do usuário com a Web3 e fornece uma carteira de criptomoedas no ecossistema *Ethereum* com suporte a tokens ERC-20.

A plataforma do presente trabalho foi projetada de forma que quando o usuário tenha terminado o desenvolvimento de seu contrato, seja por meio do editor Blockly presente na tela de edição do projeto (4.2.3) ou diretamente pela tela do compilador (4.2.4), ele possa implantar seu código com facilidade.

Quando o usuário clica em um botão na interface da plataforma para implantar o código do contrato produzido, este é enviado para um WebWorker, que compila o contrato utilizando solc-js. O resultado produzido pelo compilador é então retornado para thread principal, onde o sistema utiliza a biblioteca web3.js para criar a instância do contrato, fazendo o uso do *Application Binary Interface* (ABI) produzido pelo compilador. Em seguida, esta instância é implantada na rede através da transmissão de uma transação de criação de contrato inteligente contendo seu *bytecode* e a conta que criou o contrato.

4.5.5 Material UI e o Material Design

A interface da Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica foi inteiramente desenvolvida utilizando componentes da biblioteca Material UI[7], que inclui uma coleção abrangente de componentes prontos para uso e

[5] https://rinkebyfaucet.com/
[6] https://metamask.io/
[7] https://github.com/mui/material-ui

implementa o Material Design do Google.

O *Material* é um sistema de design elaborado pelo Google e projetado para ajudar desenvolvedores a criar experiências digitais de alta qualidade para aplicações Android, iOS e web. De acordo com Davidov (2021), o “Material Design é o padrão para projetar e criar sites e aplicativos. Surge como uma resposta aos estilos de design antigos, hostis ao usuário e caóticos e seu objetivo é trazer ordem e união ao web design“

Em 2014 o Material Design foi anunciado como o estilo oficial de aplicativos produzidos pelo Google, como o Google Drive e o Gmail e a partir deste momento inúmeros desenvolvedores começaram a usar este estilo, o que contribuiu para seu amplo reconhecimento entre os usuários de aplicações web.

Levando em consideração esses fatores, o presente trabalho optou pela construção de uma interface que faz o uso do sistema de design Material a fim de propiciar ao usuário um ambiente familiar por onde ele possa navegar com facilidade e reconhecer ícones sem dificuldades.

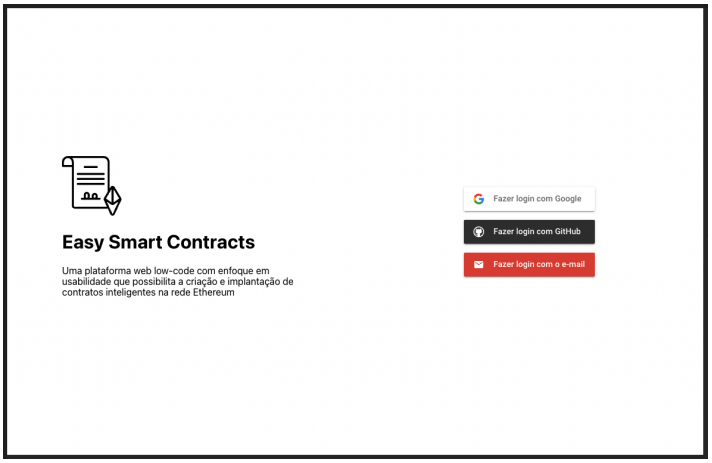
4.5.6 Telas desenvolvidos

Esta subseção é destinada para a exposição das interfaces desenvolvidas no presente trabalho, explicando o fluxo de navegação e de informação pelo sistema.

4.5.6.1 Tela de login

Ao acessar o sistema pela primeira vez o usuário é recebido através da tela de login, composta pelo nome, logo da aplicação e uma pequena descrição sobre o projeto.

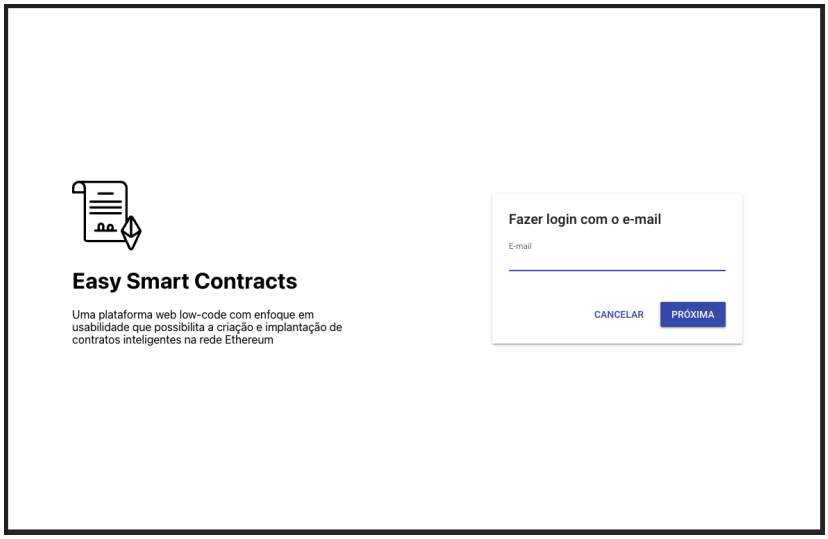
Figura 17 – Tela de login



Fonte: do Autor

Esta tela conta também com três botões na lateral direita, por onde o usuário pode se autenticar utilizando provedores do Google, Github ou email, como mostra a Figura 17.

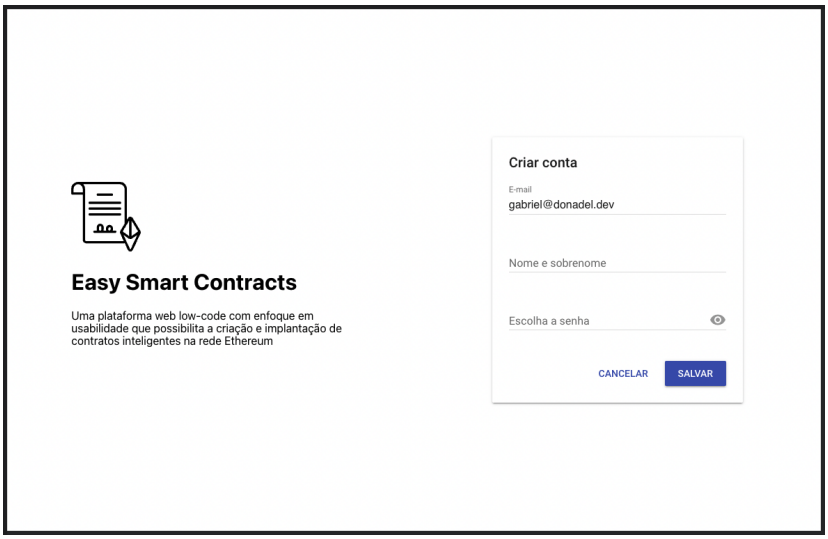
Figura 18 – Tela de login - Fluxo de autenticação por email



Fonte: do Autor

No evento de o usuário ter selecionado o email como método de autenticação e este ainda não possuir cadastro na plataforma, um formulário é mostrado pedindo que o usuário forneça seu endereço de email, nome completo e defina uma senha.

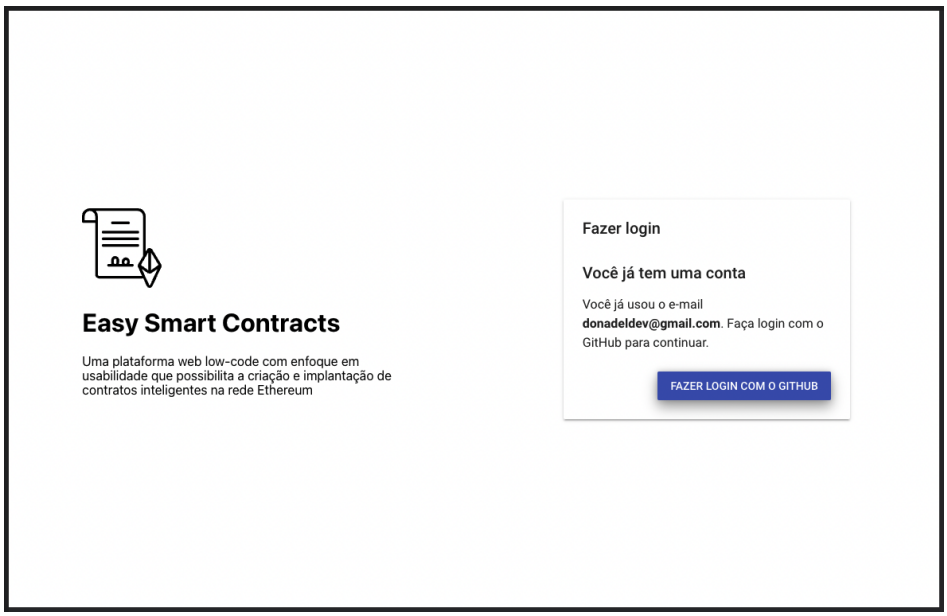
Figura 19 – Tela de login - Fluxo de cadastro por email



Fonte: do Autor

Caso o usuário tente fazer login com um email que já esteja vinculado a outro método de autenticação, como o Github por exemplo, uma mensagem é mostrada pedindo ao usuário que realize o login utilizando o método pelo qual a conta foi cadastrada.

Figura 20 – Tela de login - Fluxo de conta vinculado a outro provedor



Fonte: do Autor

4.5.6.2 Estrutura comum das telas autenticadas

Uma vez que o usuário tenha se autenticado com sucesso na plataforma, todas as telas em diante, com exceção da tela de login, seguirão uma estrutura padrão constituída por três partes principais:

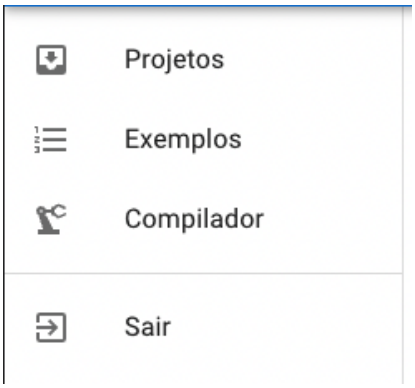
- Um cabeçalho, localizado no topo da tela como representado na Figura 21, constituído do logo e nome da aplicação do lado esquerdo e ao lado direito um botão que leva o usuário ao repositório da plataforma no Github.

Figura 21 – Cabeçalho

Fonte: do Autor

- Um menu navegacional localizado na lateral esquerda da página, que permite que o usuário alterne entre as telas de gerenciamento de projetos (4.5.6.3), exemplo (4.5.6.6) e do compilador (4.5.6.5).

Figura 22 – Menu navegacional



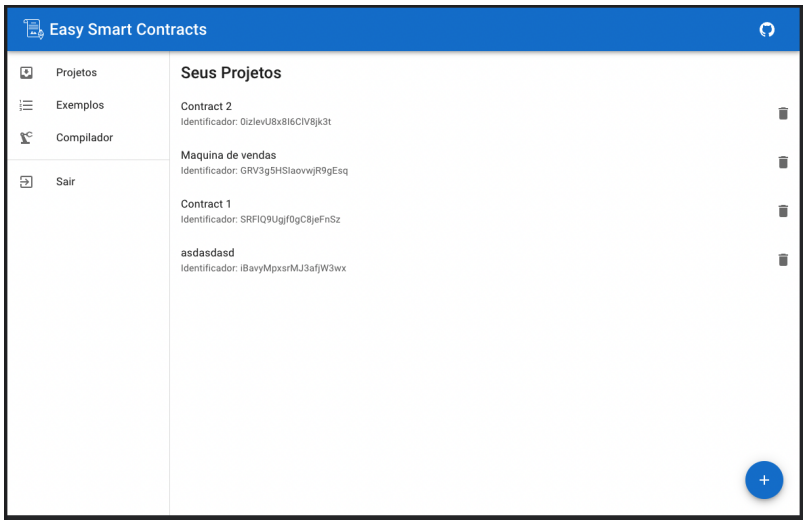
Fonte: do Autor

- E o conteúdo principal da tela, que varia conforme a rota selecionada e fica localizado ao centro.

4.5.6.3 Tela de gerenciamento de projetos

A tela de gerenciamento de projetos é a rota inicial para usuários autenticados e permite a visualização de todos os projetos de contratos inteligentes que o usuário possui.

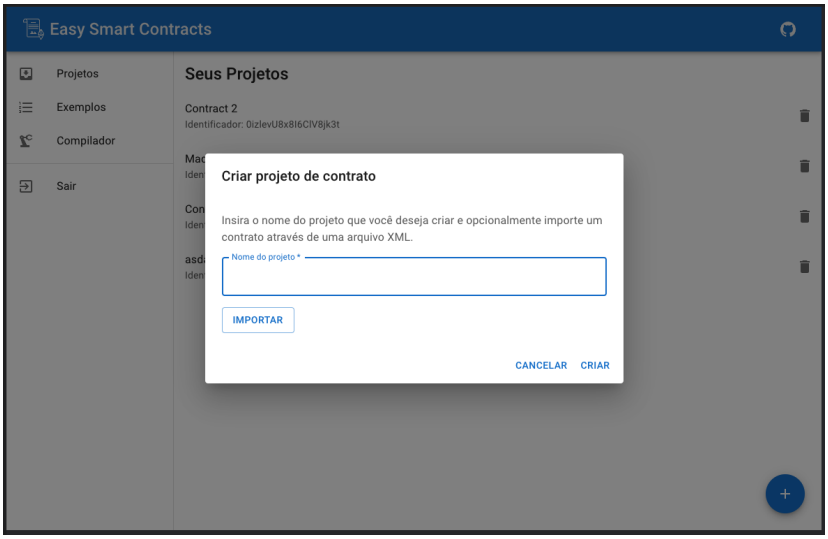
Figura 23 – Tela de gerenciamento de projetos



Fonte: do Autor

Esta tela dispõe de um botão localizado no canto inferior direito que ao ser clicado abre uma modal que permite que o usuário crie e importe novos projetos, como mostra a Figura 24.

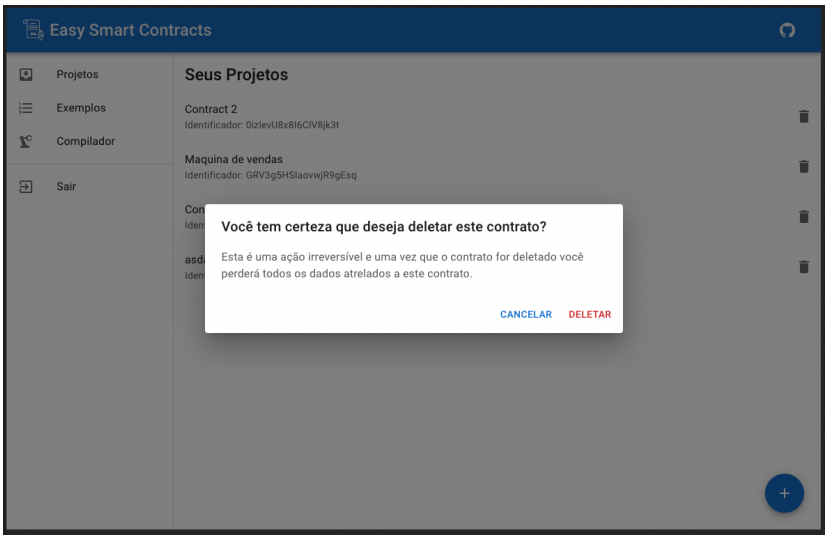
Figura 24 – Tela de gerenciamento de projetos - Modal de criação



Fonte: do Autor

Além disso, cada um dos itens mostrados na lista de projetos possui um botão de deleção, que quando clicado mostra uma mensagem pedindo a confirmação do usuário de que ele realmente deseja deletar o projeto selecionado.

Figura 25 – Tela de gerenciamento de projetos - Modal de deleção



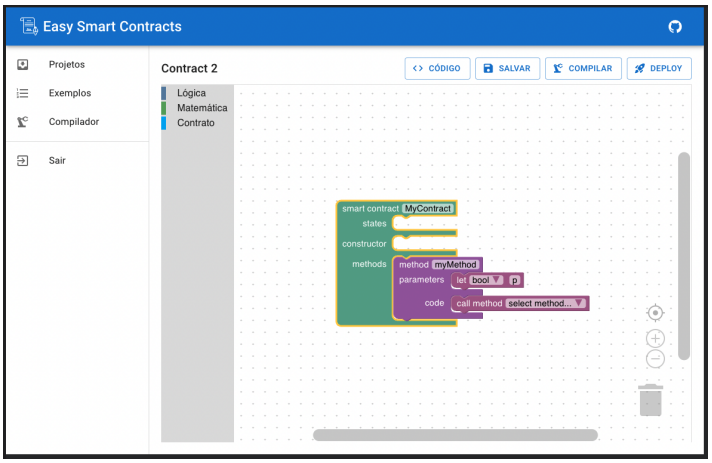
Fonte: do Autor

Caso o usuário clique sobre um item da lista de projetos, este é redirecionado para a tela de edição do projeto (4.5.6.4).

4.5.6.4 Tela de edição do projeto

A tela de edição do projeto permite com que o usuário construa seus contratos inteligentes arrastando blocos dentro do editor.

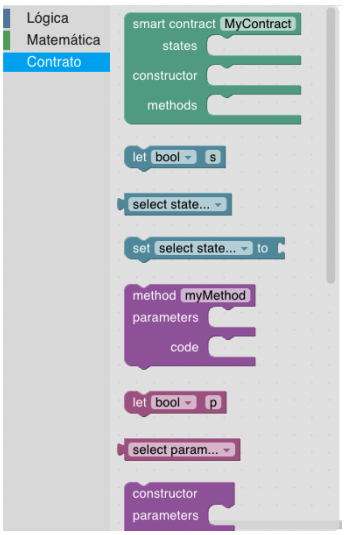
Figura 26 – Tela de edição do projeto



Fonte: do Autor

A Figura 27 mostra alguns dos blocos que o usuário pode selecionar por meio da *toolbox* para compor seu contrato inteligente.

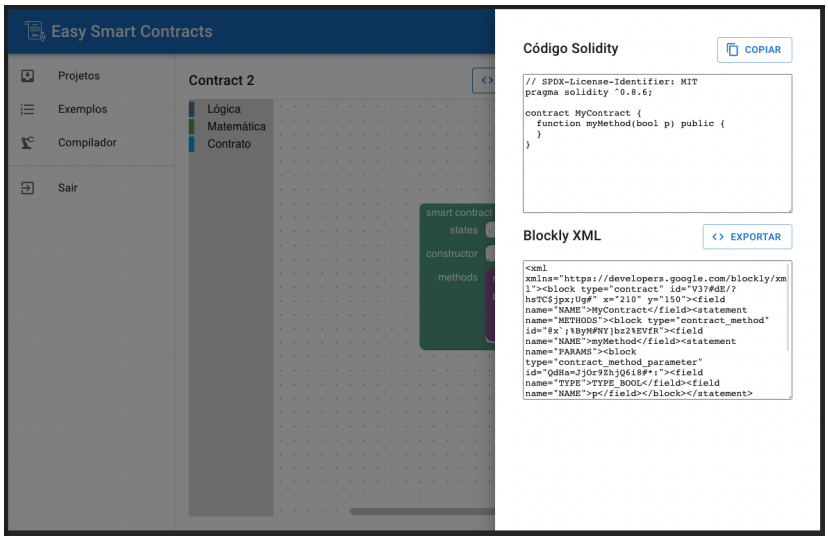
Figura 27 – Tela de edição do projeto - *Toolbox*



Fonte: do Autor

Durante qualquer momento do desenvolvimento do contrato, o usuário pode clicar no botão de “Código“ para revelar uma modal contendo o código Solidity produzido pelo editor Blockly e o XML Blockly que pode ser usado para exportar o projeto.

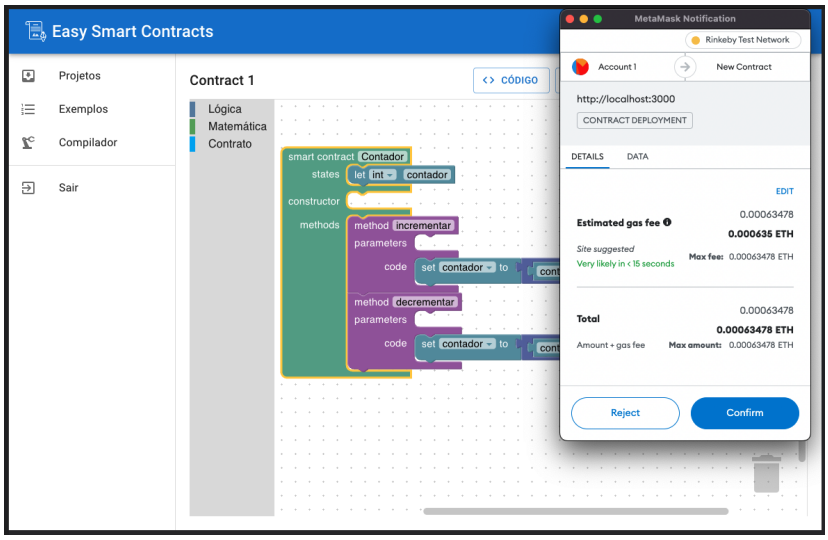
Figura 28 – Tela de edição do projeto - Modal de inspeção de código



Fonte: do Autor

Quando o usuário termina de desenvolver seu contrato inteligente e deseja implantá-lo na rede, ele pode transmitir a transação de criação do contrato por meio do botão “Deploy“. Dessa maneira um *popup* da extensão MetaMask será mostrado na tela para a confirmação da transação, como mostra a Figura 29.

Figura 29 – Tela de edição do projeto - Implantação do contrato via MetaMask

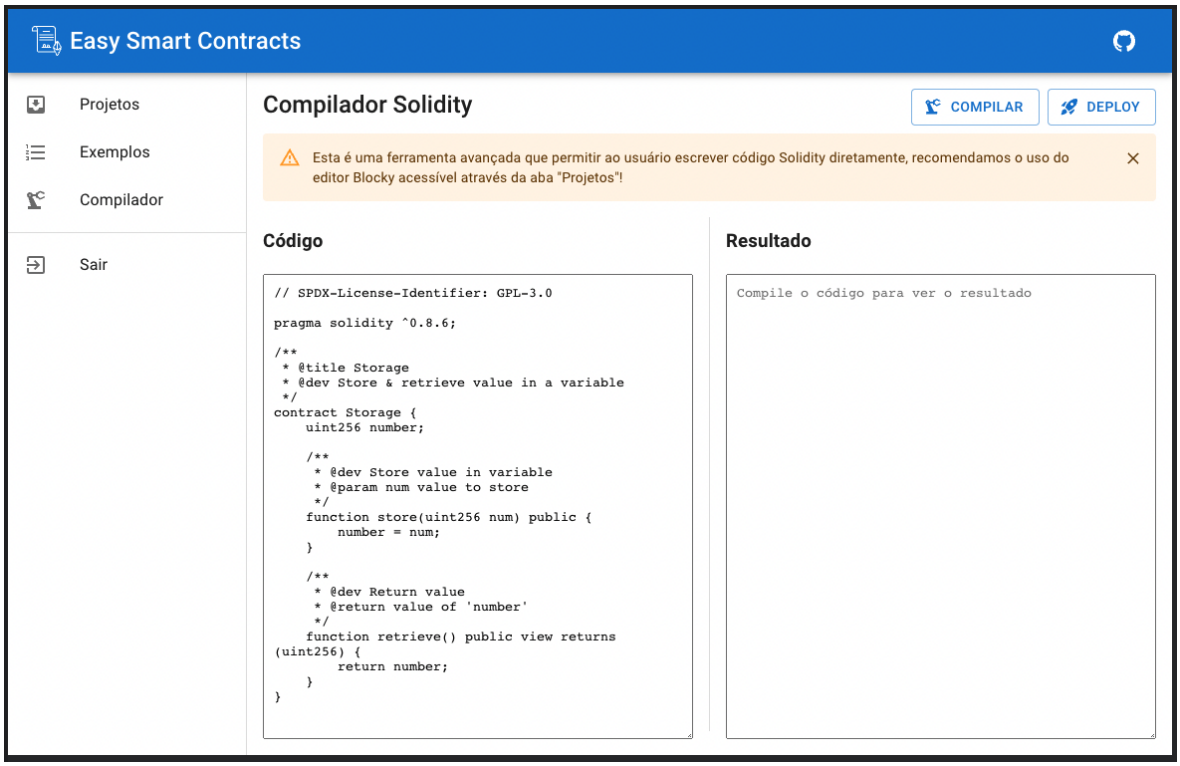


Fonte: do Autor

4.5.6.5 Tela do compilador

A tela do compilador é voltada para usuários avançados e por isso, mostra uma alerta recomendado o uso do editor Blockly para o desenvolvimento dos contratos. De forma semelhante a tela de edição do projeto (4.5.6.4), a tela do compilador permite que o usuário compile e implemente seus contratos inteligentes através de botões na interface.

Figura 30 – Tela do compilador

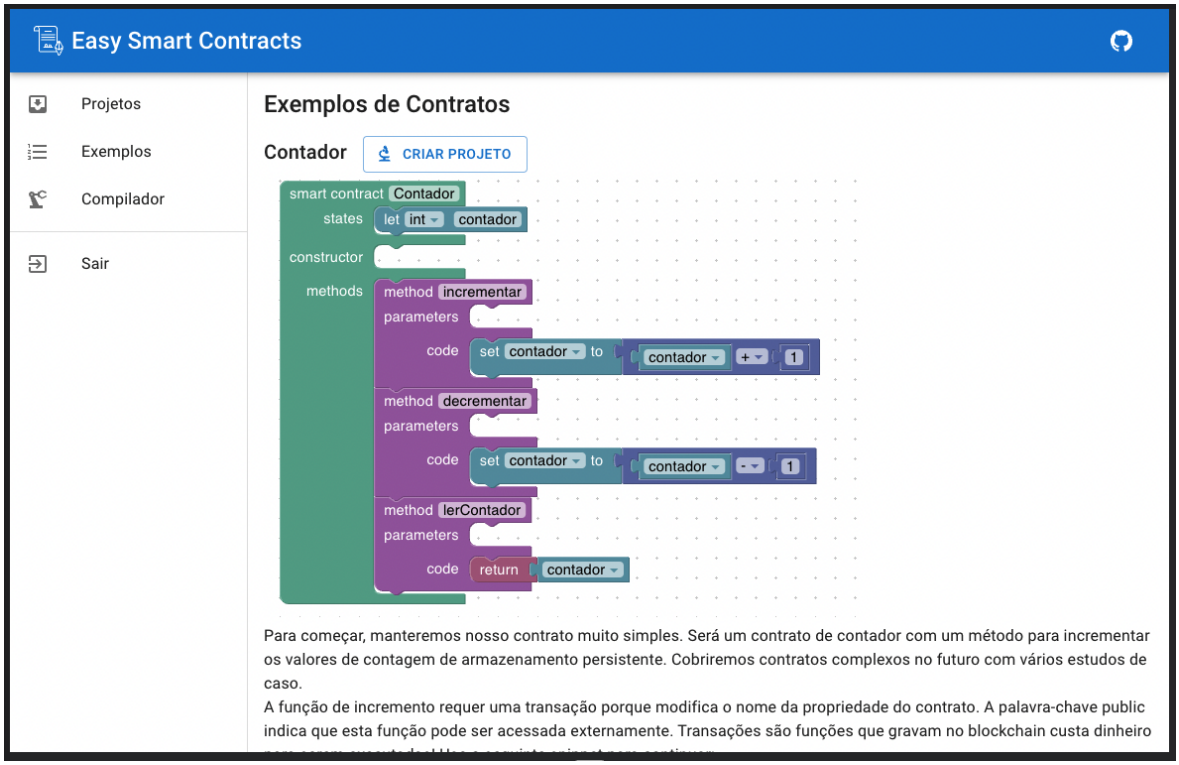


Fonte: do Autor

4.5.6.6 Tela de exemplos

Por último, a tela de exemplos permite que os usuários visualizem exemplos de contratos desenvolvidos dentro da plataforma e aprendam sobre o desenvolvimento de código Solidity através de blocos interligados.

Figura 31 – Tela de exemplos



Fonte: do Autor

5 USABILIDADE DA PLATAFORMA

A Norma NBR 9241-11 define o termo usabilidade como: “Medida na qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso.“ (ABNT, 2002) Onde eficácia: é a integridade com a qual o usuário consegue atingir determinados objetivos; eficiência: é a relação entre exatidão e integridade com o qual os usuários alcançam determinado objetivo e os recursos gastos para atingi-los e, satisfação: é o conforto dos usuários ao utilizar o sistema.

De forma parecida, Jakob Nielsen propôs dez princípios gerais para o design da interface do usuário, como mostra a Tabela 3, os quais denominou de “heurísticas“ pois, segundo ele, estes princípios estão mais na natureza das regras práticas do que nas diretrizes de usabilidade específicas (NIELSEN, 2005).

Tabela 3 – Heurísticas de Nielsen

01	Visibilidade do status do sistema: O sistema deve sempre manter o usuário informado sobre o que está acontecendo.
02	Semelhança entre o sistema e o mundo real: O sistema deve seguir as convenções do mundo real, fazendo as informações aparecerem de forma lógica e natural.
03	Controle e liberdade: Deve existir a possibilidade do usuário sair do estado em que se encontra, ou retomar facilmente ao estado anterior.
04	Consistência e padrões: Seguir convenções, indicar ações iguais de maneira similar e utilizar o mesmo tipo de linguagem em toda a interface.
05	Prevenção de erros: Design que evite que problemas ocorram, além de boas mensagens de erro.
06	Reconhecimento ao invés de recordação: Utilizar símbolos com contexto e em lugares coerentes para que o usuário entenda o funcionamento de maneira fácil.
07	Flexibilidade e eficiência de utilização: O sistema deve atender a ambos os usuários, inexperientes e experientes.
08	Design minimalista: Mensagens de diálogos não devem conter informações irrelevantes. Informações a mais conflitam com a visibilidade.
09	Ajudar o reconhecimento, diagnóstico e recuperação de erros: Mensagens de erros devem ser claras e objetivas, devem indicar o problema com precisão e sugerir uma solução.
10	Ajuda e documentação: Qualquer informação deve ser fácil de pesquisar e deve ser focada na tarefa do usuário.

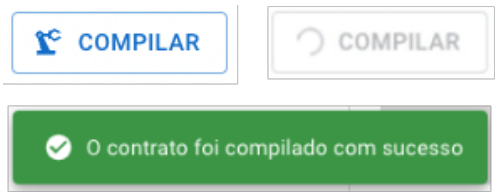
As heurísticas de Nielsen podem ser utilizadas como princípios para a avaliação da usabilidade de interfaces, podendo ser aplicadas em qualquer momento do pro-

jeto com o intuito de proporcionar uma ótima experiência aos usuários, fazendo com que apenas informações úteis sejam apresentadas, focando na fácil compreensão, interação simples e design minimalista.

Sendo assim, o presente trabalho realizou a análise da usabilidade da plataforma *low-code* desenvolvida, verificando o cumprimento de cada uma das dez Heurísticas de Nielsen.

A Heurística 01 diz respeito a visibilidade do status do sistema, requisitando que os usuários mantenham-se informados sobre o estado atual da aplicação, o que pode ser observado em diferentes partes da plataforma desenvolvida, como por exemplo, enquanto o usuário compila um contrato inteligente, o botão “compilar“ passa a mostrar um estado de carregando e depois uma mensagem de sucesso, caso o contrato tenha sido compilado corretamente, como mostra a Figura 32.

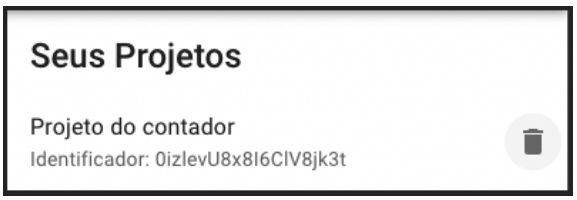
Figura 32 – Feedback da interface ao compilar um contrato



Fonte: do Autor

A Heurística 02 trata da correspondência do mundo real com o sistema, cabendo a ele utilizar a linguagem que seus usuários estão familiarizados, seja no uso de frases, imagens ou conceitos. Um exemplo do cumprimento desta heurística é visto na tela de gerenciamento de projetos (4.5.6.3), onde é feito o uso do símbolo de lixeira para representar a ação de exclusão de projetos.

Figura 33 – Lista de projetos de contratos



Fonte: do Autor

Já a Heurística 03 determina que para existir uma boa usabilidade, um sistema deve permitir que o usuário tenha o controle sobre a aplicação e possa realizar suas ações de forma livre, mas também possa reverter ações realizadas por engano. A plataforma também está em acordo com esta heurística, e uma das interfaces por

onde isso pode ser observado é no editor de projetos(4.5.6.4), que permite que usuário desfaça suas ações utilizando o atalho “Ctrl+Z“.

Na Heurística 04 Nielsen pontua a necessidade da consistência entre as interfaces de uma sistema e a utilização de um padrão para que usuários usufruam da plataforma com facilidade. A adequação de tais fatores pode ser observada por todas a plataforma, tendo em vista que esta foi desenvolvida utilizando o Material Design, como explicado na subseção 4.5.5.

A Heurística 05 fala a respeito da construção de interfaces que previnam a ocorrência de erros por parte do usuário, um exemplo do cumprimento deste requisito está presente na tela de edição do projeto (4.5.6.4), pois ela previne que o usuário cometa erros no desenvolvimento dos contratos, dado que não permite que sejam conectados blocos incompatíveis e que possam gerar erros quando o código Solidity for compilado.

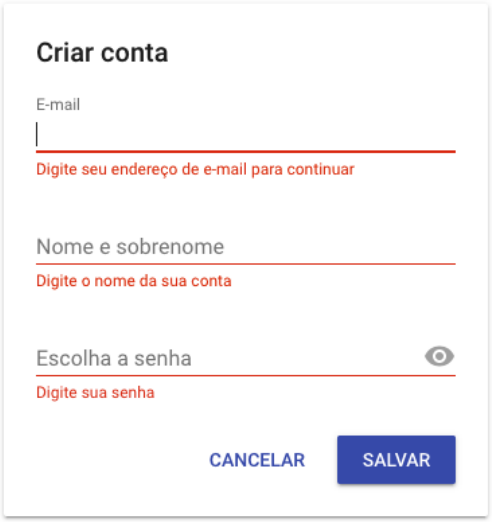
A Heurística 06 trata do fácil reconhecimento da interface ao invés da necessidade da recordação por parte do usuário, de forma que o sistema deva minimizar a quantidade de informações que o usuário precisa memorizar. O sistema desenvolvido no presente trabalho faz uso de um *layout* muito comum em aplicações web, contendo um cabeçalho no topo da página, um menu navegacional do lado esquerdo e o conteúdo principal ao centro, como recomendam as diretrizes do Material Design e por isso, também está de acordo com esta heurística.

A Heurística 07 estipula que o sistema deve atender tanto às necessidades dos usuários leigos quanto a dos experientes. A plataforma desenvolvida possui uma tela planejada especificamente para o uso de usuário experientes que é a tela do compilador (4.5.6.5), porém, a aplicação também permite que ambos os perfis de usuários desenvolvam contratos através da tela de edição do projeto (4.5.6.4), onde o usuário com um perfil avançado pode usar atalhos de teclado como, “Ctrl+C“, “Ctrl+V“, “Ctrl+Z“ e “Delete“, para acelerar o desenvolvimento do projeto.

A Heurística 08 estabelece que sistemas devem fazer o uso de um design minimalista, apresentando o conteúdo ao usuário de maneira direta. A plataforma desenvolvida utiliza uma linguagem simples e objetiva em suas interfaces, além de ser clara quando apresenta diálogos ao usuário.

A Heurística 09 diz respeito a clareza com que diálogos e mensagens de erros devem ser apresentadas ao usuário, de forma que este não fique em dúvida sobre qual erro aconteceu e que entenda como pode solucioná-lo. Um dos exemplos onde este requisito é posto em prática, no fluxo de cadastro de conta de email, como mostra a Figura 34, caso o usuário não preencha os dados necessários os campos com erro serão sinalizados em vermelho, apresentado pequenos textos que explicam ao usuário o que deve ser feito.

Figura 34 – Estados de erro no formulário de cadastro de conta por email



Fonte: do Autor

Por fim, a Heurística 10 trata da disponibilização de alguma forma de documentação ou sistema de ajuda para o usuário, de maneira que este possa sanar eventuais dúvidas sobre o sistema. A aplicação desenvolvida possui uma interface dedicada a este este aspecto, como se vê na tela de exemplos (4.5.6.6), que explica de forma geral o funcionamento da plataforma e como o usuário pode desenvolver seus projetos.

Levando em consideração as 10 heurísticas de Jakob Nielsen e o fato do sistema desenvolvido no presente trabalho atender a todos seus dez princípios gerais, pode-se afirmar que a plataforma desenvolvida possui uma ótima usabilidade na visão de Nielsen.

6 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho teve como objetivos o reconhecimento na literatura especializada do estado da arte de plataformas *low-code*; o desenvolvimento de uma plataforma *low-code* para a criação de contratos inteligentes que pudessem ser implantados na rede Ethereum, além da avaliação de sua usabilidade.

Durante a elaboração deste estudo ficou evidente que a área da qual o trabalho trata ainda é muito incipiente e que as tecnologias abordadas estão sofrendo constantes atualizações, fazendo com que a literatura encontrada no meio acadêmico fique obsoleta rapidamente.

Para o desenvolvimento do trabalho foi necessário o uso extensivo das documentações da linguagem de programação Solidity e da biblioteca Blockly, pois as funcionalidades de ambas são melhoradas pela comunidade constantemente e suas documentações são os melhores canais pelos quais desenvolvedores podem ficar a par das mudanças.

Conclui-se que o estudo alcançou os objetivos gerais e específicos estipulados, uma vez que o estado da arte de plataformas *low-code* foi reconhecido, dando suporte para o desenvolvimento de uma plataforma web *low-code* que possibilita a criação e implantação de contratos inteligentes na rede Ethereum, por meio de uma interface amigável e intuitiva, sem a necessidade de conhecimento prévio em programação, além de ser realizada também a análise da usabilidade da plataforma fundamentada nas dez heurísticas de Jakob Nielsen.

Como destacado na seção 3.5 muitos dos problemas de engenharia de software de aplicativos de *blockchain* ainda não foram bem explorados e, portanto, construir um aplicativo baseado em *blockchain* que possua qualidade comercial normalmente é muito mais desafiador do que um aplicativo corporativo tradicional. Porém, a solução desenvolvida no presente trabalho facilita tal processo de maneira com que contratos inteligentes sejam desenvolvidos de forma mais simples, e desta forma este estudo ajuda avançar o estado da arte descrito nos trabalhos correlatos, expandindo as soluções de engenharia de software relacionadas à *blockchain*.

Ganham destaque como facilitadores do desenvolvimento do trabalho, a biblioteca Blockly, a qual forneceu a interface utilizada no editor de contratos e proporcionou que o presente trabalho não tivesse que desenvolver um editor *low-code* do zero e o conjunto de soluções oferecidas pelo Firebase, tendo em vista que estas encapsularam funcionalidades que demandariam o desenvolvimento de um servidor externo para tratar da autenticação e dos dados do usuário.

O advento dos contratos inteligentes em ecossistemas que utilizam a tecnologia *blockchain* ocorreu a menos de dez anos e ainda passa por um processo contínuo de maturação, tornando esse um dos fatores limitantes do presente estudo, porém, esta la-

cuna serve como impulsionadora para trabalhos futuros, que poderão dar continuidade a plataforma desenvolvida.

Sendo assim, sugere-se o prosseguimento do presente trabalho dos seguintes pontos:

- Extensão do suporte a linguagem Solidity a biblioteca Blockly, adicionando o conceito de funções públicas, privadas, internas e externas; suporte a diretiva de *import*; adição do restante dos tipos suportados pela linguagem e a possibilidade do usuários poder escolher qual versão da linguagem desejam usar.
- Desenvolvimento de um mecanismo que permita a interação com os métodos dos contratos inteligentes implantados através da interface da plataforma.
- E a realização de uma pesquisa de campo validando a eficiência, eficácia e satisfação do usuário ao utilizar a plataforma.

Por fim, a plataforma web *low-code* desenvolvida no presente trabalho garante que de forma amigável e intuitiva possam ser criados e implementados contratos inteligentes na rede Ethereum, independentemente do nível de conhecimento do usuário.

REFERÊNCIAS

ANDRADE, Maria Margarida de. **Introdução à metodologia do trabalho científico**. São Paulo: Atlas, 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 9241-11**: Requisitos ergonômicos para trabalho de escritórios com computadores. Parte 11-Orientações sobre usabilidade. Rio de Janeiro, 2002.

BLOCKLY. **Introduction to Blockly**. [*S.l.*: *s.n.*], 2022. Disponível em: https://developers.google.com/blockly/guides/overview.

BUTERIN, Vitalik. Design rationale, 2017. Disponível em: https://eth.wiki/en/fundamentals/design-rationale.

CRUZ, Mauro A. A. da; PAULA, Heitor T. L. de; CAPUTO, Bruno P. G.; MAFRA, Samuel B.; LORENZ, Pascal; RODRIGUES, Joel J. P. C. OLP—A RESTful Open Low-Code Platform. **Future Internet**, v. 13, n. 10, 2021. ISSN 1999-5903. DOI: 10.3390/fi13100249. Disponível em: https://www.mdpi.com/1999-5903/13/10/249.

DAVIDOV, Sergei. **What Is Material Design? Definition, Uses, and Examples**. [*S.l.*: *s.n.*], fev. 2021. Disponível em: https://elementor.com/blog/what-is-material-design/.

DE ANGELIS, Stefano; ANIELLO, Leonardo; BALDONI, Roberto; LOMBARDI, Federico; MARGHERI, Andrea; SASSONE, Vladimiro. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain, 2018.

DESTEFANIS, Giuseppe. Design Patterns for Smart Contract in Ethereum. *In*: IEEE. 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C). [*S.l.*: *s.n.*], 2021. P. 121–122.

DUFFY, Shannon. **Salesforce Is Leader in 2019 Gartner Magic Quadrant for Low Code Application Platforms**. [*S.l.*: *s.n.*], ago. 2019. Disponível em: https://www.salesforce.com/blog/gartner-lcap/. Acesso em: 2 dez. 2021.

ELI, Paulo Henrique. Desenvolvimento de um ambiente de apoio ao ensino de algoritmos e programação: usando blockly. Programa de Pós-Graduação em Tecnologia da Informação e Comunicação, Universidade Federal de Santa Catarina, Araranguá, p. 140, 2017. Disponível em: `https://repositorio.ufsc.br/handle/123456789/185430`. Acesso em: 2 dez. 2021.

ETHEREUM. **Language Influences — Solidity 0.8.15 documentation**. [*S.l.*: *s.n.*], 2022. Disponível em: `https://docs.soliditylang.org/en/v0.8.15/language-influences.html`.

EUROPEAN COMMISSION. **How can europe benefit from blockchain technologies?** [*S.l.*], 2019. Disponível em: `http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=49649`. Acesso em: 13 mar. 2022.

FERRETTI, Stefano; D'ANGELO, Gabriele. On the Ethereum blockchain structure: A complex networks theory perspective. **Concurrency and Computation: Practice and Experience**, v. 32, n. 12, e5493, 2020. e5493 cpe.5493. DOI: `https://doi.org/10.1002/cpe.5493`.

FIREBASE. **Firebase Pricing**. [*S.l.*: *s.n.*], 2022. Disponível em: `https://firebase.google.com/pricing?hl=pt-br`.

GIL, Antônio Carlos. **Como Elaborar Projetos de Pesquisa**. São Paulo: Atlas, 2002.

HECHT, Robin; JABLONSKI, Stefan. NoSQL evaluation: A use case oriented survey. *In*: 2011 International Conference on Cloud and Service Computing. [*S.l.*: *s.n.*], 2011. P. 336–341. DOI: `10.1109/CSC.2011.6138544`.

KING, Sunny; NADAL, Scott. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. **self-published paper, August**, v. 19, n. 1, 2012.

LARMAN, Craig. **Utilizando UML e Padrões**. [*S.l.*]: Bookman Editora, 2000. ISBN 9788577800476. Disponível em: `https://books.google.com.br/books?id=hzl2tmT8QkUC`. Acesso em: 13 mar. 2022.

LIAO, Chun-Feng; CHENG, Ching-Ju; CHEN, Kung; LAI, Chen-Ho; CHIU, Tien; WU-LEE, Chi. Toward a service platform for developing smart contracts on blockchain in bdd and tdd styles. *In*: IEEE. 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA). [*S.l.*: *s.n.*], 2017. P. 133–140.

MARCHESI, Lodovica; MARCHESI, Michele; DESTEFANIS, Giuseppe; BARABINO, Giulio; TIGANO, Danilo. Design patterns for gas optimization in ethereum. *In*: IEEE. 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). [*S.l.*: *s.n.*], 2020. P. 9–15.

MDN. **Web Workers API - APIs da Web**. [*S.l.*: *s.n.*], 2022. Disponível em: `https://developer.mozilla.org/pt-BR/docs/Web/API/Web_Workers_API`.

NAKAMOTO, Satoshi. Bitcoin open source implementation of P2P currency. **P2P foundation**, v. 18, 2009.

NAKAMOTO, Satoshi. Bitcoin: A peer-to-peer electronic cash system. **Decentralized Business Review**, p. 21260, 2008.

NIELSEN, Jakob. **Ten usability heuristics**. [*S.l.*]: http://www. nngroup. com/articles/ten-usability-heuristics/(acc-essed . . ., 2005.

OLIVEIRA, Maria Marly de. **Como fazer pesquisa qualitativa**. Rio de Janeiro: Vozes, 2007.

REALTIMEBOARD, INC. **Miro**. [*S.l.*: *s.n.*], 1 mai. 2022. Disponível em: `https://miro.com/pt/`.

ROSEMBERG, Carlos; SCHILLING, Albert; BASTOS, Cristianne; ARARIPE, Rodrigo. Prototipação de software e design participativo: uma experiência do atlântico. **IHC**, v. 8, p. 312–315, 2008.

SAHAY, Apurvanand; INDAMUTSA, Arsene; DI RUSCIO, Davide; PIERANTONIO, Alfonso. Supporting the understanding and comparison of low-code development platforms. *In*: IEEE. 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). [*S.l.*: *s.n.*], 2020. P. 171–178.

SAYEED, Sarwar; MARCO-GISBERT, Hector; CAIRA, Tom. Smart Contract: Attacks and Protections. **IEEE Access**, PP, p. 1–1, jan. 2020. DOI: 10.1109/ACCESS.2020.2970495.

SZABO, Nick. **Smart contracts**. [*S.l.*: *s.n.*], 1994. Disponível em: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. Acesso em: 13 mar. 2022.

TIKHOMIROV, Sergei; VOSKRESENSKAYA, Ekaterina; IVANITSKIY, Ivan; TAKHAVIEV, Ramil; MARCHENKO, Evgeny; ALEXANDROV, Yaroslav. Smartcheck: Static analysis of ethereum smart contracts. *In*: PROCEEDINGS of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. [*S.l.*: *s.n.*], 2018. P. 9–16.

TSANKOV, Petar; DAN, Andrei; DRACHSLER-COHEN, Dana; GERVAIS, Arthur; BUENZLI, Florian; VECHEV, Martin. Securify: Practical security analysis of smart contracts. *In*: PROCEEDINGS of the 2018 ACM SIGSAC Conference on Computer and Communications Security. [*S.l.*: *s.n.*], 2018. P. 67–82.

VADGAMA, Nikhil; TASCA, Paolo. An Analysis of blockchain adoption in supply chains between 2010 and 2020. **Frontiers in Blockchain**, Frontiers, v. 4, p. 8, 2021.

WACKEROW, Paul. **JSON-RPC API**. [*S.l.*: *s.n.*], 2022. Disponível em: https://ethereum.org/pt/developers/docs/apis/json-rpc/.

WANG, Qin; LI, Rujia; WANG, Qi; CHEN, Shiping; RYAN, Mark; HARDJONO, Thomas. Exploring Web3 From the View of Blockchain. **arXiv preprint arXiv:2206.08821**, 2022.

WASZKOWSKI, Robert. Low-code platform for automating business processes in manufacturing. **IFAC-PapersOnLine**, v. 52, n. 10, p. 376–381, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019. ISSN 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.10.060. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405896319309152.

Apêndices

APÊNDICE A – ARTIGO PLATAFORMA DE DESENVOLVIMENTO DE *SMART CONTRACTS* BASEADA EM INTERFACE GRÁFICA

Plataforma de Desenvolvimento de Smart Contracts Baseada em Interface Gráfica

Gabriel Donadel Dall'Agnol, Jean Everson Martina

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

gabriel.donadel@grad.ufsc.br, jean.martina@ufsc.br

Abstract. *The business market has been experiencing a high demand for development and deployment platforms based on graphical user interfaces that are focused on usability and above all, user experience. In parallel, blockchain technology has become increasingly popular, especially the Ethereum platform, which has the differential of creating smart contracts and developing decentralized applications. However, the lack of a user-friendly interface and the requirement of specific technical skills create a high barrier to entry for the general public, making few able to extract the full potential of the technology. This work aims to develop a low-code web platform with a focus on usability that enables the creation and implementation of smart contracts.*

Resumo. *O mercado vem experienciando uma alta demanda por plataformas de desenvolvimento de software baseadas em interface gráfica, focadas na usabilidade e experiência do usuário. Em paralelo, a tecnologia blockchain tem se tornado cada vez mais popular, em especial a da plataforma Ethereum, que tem como diferencial a criação de contratos inteligentes. Porém, a falta de uma interface amigável e o requerimento de habilidades técnicas criam uma alta barreira de entrada ao público geral, fazendo com que poucos estejam aptos a extrair o potencial máximo da tecnologia. O objetivo deste trabalho é desenvolver uma plataforma web low-code com enfoque em usabilidade que possibilite a criação e implantação de contratos inteligentes.*

1. Introdução

A tecnologia blockchain vem ganhando muita força nos últimos anos, estando presente nos mais diversos setores, desde a área da agricultura, saúde, finanças, logística, setores governamentais e na indústria em geral (VADGAMA; TASCA, 2021). Entre as diversas blockchains existentes no mercado destaca-se a utilizada pela moeda digital Ether, a plataforma Ethereum, que tem como principal diferencial a possibilidade da criação de contratos inteligentes e o desenvolvimento de aplicações descentralizadas. (DESTEFANIS, 2021)

No entanto, esta tecnologia ainda é relativamente nova, em especial quando falamos de contratos inteligentes, que foram apenas introduzidos em 2015 com o lançamento oficial da rede Ethereum. Atualmente, a falta de uma interface amigável para o desenvolvimento de contratos inteligentes e o requerimento de habilidades técnicas criam uma alta barreira de entrada para o público geral, fazendo com que poucos estejam aptos a extrair o potencial máximo da tecnologia e de fato utilizem contratos

inteligentes, os quais possibilitam negociações confiáveis sem a necessidade de uma entidade mediadora (TSANKOV et al., 2018).

Ao mesmo tempo observa-se uma alta demanda no mercado por plataformas low-code, plataformas de desenvolvimento e implantação de software baseadas em interface gráfica com abstrações visuais, de forma a exigir pouco ou nenhum desenvolvimento de código. (SAHAY et al., 2020)

É neste contexto que o trabalho será elaborado, visando compreender as problemáticas no cenário dos contratos inteligentes frente à análise da usabilidade de plataformas baseadas em interface gráfica e desenvolver uma plataforma web lowcode que possibilite o desenvolvimento e a implantação de contratos inteligentes na rede Ethereum.

2. Desenvolvimento

2.1. Análise de Requisitos

De acordo com Larman requisitos são uma descrição das necessidades ou dos desejos para um produto, onde o objetivo básico é identificar e documentar o que é realmente necessário, de forma a comunicar claramente essa informação tanto ao cliente quanto aos membros da equipe de desenvolvimento, sendo de suma importância a definição de requisitos de maneira não-ambígua, para que assim os riscos sejam identificados e não aconteçam surpresas durante o desenvolvimento do produto (LARMAN, 2000).

Os requisitos de um software podem ser classificados em Requisitos Funcionais (RF) ou Requisitos Não-Funcionais (RNF). Os Requisitos Funcionais são os responsáveis por definir funções e comportamentos do software, já os Requisitos Não Funcionais dizem respeito à restrições de desenvolvimento, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade e padrões a serem seguidos.

2.1.1. Requisitos Funcionais

Para o levantamento dos RF foram levados em consideração os recursos mínimos necessários para o funcionamento da aplicação e as principais funcionalidades encontradas em outras plataformas low-code.

Tabela 1. Requisitos Funcionais da Plataforma

RF	Descrição
RF-001	O sistema deve permitir o usuário se autenticar com o Google, Github ou email.
RF-002	O sistema deve permitir o usuário se cadastrar.
RF-003	O sistema deve permitir a criação e edição de contratos inteligentes por meio de blocos.
RF-004	O sistema deve permitir salvar e excluir contratos.
RF-005	O sistema deve permitir a compilação de contratos.
RF-006	O sistema deve permitir implantar na rede Ethereum os contratos criados.
RF-007	O sistema deve permitir o usuário desfazer ações.
RF-008	O sistema deve permitir importar e exportar contratos.
RF-009	O sistema deve permitir o usuário listar exemplos de contratos.

2.1.2. Requisitos Não-Funcionais

Para o levantamento dos RNF foram levadas em consideração as principais tecnologias utilizadas para o desenvolvimento de plataformas Web3 e o conhecimento prévio do autor.

Tabela 2. Requisitos Não-Funcionais da Plataforma

RNF	Descrição
RNF-001	A ferramenta deve suportar os navegadores Google Chrome e Microsoft Edge.
RNF-002	A interface da aplicação deve ser desenvolvida utilizado o framework React JS.
RNF-003	Para a utilização da ferramenta deve ser necessário o acesso à rede mundial de computadores.
RNF-004	Para a implantação dos contratos através da ferramenta deve ser necessário a instalação da extensão de navegador MetaMask.
RNF-005	A plataforma deve ser desenvolvida utilizando como base a biblioteca Blockly.
RNF-006	A aplicação deve compilar contratos por meio de um *Web Worker*.

2.2. Prototipação

No campo de desenvolvimento de sistemas, protótipos são representações limitadas de um design, podendo ser usados como ferramenta de comunicação entre membros de uma equipe de desenvolvimento ou ainda puramente como meio para explorar ideias (ROSEMBERG et al., 2008).

De maneira geral protótipos podem ser classificados em algumas categorias, sendo elas: protótipos de baixa fidelidade, que tem como foco a interação do usuário com o sistema, os componentes da interface e estrutura geral; protótipos de alta fidelidade, úteis para demonstrar padrões e guias de estilo, tendo em vista que representam uma imagem real

do sistema; protótipos executáveis, que focam na navegação do sistema, possibilitando testar o uso da interface, normalmente envolvendo código em uma linguagem de programação.

Para este trabalho optou-se apenas pelo uso de protótipos de baixa fidelidade, levando em consideração o fato de poderem ser construídos de maneira ágil e prática, ao mesmo tempo que são capazes de proporcionar uma visão geral do sistema, demonstrando as possibilidades de navegação e quais atividades o sistema comporta.

Durante a etapa de prototipação, foi utilizada a ferramenta Miro (2022), uma aplicação web que permite a criação de protótipos de baixa fidelidade através de wireframes.

Após uma série de iterações optou-se pelo uso do nome Easy Smart Contracts para a plataforma, com o intuito de enfatizar a facilidade com que o usuário pode desenvolver contratos inteligentes através da ferramenta.

2.2.1. Tela de login

Ao acessar o sistema pela primeira vez o usuário é direcionado para a tela de login, Figura 1, constituída por: no lado esquerdo um logo e o nome da aplicação e no lado direito três botões de autenticação.

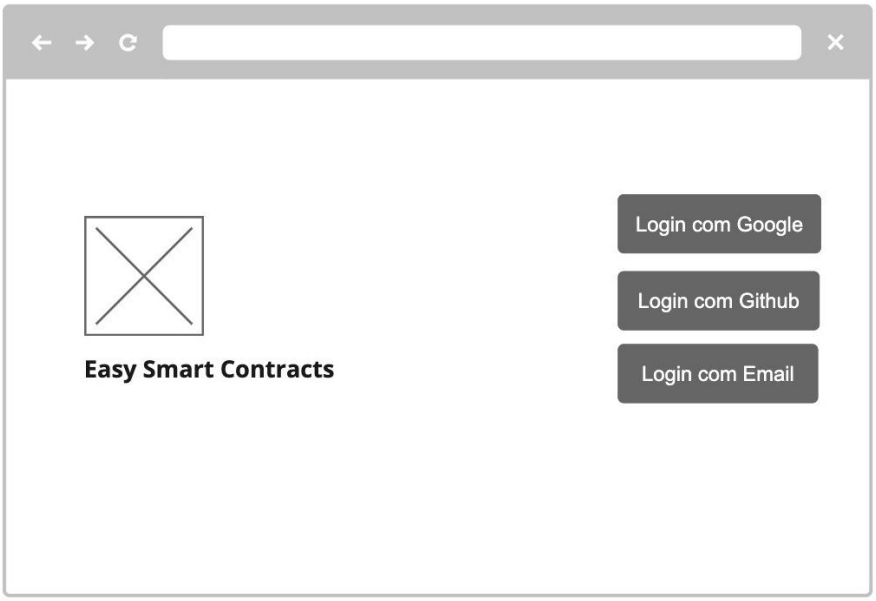


Figura 1. Tela de login

Interagindo com os botões na lateral direita o usuário pode se autenticar através do Google, Github ou email, cada um deles possuindo um fluxo próprio, lidando com estados de carregamento e eventuais mensagens de erro, uma vez autenticado o usuário é redirecionado a tela de gerenciamento dos projetos. Na eventualidade do usuário ainda não possuir uma conta, esta será criada automaticamente.

2.2.2. Tela de gerenciamento dos projetos

A tela de gerenciamento dos projetos é a rota inicial para usuários autenticados, e é por meio dela que os usuários podem listar, criar, deletar e importar projetos de contratos inteligentes.

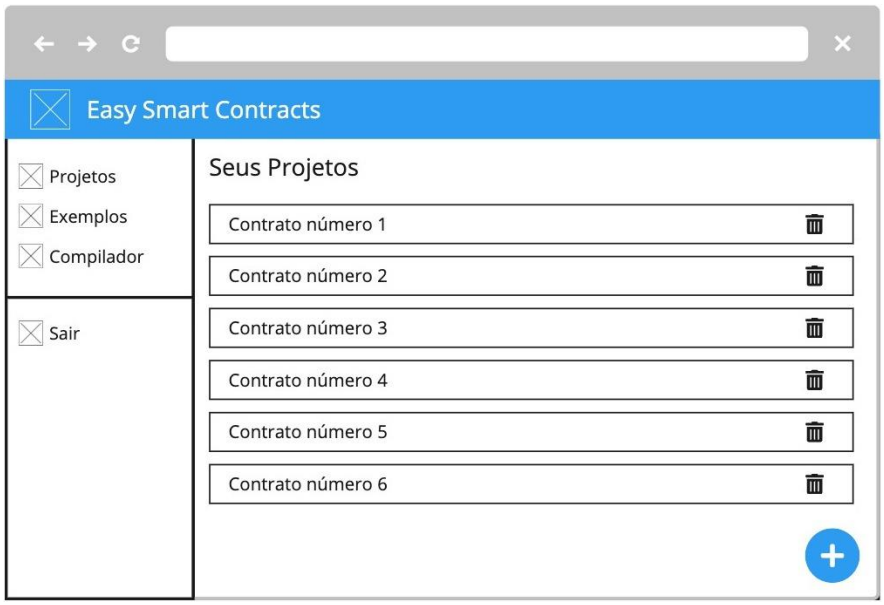


Figura 2. Tela de gerenciamento dos projetos

Como pode-se observar na Figura 2, esta tela é composta por três principais seções: um cabeçalho, um menu navegacional na lateral esquerda e a lista dos projetos do usuário ao centro.

O cabeçalho e o menu navegacional são os mesmos apresentados em todas telas, com exceção da tela de login 2.2.1. É através do menu na lateral esquerda que o usuário pode navegar para as outras telas da aplicação, como a tela do compilador 2.2.4, ou a tela de exemplos 2.2.5. Além disso, este menu permite que o usuário saia de sua conta, redirecionando-o para a tela de login 2.2.1.

Já o conteúdo principal está localizado ao centro da tela, contendo um título, a lista dos projetos que o usuário possui e um botão para adição de projetos no canto inferior direito.

Cada item da lista conta com o nome do projeto e um ícone de lixeira, ao clicar em um dos itens o usuário é redirecionado para a tela de edição do projeto selecionado 2.2.3 e ao clicar no ícone de lixeira o usuário pode deletar seu projeto.

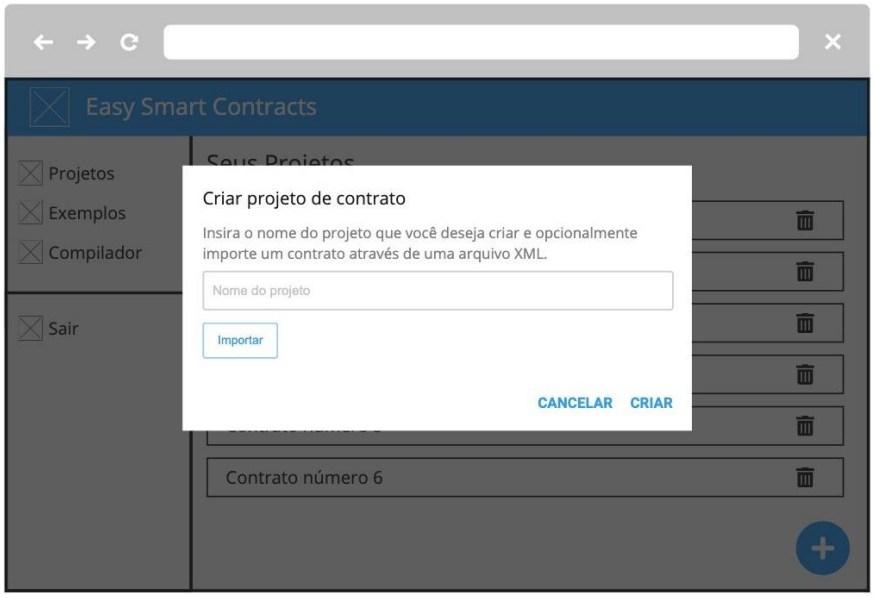


Figura 3. Modal de criação de projeto

Clicando no botão no canto inferior direito uma modal é aberta, como pode-se observar pela Figura 3, por onde o usuário pode escolher o nome do projeto que deseja criar e opcionalmente importar um arquivo de um projeto criado anteriormente, ao clicar no botão de criar o projeto é automaticamente adicionado à lista principal.

2.2.3. Tela de edição do projeto

A tela de edição do projeto é a principal tela do sistema, é por meio dela que os usuários podem desenvolver, salvar, compilar e implantar seus contratos inteligentes na rede Ethereum.

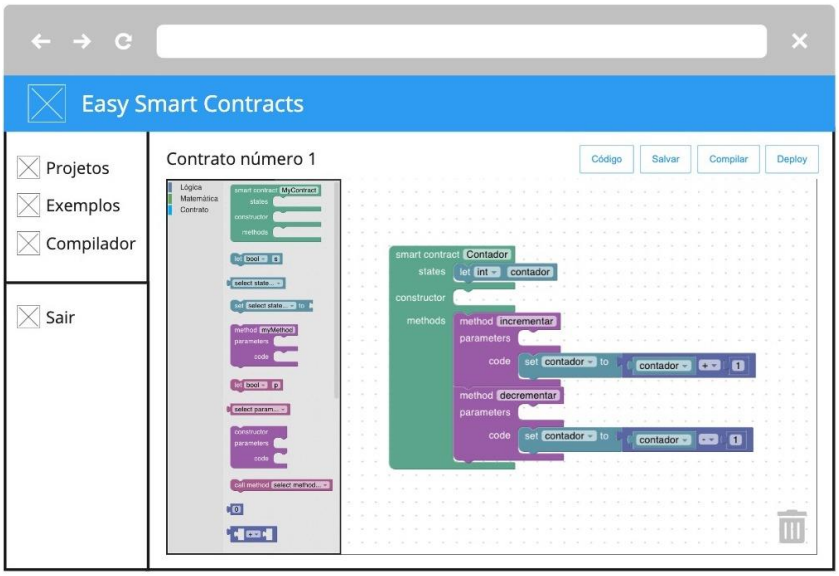


Figura 4. Tela do Editor

Como é possível observar na Figura 4, esta tela é composta por duas seções principais: um cabeçalho que mostra o nome do projeto que está sendo editado juntamente com quatro botões de ação e o editor, que é por onde o usuário pode desenvolver seu contrato inteligente utilizando blocos interligados que representam código Solidity.

O Editor possui um menu interno que permite a seleção de diversos blocos de código organizados em categorias, sendo elas, operações lógicas, operações matemáticas e operações relacionadas à entidade do contrato.

Uma vez que o usuário tenha desenvolvido seu contrato ele pode clicar no botão "código" para revelar uma modal contendo o código Solidity que foi produzido e o XML utilizado pela biblioteca Blockly para renderizar os blocos, como mostra a Figura 5.

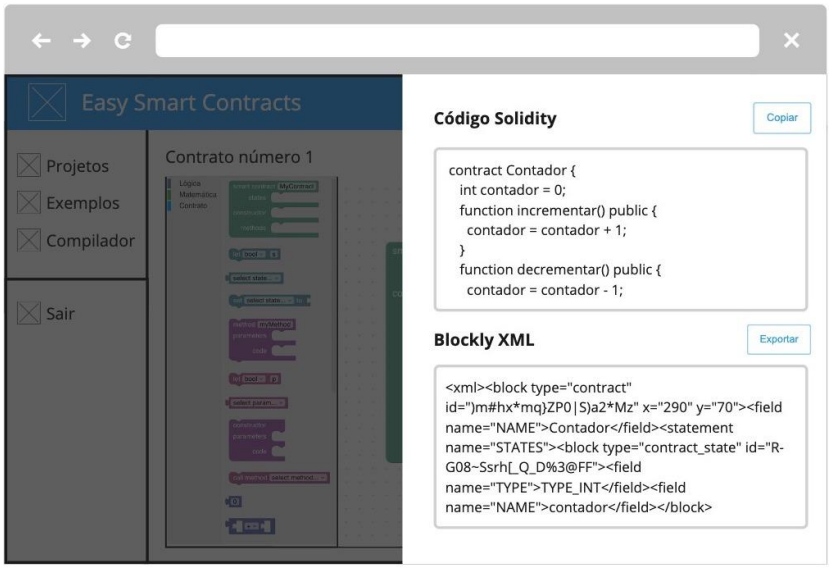


Figura 5. Modal de inspeção de código

Por meio dos botões localizados na parte superior do editor o usuário pode também salvar as alterações feitas no projeto, compilar o contrato desenvolvido, verificando eventuais erros que possam ter sido cometidos, assim como implementar o contrato na rede Ethereum através do MetaMask.

2.2.4. Tela do compilador

A Tela do compilador foca em usuários com um perfil mais avançado, que já tenham algum conhecimento a respeito de programação orientada a objetos e que desejam ter um maior controle sobre as linhas de código Solidity utilizados em seus contratos inteligentes.

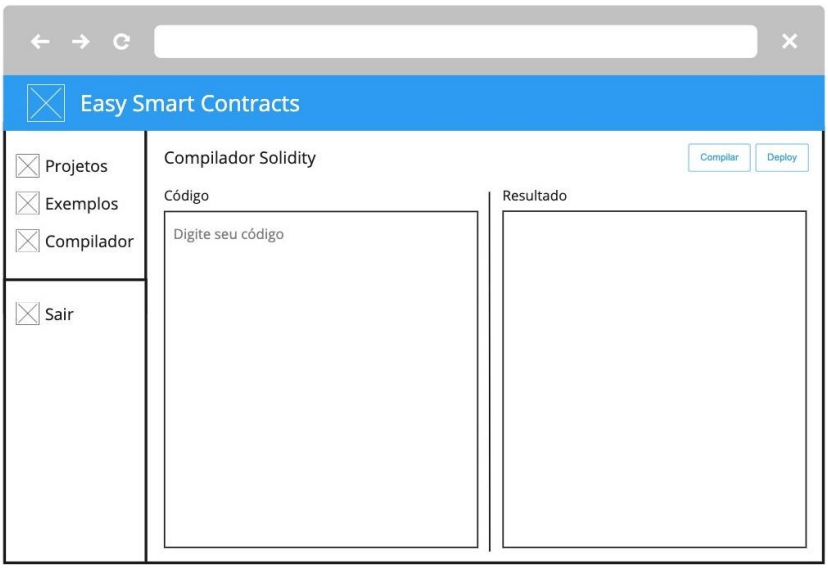


Figura 6. Tela do compilador

A interface desta tela é constituída por duas principais partes: um campo de texto a esquerda, onde o usuário pode escrever o código do contrato inteligente a ser criado e um bloco não editável a direita, no qual são mostrados os resultados detalhados da compilação dos contratos, como mostra a Figura 6.

Esta tela conta também com dois botões localizados no canto superior direito, que permitem ao usuário compilar e implantar contratos inteligentes na rede Ethereum. A principal vantagem do uso desta interface é a possibilidade do desenvolvimento de contratos que utilizem diretivas mais avançadas não suportadas pelo Editor de blocos. Além disso, a interface com acesso direto ao compilador serve como porta de entrada para o momento em que o usuário queira começar a escrever seus contratos diretamente em Solidity.

2.2.5. Tela de exemplos

A tela de exemplos tem um propósito puramente educativo e é direcionada a usuários que nunca tiveram o contato com contratos inteligentes, permitindo-os visualizar exemplos de contratos desenvolvidos dentro da plataforma e aprender sobre o desenvolvimento de código Solidity através de blocos interligados.

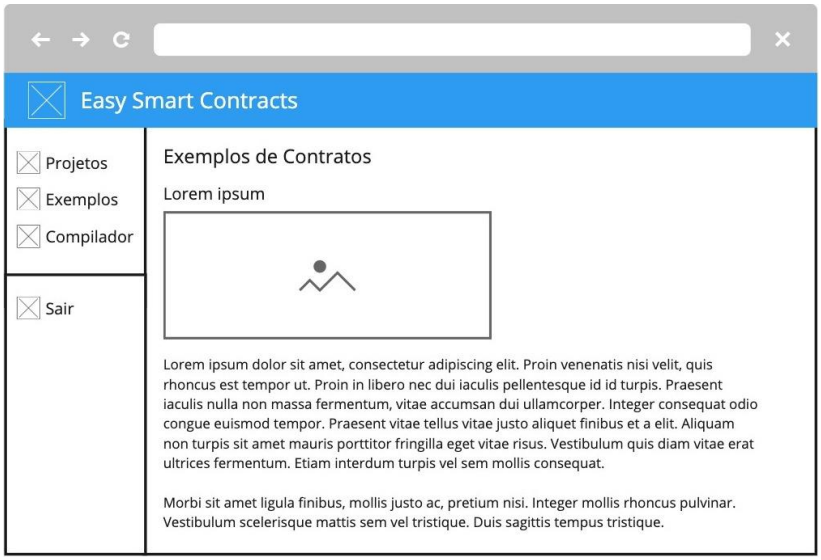


Figura 7. Tela de exemplos

A interface demonstrada na Figura 7 permite também que o usuário crie projetos de contratos inteligentes com o código demonstrado em cada um dos exemplos, tornando, desta forma, a experiência mais imersiva.

2.3. Arquitetura

A arquitetura do sistema foi projetada de forma a dar ênfase ao objeto de estudo do presente trabalho, os contratos inteligentes e as plataformas de desenvolvimento low-code. Por este motivo optou-se pelo uso de serviços como o Firebase Authentication, Cloud Firestore e o Firebase Hosting, que facilitam o gerenciamento da autenticação de usuários, o uso de bancos de dados flexíveis e escalonáveis para desenvolvimento focado em dispositivos Web e a hospedagem de conteúdo na Web, respectivamente.

Em relação ao desenvolvimento da interface da plataforma optou-se pelo uso do TypeScript, por ser uma linguagem de programação amplamente usada no desenvolvimento de sistemas e serviços web, além de possuir uma ótima integração com a biblioteca React. Como base para a plataforma low-code foi definida a utilização da biblioteca Blockly, pois esta já inclui as principais mecânicas esperadas de uma plataforma low-code, como blocos gráficos interligados que representam conceitos de código, bastando com que seja adicionado o suporte a linguagem de programação Solidity. Além disso, optou-se pelo uso da biblioteca solc-js para a compilação do código Solidity em bytecode e da integração com a extensão MetaMask para a implantação dos contratos inteligentes da rede Ethereum.

2.3.1. Modelagem do banco de dados

Ao optar-se pelo uso de um banco de dados NoSQL baseado em documentos como o Cloud Firestore, tem-se algumas vantagens frente a bancos de dados mais tradicionais.

Os bancos de dados baseados em documentos costumam trabalhar com documentos JavaScript Object Notation (JSON) ou semelhantes, armazenando dados em pares chave-valor. Em cada um dos documentos as chaves devem ser exclusivas e todo documento contém uma chave de identificação única dentro de uma coleção de documentos, que identifica um documento explicitamente. Desta forma estruturas de dados complexas, como objetos aninhados, podem ser manipuladas de forma mais conveniente (HECHT; JABLONSKI, 2011).

Outra característica de bancos de dados baseados em documentos é a não existência de restrições de esquema, o que torna possível o armazenamento de novos documentos com qualquer tipo de atributo, podendo ser feito tão facilmente quanto adicionar novos atributos a documentos existentes em tempo de execução.

Levando em consideração estes fatores e os RF definidos no começo do capítulo, o banco de dados da aplicação foi projetado de maneira simples, possuindo apenas uma principal coleção, chamada User, responsável por armazenar os dados de todos os usuários, como demonstra a Figura 8.

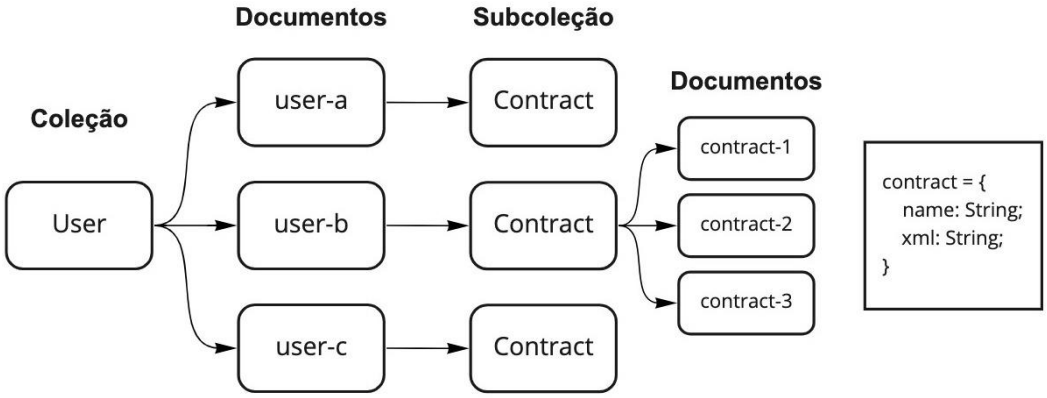


Figura 8. Diagrama da estrutura dos dados no Cloud Firestore

Cada documento pertencente à coleção User representa um usuário cadastrado na plataforma e por sua vez, possui uma subcoleção chamada Contract, que armazena quaisquer documentos de projetos de contratos inteligentes que o usuário possua. Todos os documentos dentro da subcoleção Contract seguem uma estrutura composta por dois campos: name, que representa o nome que o usuário deu ao projeto durante a etapa de criação e o campo xml que guarda em forma de texto a posição e a as interações de cada bloco de código do editor utilizado no projeto.

2.4. Implementação

Esta seção relata alguns dos pontos que se destacaram durante a etapa de implementação da Plataforma de Desenvolvimento de Smart Contracts Baseada em Interface Gráfica.

2.4.1. Cloud Firestore, Firebase Authentication e Firebase Hosting

A plataforma de desenvolvimento Firebase trabalha com um modelo conhecido como Backend as a Service (BaaS) e fornece um conjunto de ferramentas e serviços prontas para o uso, possibilitando ao desenvolvedor criar aplicações completas sem a necessidade de arquitetar sua própria infraestrutura, com servidores, load balancers, proxies, etc, ou ainda se preocupar com a escalabilidade da aplicação. Para o uso do Firebase é necessário optar entre um de seus dois planos: o Plano Spark ou o Plano Blaze. O Plano Blaze é voltado para aplicações de médio e grande porte, nele o usuário paga um valor baseado nas taxas de consumo de cada um dos serviços disponíveis, como por exemplo o número de leituras de documentos no banco de dados. Já o Plano Spark é totalmente gratuito e é comumente usado por startups e protótipos, tendo em vista que disponibiliza limites generosos de seus serviços sem custo algum.

Para o desenvolvimento da aplicação optou-se pelo uso do Plano Spark, levando em consideração o fato de ser gratuito e de possuir ótimos limites do serviço Cloud Firestore, permitindo o armazenamento de até 1 GiB de dados, 20 mil gravações e exclusões ao dia e 50 mil leituras diárias de documentos (FIREBASE, 2022), limite que ultrapassa em muito o consumo previsto no tocante ao número de usuários que farão uso da plataforma neste primeiro momento.

A plataforma desenvolvida no presente trabalho faz o uso de apenas três dos serviços disponíveis: Cloud Firestore, um banco de dados NoSQL baseado em documentos; Firebase Authentication, um serviço de autenticação com interface do usuário localizada, suporte a autenticação através de diversas plataformas como Google, Facebook e Twitter e integração nativa com o Cloud Firestore além do Firebase Hosting, um serviço de hospedagem de sites dinâmicos e estáticos oferecendo domínios customizados com certificados SSL.

Em relação a implantação do Cloud Firestore na plataforma foi necessário definir um conjunto de permissões para restringir o acesso dos usuários a documentos e coleções no banco de dados, tendo em vista que a interface do cliente possui acesso direto ao Software Development Kit (SDK) do Cloud Firestore e pode executar qualquer ação caso as permissões não tenham sido definidas.

```
1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      match /user/{userId}/{document=**} {
5        allow read,write: if request.auth.uid == userId;
6      }
7    }
8  }
```

Figura 9. Regras de segurança do Cloud Firestore

Conforme mostra a Figura 9, as regras estabelecidas pelo painel do Cloud Firestore permitem que apenas usuários autenticados tenham acesso ao seu próprio diretório, de forma a permitir a leitura e a escrita do documento e subcoleções dentro da coleção user que possuir o mesmo identificador que o usuário autenticado.

A plataforma desenvolvida utilizou também o Firebase Authentication, pois este permite a fácil integração com provedores de autenticação de terceiros. Foram configurados a autenticação por meio de contas do Google, Github e contras de e-mail como previsto nos RFs definidos em 2.1.1.

Em relação a interface de autenticação, foi utilizada a biblioteca FirebaseUI, disponibilizada pelo próprio Firebase, que implementa as melhores práticas recomendadas para autenticação em dispositivos móveis e sites, ajudando na conversão de login e no cadastro de novas contas. A biblioteca lida também com edge cases, tais como, a recuperação de contas e a vinculação automática de contas que possuam o mesmo endereço de email.

Para o uso do Firebase Hosting foi necessário a instalação da Command-line interface (CLI) do Firebase, ferramenta que permitiu a implantação de arquivos de diretórios locais nos servidores de Hosting do Firebase.

Por padrão todos os projetos do Firebase Hosting têm direito a utilização de subdomínios nos domínios web.app e firebaseapp.com sem custos financeiros, motivo pelo qual as versões de teste da plataforma desenvolvida foram distribuídas por meio do endereço web https://easy-smart-contracts.web.app/.

2.4.2. Suporte da linguagem Solidity a biblioteca Blockly

Blockly é uma biblioteca de código aberto desenvolvida pelo Google que possibilita a adição de um editor de código low-code a aplicações web. Através de blocos interligados, o editor Blockly permite a representação de diversos conceitos de código como variáveis, expressões lógicas entre outros, o que possibilita que o usuário construa seu código de maneira intuitiva e visual, sem ter que se preocupar com detalhes de sintaxe.

Do ponto de vista de um desenvolvedor, o Blockly é uma interface de usuário pronta para criar uma linguagem visual, capaz de emitir códigos gerados pelos usuários que sejam sempre sintaticamente corretos. (BLOCKLY, 2022)

Por padrão o Blockly exporta blocos para diversas linguagens de programação como JavaScript, Python, PHP, Lua e Dart, no entanto, a biblioteca não possui suporte nativo a linguagem de programação Solidity e, portanto, esta teve que ser adicionada manualmente.

O Blockly é uma biblioteca muito flexível e permite a extensão do suporte a outras linguagens de programação por meio de “geradores“. Para cada gerador de linguagem que tenha sido criado é necessário também a adição de geradores de código de bloco, que devem ser desenvolvidos para cada bloco que esta nova linguagem suportar.

Um gerador de uma linguagem define as propriedades básicas da linguagem, como o por exemplo o funcionamento da indentação, já os geradores de blocos definem como os blocos individuais são transformados em código e devem ser definidos para cada um dos blocos disponíveis ao usuário por meio do editor.

A linguagem de programação Solidity é uma linguagem de alto nível orientada a objetos utilizada para a implementação de contratos inteligentes, que nada mais são do que programas que regem o comportamento das contas existentes na blockchain do Ethereum.

A Solidity é uma linguagem de colchetes projetada para gerar código para a EVM e que foi influenciada e inspirada em várias linguagens de programação conhecidas, sendo mais profundamente influenciada por C++, porém, também emprega conceitos de linguagens como JavaScript, Python e outras (ETHEREUM, 2022).

A influência da linguagem C++ é nítida, podendo ser observada na sintaxe para declarações de variáveis, na utilização de laços, no conceito de sobrecarga de funções, conversões de tipos implícitos e explícitos e em muitos outros detalhes.

Além disso, a Solidity conta com uma ótima documentação em relação ao funcionamento da linguagem. Ela contém explicações a respeito da ordem de precedência de cada um dos seus operadores, detalha todas as variáveis globais suportadas, lista palavras reservadas, enumera seus modificadores, explica como funcionam os especificadores de visibilidade das funções, entre outros. O que foi essencial para a adição do suporte da Solidity ao Blockly.

Da mesma forma a biblioteca Blockly também dispõe de uma boa documentação, onde disponibiliza através do site Blockly Codelabs[1] uma série de tutoriais a respeito de como customizar a biblioteca, dando detalhes da criação de geradores customizados, de como personalizar a toolbox, entre outros.

Para adicionar o suporte da linguagem Solidity foi necessário a criação de um gerador de linguagem e diversos geradores de blocos de código. Durante a etapa de desenvolvimento foram utilizados como referência os geradores de linguagens já suportadas, em especial o da linguagem JavaScript.

Isso posto, foram implementados o suporte a algumas diretivas de linguagem Solidity para a biblioteca Blockly, entre elas: a declaração de entidades do tipo contract, incluindo o uso de construtores e métodos; estruturas de condição if then else; suporte a operações matemáticas de soma, subtração, multiplicação e divisão; suporte aos tipos bool, int e uint, além do suporte a declaração e atualização de variáveis e estados.

2.4.3. Compilador Solidity para Web

Para que se possa implantar um contrato inteligente é necessário primeiramente transformar o código Solidity para bytecode da EVM. Esta tarefa é realizada através de um compilador, que no caso da linguagem de programação Solidity se chama Solc.

[1] https://blocklycodelabs.dev/

Solc é originalmente um compilador desenvolvido em C++, porém, no campo das aplicações web ele é mais comumente encontrado em sua versão JavaScript chamada solc-js, uma vez que os navegadores não possuem suporte para código em C++.

O projeto solc-js nada mais é que uma derivação do compilador Solc utilizando Emscripten, desta forma ambos utilizam o mesmo código-fonte do compilador e tem suporte às mesmas funcionalidades. Emscripten é um compilador de código fonte C e C++ para WebAssembly que utiliza LLVM e Binaryen e possibilita que aplicações e bibliotecas escritas em linguagens C e C++ sejam compiladas antecipadamente e executadas de forma eficiente em navegadores da web e ambientes Node.js.

No entanto a tecnologia WebAssembly possui algumas limitações ao rodar na thread principal dos navegadores, dado que em navegadores baseados em chromium a diretiva WebAssembly.Compile pode apenas ser usada com buffers menores de 4KB, o que efetivamente impede que a plataforma desenvolvida no presente trabalho carregue o solc-js na thread principal.

Para contornar este problema foi necessário a utilização de WebWorkers, mecanismos que permitem que um dado script seja executado em uma thread diferente da thread principal da aplicação.

A thread principal dos navegadores e as threads dos WebWorkers comunicam se através de um sistema de mensagens que permite a troca de dados entre as diferentes threads, ambos os lados podem enviar mensagens usando o método postMessage() e respondem as mensagens através do manipulador de eventos onmessage (MDN, 2022). É importante ressaltar que os dados transmitidos entre as threads são copiados e não compartilhados, de maneira a evitar o acesso indevido de memória por ambas as partes.

Uma das principais vantagens ao fazer o uso de WebWorkers para a execução do solc-js no sistema desenvolvido consiste no fato de não ocorrer o bloqueio da interface do usuário no sistema durante a compilação de contratos, o que possibilita que estados de carregamento sejam mostrados ao usuário e assegura uma boa usabilidade.

A documentação da linguagem de programação Solidity recomenda que sempre sejam usadas as versões mais recentes da linguagem para a compilação e implantação de contratos, tendo em vista que a cada nova versão bugs são corrigidos e novos recursos são lançados.

As versões da Solidity seguem as especificações do Versionamento Semântico[2], sendo que as versões de nível de patch com versão principal 0, ou seja, 0.x.y, não conterão breaking changes, o que significa que um código que compila com a versão 0.x.y sempre poderá ser compilado com 0.x.z, onde z > y. Levando todos estes fatores em consideração a versão escolhida para o compilador foi a 0.8.6, por ser a versão mais atual no momento da elaboração do projeto desenvolvido no presente trabalho.

[2] https://semver.org/lang/pt-BR/

2.4.4. Web3 e a implantação de contratos inteligentes

Web3, também conhecida como Web 3.0 ou web descentralizada, foi um termo cunhado por Gavin Wood que propõe o fornecimento de serviços de internet distribuídos, sem a necessidade de agentes terceiros confiáveis, desta maneira oferecendo aos usuários um maior controle sobre seus dados.

Nos últimos anos, o termo Web3 tem feito grande sucesso no mercado de criptomoedas e nas comunidades relacionadas à blockchain, tornando-se o termo mais prevalente do setor. (WANG et al., 2022)

O princípio básico por trás de aplicações que utilizam Web3 é que os usuários podem manter o controle total sobre seus dados, ao invés de serem gerenciados por organizações centralizadas, como ocorre na Web1 e Web2. A Web3 move os dados para longe dessas autoridades centrais e da ênfase na descentralização, estabelecendo aplicativos e serviços em torno das blockchains.

Para que um software possa interagir com a blockchain Ethereum, seja lendo dados ou enviando transações para a rede, ele deve primeiro se conectar a um nodo Ethereum. Para que isso seja possível, todo cliente Ethereum implementa uma especificação JSON-RPC[3], fazendo com que exista um conjunto uniforme de métodos padronizados nos quais os aplicativos podem confiar.

O JSON-RPC é um protocolo de chamada de procedimento remoto, leve, sem estado e que define várias estruturas de dados e as regras em torno de seu processamento. O protocolo é agnóstico de transporte, e utiliza JSON como formato de dados. (WACKEROW, 2022)

A plataforma desenvolvida neste trabalho faz o uso da biblioteca web3.js, que é a API JavaScript do Ethereum que se conecta ao protocolo JSON-RPC. É por meio desta biblioteca que a aplicação interage com o ecossistema Ethereum e por onde os contratos inteligentes são implantados.

Tendo em vista que o Ethereum é na verdade um protocolo, podem haver várias redes independentes que estejam em conformidade com ele, cada uma contendo sua própria blockchain, o que efetivamente permite a existência de vários ambientes, como ambientes de desenvolvimento local, de testes e produção.

A rede chamada Mainnet é a principal rede de produção pública do Ethereum, sendo essa a blockchain que possui real valor monetário. Já as Testnets são redes de teste públicas que não tem valor real atrelado e são usadas por desenvolvedores para testar contratos inteligentes antes de serem implantados na Mainnet.

Durante os testes da aplicação desenvolvida foi utilizada a rede Rinkeby, uma Testnet que usa o conceito de proof-of-authority como mecanismo de consenso, onde um pequeno número de nodos é escolhido para validar transações e criar novos blocos. A rede Rinkeby dispõe de sites como o Rinkeby Faucet[4] que distribuem de maneira gratuita algumas frações de Ether diariamente, possibilitando que desenvolvedores testem seus contratos inteligentes sem custos.

[3] https://www.jsonrpc.org/specification
[4] https://rinkebyfaucet.com/

Para que os contratos possam ser implantados através da plataforma é necessário que o usuário faça a instalação de uma extensão para navegador como a MetaMask[5], uma ferramenta que permite interações do usuário com a Web3 e fornece uma carteira de criptomoedas no ecossistema Ethereum com suporte a tokens ERC-20.

A plataforma do presente trabalho foi projetada de forma que quando o usuário tenha terminado o desenvolvimento de seu contrato, seja por meio do editor Blockly presente na tela de edição do projeto (2.2.3) ou diretamente pela tela do compilador (2.2.4), ele possa implantar seu código com facilidade.

Quando o usuário clica em um botão na interface da plataforma para implantar o código do contrato produzido, este é enviado para um WebWorker, que compila o contrato utilizando solc-js. O resultado produzido pelo compilador é então retornado para thread principal, onde o sistema utiliza a biblioteca web3.js para criar a instância do contrato, fazendo o uso do Application Binary Interface (ABI) produzido pelo compilador. Em seguida, esta instância é implantada na rede através da transmissão de uma transação de criação de contrato inteligente contendo seu bytecode e a conta que criou o contrato.

2.4.5. Material UI e o Material Design

A interface da Plataforma de Desenvolvimento de Smart Contracts Baseada em Interface Gráfica foi inteiramente desenvolvida utilizando componentes da biblioteca Material UI[6] , que inclui uma coleção abrangente de componentes prontos para uso e implementa o Material Design do Google.

O Material é um sistema de design elaborado pelo Google e projetado para ajudar desenvolvedores a criar experiências digitais de alta qualidade para aplicações Android, iOS e web. De acordo com Davidov (2021), o "Material Design é o padrão para projetar e criar sites e aplicativos. Surge como uma resposta aos estilos de design antigos, hostis ao usuário e caóticos e seu objetivo é trazer ordem e união ao web design"

Em 2014 o Material Design foi anunciado como o estilo oficial de aplicativos produzidos pelo Google, como o Google Drive e o Gmail e a partir deste momento inúmeros desenvolvedores começaram a usar este estilo, o que contribuiu para seu amplo reconhecimento entre os usuários de aplicações web.

Levando em consideração esses fatores, o presente trabalho optou pela construção de uma interface que faz o uso do sistema de design Material a fim de propiciar ao usuário um ambiente familiar por onde ele possa navegar com facilidade e reconhecer ícones sem dificuldades.

[5] https://metamask.io/

[6] https://github.com/mui/material-ui

2.4.6. Telas desenvolvidos

Esta subseção é destinada para a exposição das interfaces desenvolvidas no presente trabalho, explicando o fluxo de navegação e de informação pelo sistema.

2.4.6.1. Tela de login

Ao acessar o sistema pela primeira vez o usuário é recebido através da tela de login, composta pelo nome, logo da aplicação e uma pequena descrição sobre o projeto.

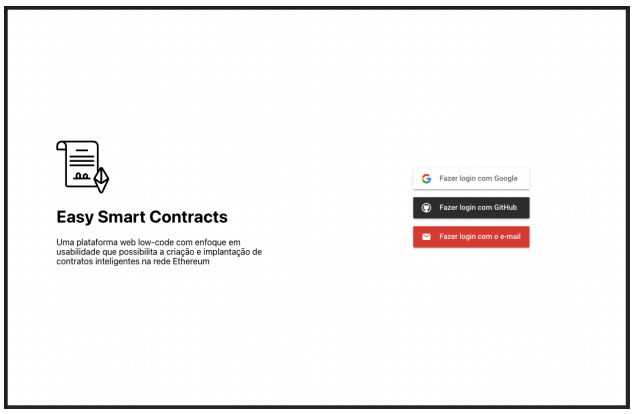


Figura 10. Tela de login

Esta tela conta também com três botões na lateral direita, por onde o usuário pode se autenticar utilizando provedores do Google, Github ou email, como mostra a Figura 10.

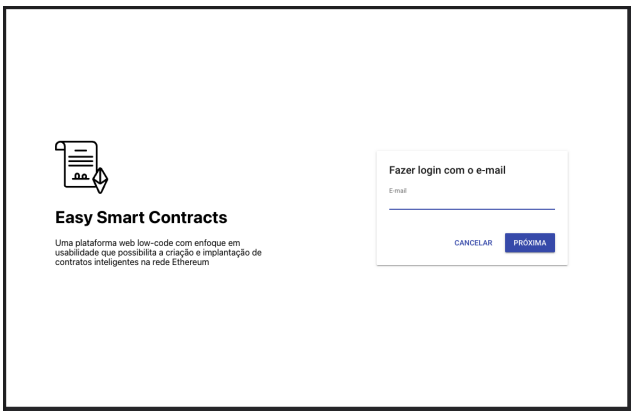


Figura 11. Tela de login - Fluxo de autenticação por email

No evento de o usuário ter selecionado o email como método de autenticação e este ainda não possuir cadastro na plataforma, um formulário é mostrado pedindo que o usuário forneça seu endereço de email, nome completo e defina uma senha.

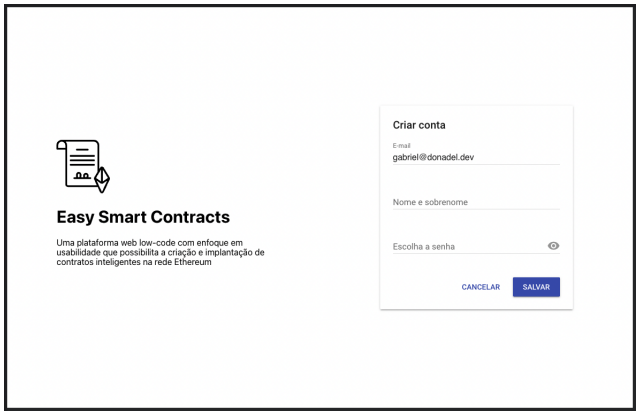


Figura 12. Tela de login - Fluxo de cadastro por email

Caso o usuário tente fazer login com um email que já esteja vinculado a outro método de autenticação, como o Github por exemplo, uma mensagem é mostrada pedindo ao usuário que realize o login utilizando o método pelo qual a conta foi cadastrada.

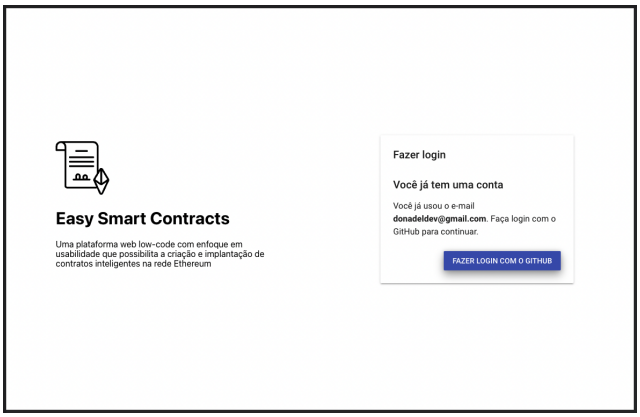


Figura 13. Tela de login - Fluxo de conta vinculado a outro provedor

2.4.6.2. Estrutura comum das telas autenticadas

Uma vez que o usuário tenha se autenticado com sucesso na plataforma, todas as telas em diante, com exceção da tela de login, seguirão uma estrutura padrão constituída por três partes principais:

- Um cabeçalho, localizado no topo da tela como representado na Figura 14, constituído do logo e nome da aplicação do lado esquerdo e ao lado direito um botão que leva o usuário ao repositório da plataforma no Github.

Figura 14. Cabeçalho

- Um menu navegacional localizado na lateral esquerda da página, que permite que o usuário alterne entre as telas de gerenciamento de projetos (2.4.6.3), exemplo (2.4.6.6) e do compilador (2.4.6.5).

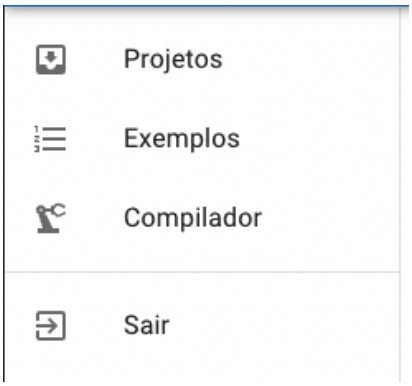


Figura 15. Menu navegacional

- E o conteúdo principal da tela, que varia conforme a rota selecionada e fica localizado ao centro.

2.4.6.3. Tela de gerenciamento de projetos

A tela de gerenciamento de projetos é a rota inicial para usuários autenticados e permite a visualização de todos os projetos de contratos inteligentes que o usuário possui.

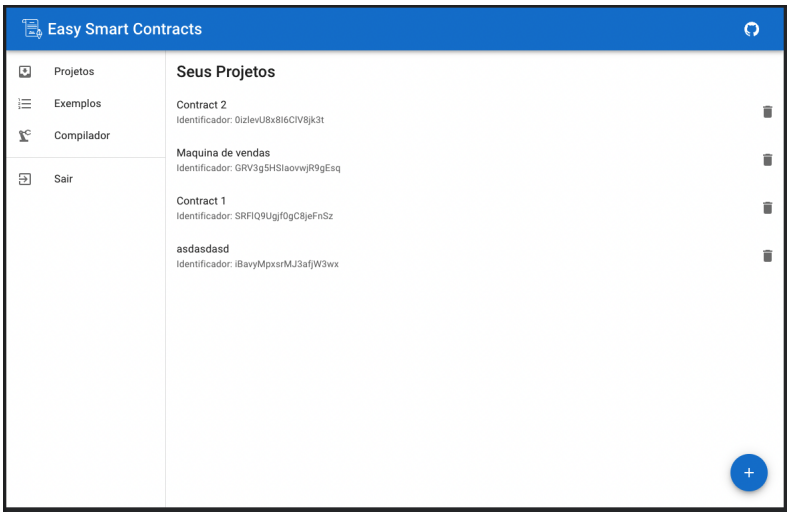


Figura 16. Tela de gerenciamento de projetos

Esta tela dispõe de um botão localizado no canto inferior direito que ao ser clicado abre uma modal que permite que o usuário crie e importe novos projetos, como mostra a Figura 17.

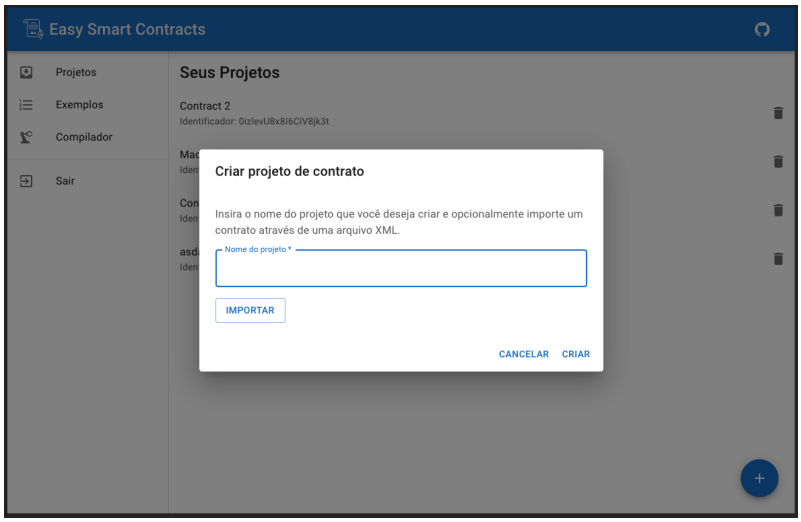


Figura 17. Tela de gerenciamento de projetos - Modal de criação

Além disso, cada um dos itens mostrados na lista de projetos possui um botão de deleção, que quando clicado mostra uma mensagem pedindo a confirmação do usuário de que ele realmente deseja deletar o projeto selecionado.

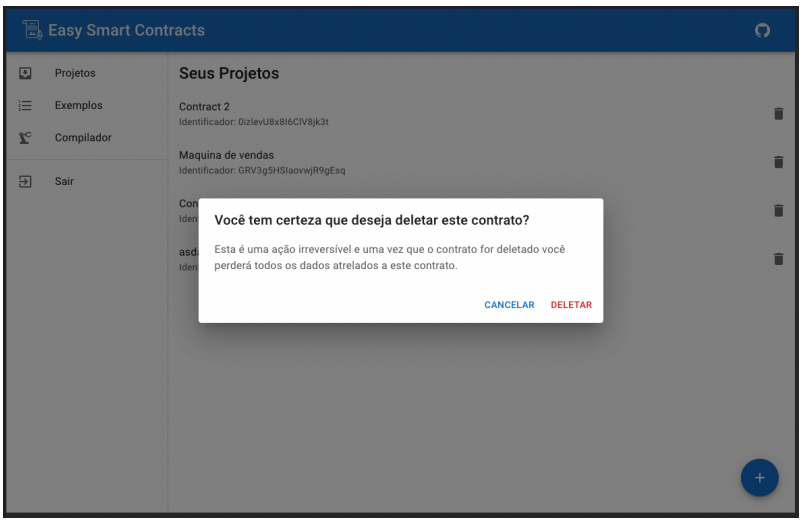


Figura 18. Tela de gerenciamento de projetos - Modal de deleção

Caso o usuário clique sobre um item da lista de projetos, este é redirecionado para a tela de edição do projeto (2.4.6.4).

2.4.6.4. Tela de edição do projeto

A tela de edição do projeto permite com que o usuário construa seus contratos inteligentes arrastando blocos dentro do editor.

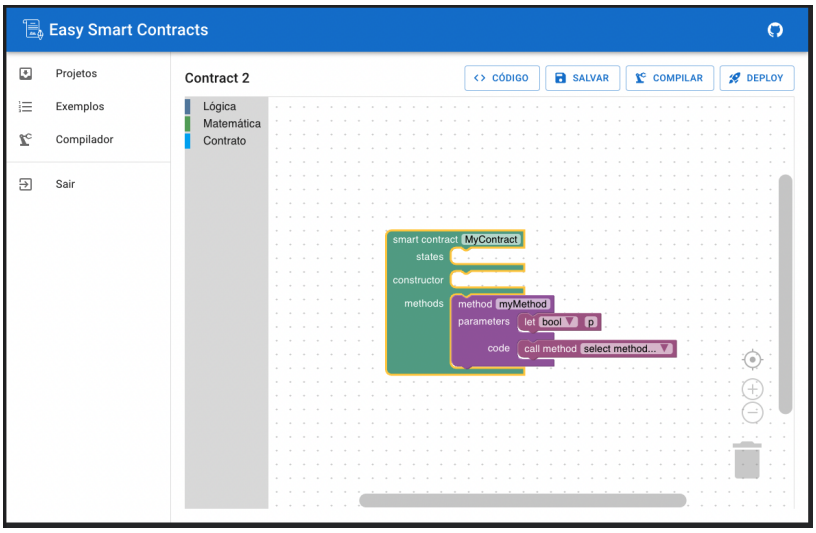


Figura 19. Tela de edição do projeto

A Figura 20 mostra alguns dos blocos que o usuário pode selecionar por meio da toolbox para compor seu contrato inteligente.

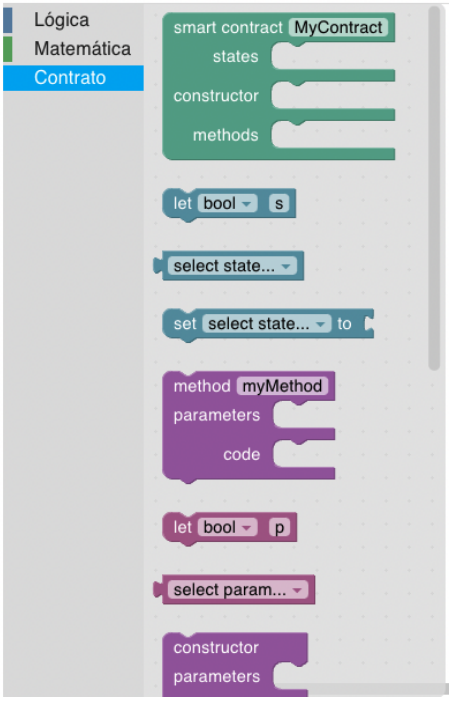


Figura 20. Tela de edição do projeto - Toolbox

Durante qualquer momento do desenvolvimento do contrato, o usuário pode clicar no botão de “Código“ para revelar uma modal contendo o código Solidity produzido pelo editor Blockly e o XML Blockly que pode ser usado para exportar o projeto.

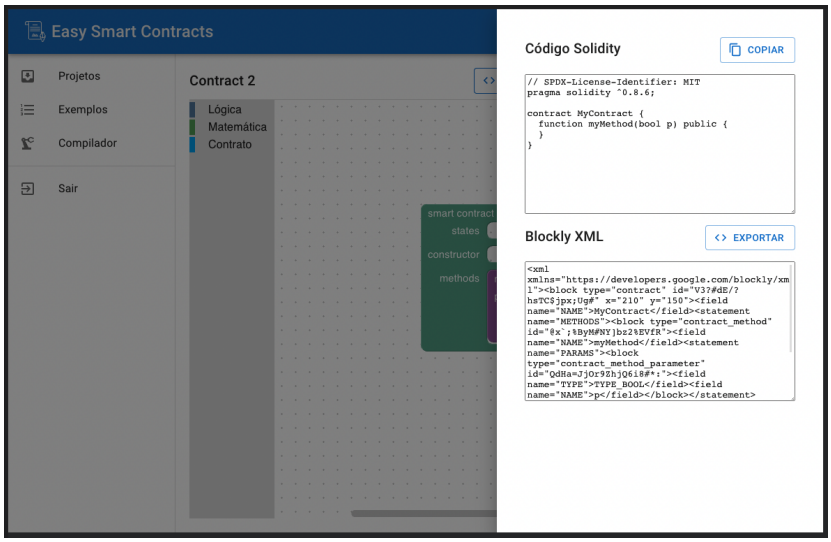


Figura 21. Tela de edição do projeto - Modal de inspeção de código

Quando o usuário termina de desenvolver seu contrato inteligente e deseja implantá-lo na rede, ele pode transmitir a transação de criação do contrato por meio do botão “Deploy“. Dessa maneira um popup da extensão MetaMask será mostrado na tela para a confirmação da transação, como mostra a Figura 22.

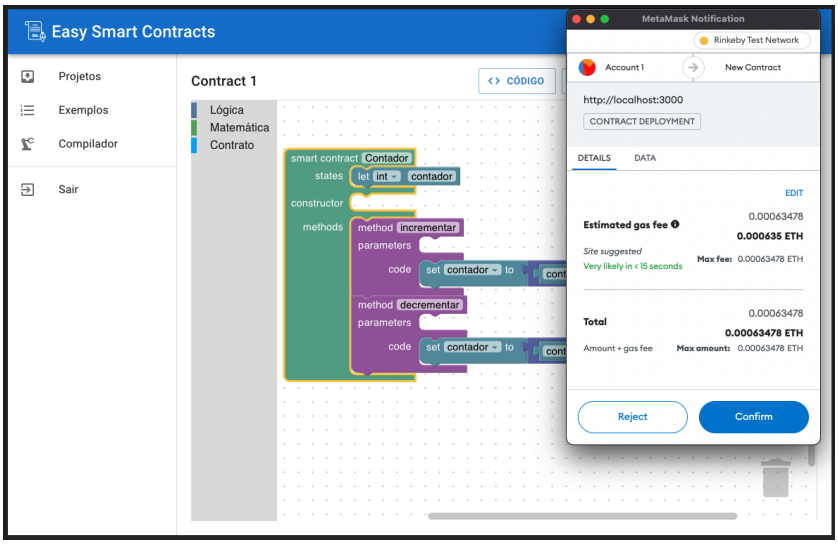


Figura 22. Tela de edição do projeto - Implantação do contrato via MetaMask

2.4.6.5. Tela do compilador

A tela do compilador é voltada para usuários avançados e por isso, mostra uma alerta recomendado o uso do editor Blockly para o desenvolvimento dos contratos. De forma semelhante a tela de edição do projeto (2.4.6.4), a tela do compilador permite que o usuário compile e implemente seus contratos inteligentes através de botões na interface.

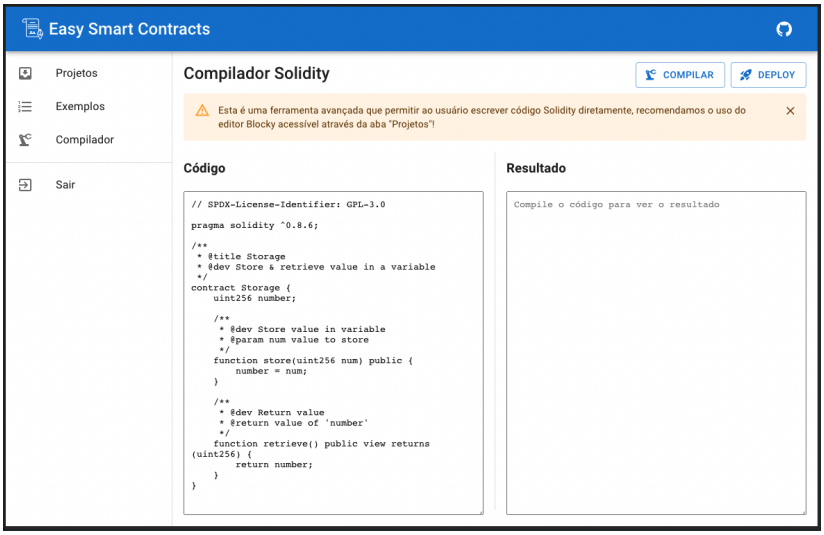


Figura 23. Tela do compilador

2.4.6.6. Tela de exemplos

Por último, a tela de exemplos permite que os usuários visualizem exemplos de contratos desenvolvidos dentro da plataforma e aprendam sobre o desenvolvimento de código Solidity através de blocos interligados.

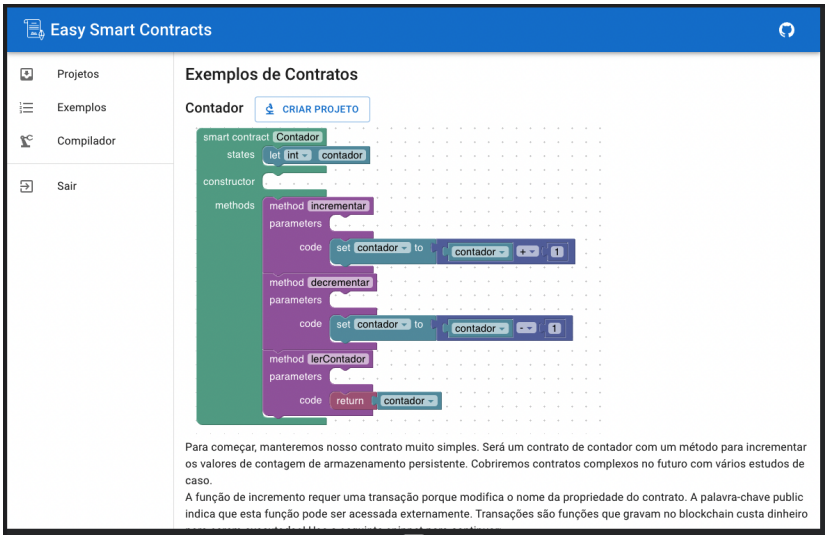


Figura 24. Tela de exemplos

3. Usabilidade da plataforma

A Norma NBR 9241-11 define o termo usabilidade como: “Medida na qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso.“ (ABNT, 2002) Onde eficácia: é a integridade com a qual o usuário consegue atingir determinados objetivos; eficiência: é a relação entre exatidão e integridade com o qual os usuários alcançam determinado objetivo e os recursos gastos para atingi-los e, satisfação: é o conforto dos usuários ao utilizar o sistema.

De forma parecida, Jakob Nielsen propôs dez princípios gerais para o design da interface do usuário, como mostra a Tabela 3, os quais denominou de “heurísticas“ pois, segundo ele, estes princípios estão mais na natureza das regras práticas do que nas diretrizes de usabilidade específicas (NIELSEN, 2005).

Tabela 3. Heurísticas de Nielsen

01	Visibilidade do status do sistema: O sistema deve sempre manter o usuário informado sobre o que está acontecendo.
02	Semelhança entre o sistema e o mundo real: O sistema deve seguir as convenções do mundo real, fazendo as informações aparecerem de forma lógica e natural.
03	Controle e liberdade: Deve existir a possibilidade do usuário sair do estado em que se encontra, ou retomar facilmente ao estado anterior.
04	Consistência e padrões: Seguir convenções, indicar ações iguais de maneira similar e utilizar o mesmo tipo de linguagem em toda a interface.
05	Prevenção de erros: Design que evite que problemas ocorram, além de boas mensagens de erro.
06	Reconhecimento ao invés de recordação: Utilizar símbolos com contexto e em lugares coerentes para que o usuário entenda o funcionamento de maneira fácil.
07	Flexibilidade e eficiência de utilização: O sistema deve atender a ambos os usuários, inexperientes e experientes.
08	Design minimalista: Mensagens de diálogos não devem conter informações irrelevantes. Informações a mais conflitam com a visibilidade.
09	Ajudar o reconhecimento, diagnóstico e recuperação de erros: Mensagens de erros devem ser claras e objetivas, devem indicar o problema com precisão e sugerir uma solução.
10	Ajuda e documentação: Qualquer informação deve ser fácil de pesquisar e deve ser focada na tarefa do usuário.

As heurísticas de Nielsen podem ser utilizadas como princípios para a avaliação da usabilidade de interfaces, podendo ser aplicadas em qualquer momento do projeto com o intuito de proporcionar uma ótima experiência aos usuários, fazendo com que apenas informações úteis sejam apresentadas, focando na fácil compreensão, interação simples e design minimalista.

Sendo assim, o presente trabalho realizou a análise da usabilidade da plataforma low-code desenvolvida, verificando o cumprimento de cada uma das dez Heurísticas de Nielsen.

A Heurística 01 diz respeito a visibilidade do status do sistema, requisitando que os usuários mantenham-se informados sobre o estado atual da aplicação, o que pode ser observado em diferentes partes da plataforma desenvolvida, como por exemplo, enquanto o usuário compila um contrato inteligente, o botão “compilar“ passa a mostrar um estado de carregando e depois uma mensagem de sucesso, caso o contrato tenha sido compilado corretamente, como mostra a Figura 25.

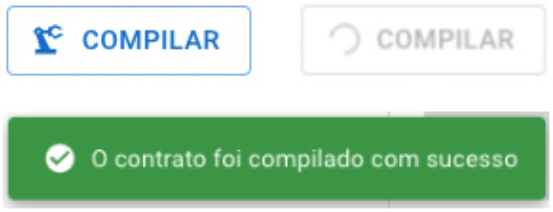


Figura 25. Feedback da interface ao compilar um contrato

A Heurística 02 trata da correspondência do mundo real com o sistema, cabendo a ele utilizar a linguagem que seus usuários estão familiarizados, seja no uso de frases, imagens ou conceitos. Um exemplo do cumprimento desta heurística é visto na tela de gerenciamento de projetos (2.4.6.3), onde é feito o uso do símbolo de lixeira para representar a ação de exclusão de projetos.

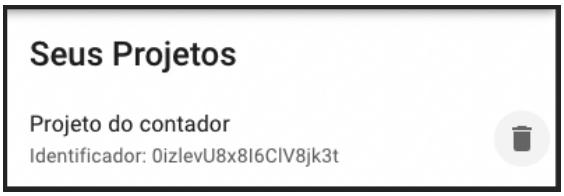


Figura 26. Lista de projetos de contratos

Já a Heurística 03 determina que para existir uma boa usabilidade, um sistema deve permitir que o usuário tenha o controle sobre a aplicação e possa realizar suas ações de forma livre, mas também possa reverter ações realizadas por engano. A plataforma também está em acordo com esta heurística, e uma das interfaces por onde isso pode ser observado é no editor de projetos(2.4.6.4), que permite que usuário desfaça suas ações utilizando o atalho “Ctrl+Z“.

Na Heurística 04 Nielsen pontua a necessidade da consistência entre as interfaces de uma sistema e a utilização de um padrão para que usuários usufruam da plataforma com facilidade. A adequação de tais fatores pode ser observada por todas a plataforma, tendo em vista que esta foi desenvolvida utilizando o Material Design, como explicado na subseção 2.4.5.

A Heurística 05 fala a respeito da construção de interfaces que previnam a ocorrência de erros por parte do usuário, um exemplo do cumprimento deste requisito está presente na tela de edição do projeto (2.4.6.4), pois ela previne que o usuário cometa erros no desenvolvimento dos contratos, dado que não permite que sejam conectados blocos incompatíveis e que possam gerar erros quando o código Solidity for compilado.

A Heurística 06 trata do fácil reconhecimento da interface ao invés da necessidade da recordação por parte do usuário, de forma que o sistema deva minimizar a quantidade de informações que o usuário precisa memorizar. O sistema desenvolvido no presente trabalho faz uso de um layout muito comum em aplicações web, contendo um cabeçalho no topo da página, um menu navegacional do lado esquerdo e o conteúdo principal ao centro, como recomendam as diretrizes do Material Design e por isso, também está de acordo com esta heurística.

A Heurística 07 estipula que o sistema deve atender tanto às necessidades dos usuários leigos quanto a dos experientes. A plataforma desenvolvida possui uma tela planejada especificamente para o uso de usuário experientes que é a tela do compilador (2.4.6.5), porém, a aplicação também permite que ambos os perfis de usuários desenvolvam contratos através da tela de edição do projeto (2.4.6.4), onde o usuário com um perfil avançado pode usar atalhos de teclado como, “Ctrl+C“, “Ctrl+V“, “Ctrl+Z“ e “Delete“, para acelerar o desenvolvimento do projeto.

A Heurística 08 estabelece que sistemas devem fazer o uso de um design minimalista, apresentando o conteúdo ao usuário de maneira direta. A plataforma desenvolvida utiliza uma linguagem simples e objetiva em suas interfaces, além de ser clara quando apresenta diálogos ao usuário.

A Heurística 09 diz respeito a clareza com que diálogos e mensagens de erros devem ser apresentadas ao usuário, de forma que este não fique em dúvida sobre qual erro aconteceu e que entenda como pode solucioná-lo. Um dos exemplos onde este requisito é posto em prática, no fluxo de cadastro de conta de email, como mostra a Figura 27, caso o usuário não preencha os dados necessários os campos com erro serão sinalizados em vermelho, apresentado pequenos textos que explicam ao usuário o que deve ser feito.

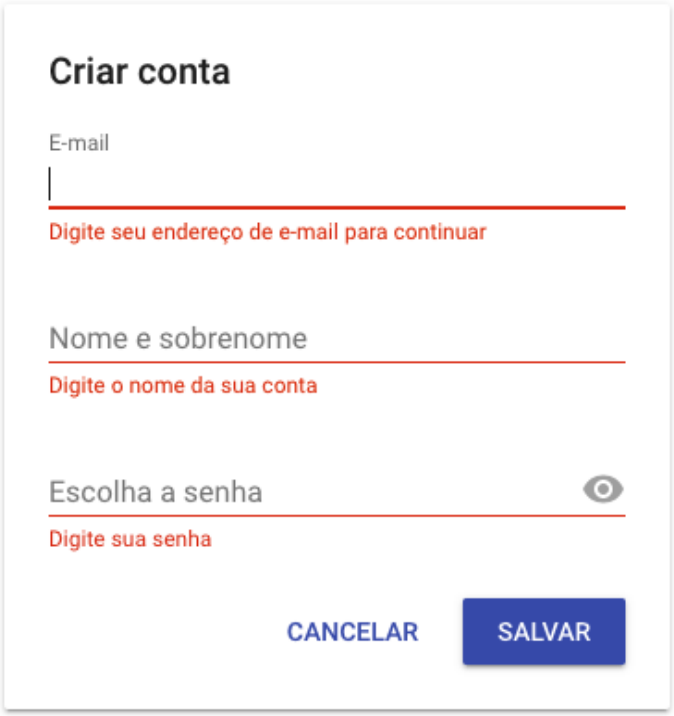


Figura 27. Estados de erro no formulário de cadastro de conta por email

Por fim, a Heurística 10 trata da disponibilização de alguma forma de documentação ou sistema de ajuda para o usuário, de maneira que este possa sanar eventuais dúvidas sobre o sistema. A aplicação desenvolvida possui uma interface dedicada a este este aspecto, como se vê na tela de exemplos (2.4.6.6), que explica de forma geral o funcionamento da plataforma e como o usuário pode desenvolver seus projetos.

Levando em consideração as 10 heurísticas de Jakob Nielsen e o fato do sistema desenvolvido no presente trabalho atender a todos seus dez princípios gerais, pode-se afirmar que a plataforma desenvolvida possui uma ótima usabilidade na visão de Nielsen.

3. Conclusão e Trabalhos Futuros

O presente trabalho teve como objetivos o reconhecimento na literatura especializada do estado da arte de plataformas low-code; o desenvolvimento de uma plataforma low-code para a criação de contratos inteligentes que pudessem ser implantados na rede Ethereum, além da avaliação de sua usabilidade.

Durante a elaboração deste estudo ficou evidente que a área da qual o trabalho trata ainda é muito incipiente e que as tecnologias abordadas estão sofrendo constantes atualizações, fazendo com que a literatura encontrada no meio acadêmico fique obsoleta rapidamente.

Para o desenvolvimento do trabalho foi necessário o uso extensivo das documentações da linguagem de programação Solidity e da biblioteca Blockly, pois as funcionalidades de ambas são melhoradas pela comunidade constantemente e suas documentações são os melhores canais pelos quais desenvolvedores podem ficar a par das mudanças.

Conclui-se que o estudo alcançou os objetivos gerais e específicos estipulados, uma vez que o estado da arte de plataformas low-code foi reconhecido, dando suporte para o desenvolvimento de uma plataforma web low-code que possibilita a criação e implantação de contratos inteligentes na rede Ethereum, por meio de uma interface amigável e intuitiva, sem a necessidade de conhecimento prévio em programação, além de ser realizada também a análise da usabilidade da plataforma fundamentada nas dez heurísticas de Jakob Nielsen.

Ganham destaque como facilitadores do desenvolvimento do trabalho, a biblioteca Blockly, a qual forneceu a interface utilizada no editor de contratos e proporcionou que o presente trabalho não tivesse que desenvolver um editor low-code do zero e o conjunto de soluções oferecidas pelo Firebase, tendo em vista que estas encapsularam funcionalidades que demandariam o desenvolvimento de um servidor externo para tratar da autenticação e dos dados do usuário.

O advento dos contratos inteligentes em ecossistemas que utilizam a tecnologia blockchain ocorreu a menos de dez anos e ainda passa por um processo contínuo de maturação, tornando esse um dos fatores limitantes do presente estudo, porém, esta lacuna serve como impulsionadora para trabalhos futuros, que poderão dar continuidade a plataforma desenvolvida.

Sendo assim, sugere-se o prosseguimento do presente trabalho dos seguintes pontos:

- Extensão do suporte a linguagem Solidity a biblioteca Blockly, adicionando o conceito de funções públicas, privadas, internas e externas; suporte a diretiva de import; adição do restante dos tipos suportados pela linguagem e a possibilidade do usuários poder escolher qual versão da linguagem desejam usar.
- Desenvolvimento de um mecanismo que permita a interação com os métodos dos contratos inteligentes implantados através da interface da plataforma.
- E a realização de uma pesquisa de campo validando a eficiência, eficácia e satisfação do usuário ao utilizar a plataforma.

Por fim, a plataforma web low-code desenvolvida no presente trabalho garante que de forma amigável e intuitiva possam ser criados e implementados contratos inteligentes na rede Ethereum, independentemente do nível de conhecimento do usuário.

Referência

Blockly. (2022). Introduction to Blockly. [S.l.: s.n.]

Destefanis, G. (2021, March). Design patterns for smart contract in ethereum. In 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C) (pp. 121-122). IEEE.

Ethereum. (2022). Language Influences — Solidity 0.8.15 documentation. [S.l.: s.n.]

Firebase (2022). Firebase Pricing. [S.l.: s.n.]

Hecht, R., & Jablonski, S. (2011, December). NoSQL evaluation: A use case oriented survey. In 2011 International Conference on Cloud and Service Computing (pp. 336-341). IEEE.

Larman, C. (2000). Utilizando UML e padrões. Bookman Editora.

MDN. (2022). Web Workers API - APIs da Web. [S.l.: s.n.]

NBR, A. (2002). 9241-11: requisitos ergonômicos para trabalho de escritório com computador–Parte 11–orientações sobre usabilidade. Rio de Janeiro, 9241-11.

Nielsen, J. (2005). Ten usability heuristics.

Rosemberg, C., Schilling, A., Bastos, C., & Araripe, R. (2008). Prototipação de software e design participativo: uma experiência do atlântico. IHC, 8, 312-315.

Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020, August). Supporting the understanding and comparison of low-code development platforms. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 171-178). IEEE.

Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., & Vechev, M. (2018, October). Securify: Practical security analysis of smart contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 67-82).

Vadgama, N., & Tasca, P. (2021). An analysis of blockchain adoption in supply chains between 2010 and 2020. Frontiers in Blockchain, 4, 610476.

Wang, Q., Li, R., Wang, Q., Chen, S., Ryan, M., & Hardjono, T. (2022). Exploring Web3 From the View of Blockchain. arXiv preprint arXiv:2206.08821.

Wackerow, P. (2022). JSON-RPC API. [S.l.: s.n.]

APÊNDICE B – CÓDIGO FONTE

Todo o código fonte desenvolvido para a Plataforma de Desenvolvimento de *Smart Contracts* Baseada em Interface Gráfica durante este trabalho de conclusão do curso de graduação em Sistemas de Informação está disponível no sistema de arquivos da Universidade Federal de Santa Catarina e pode ser baixado por meio do link `https://arquivos.ufsc.br/f/a6b8f061aeab454ea6ae/` onde o código está compactado em um arquivo *.zip*. Além disso, o código também está disponível através da plataforma Github no link `https://github.com/gabrieldonadel/easy-smart-contracts`.