Desenvolvimento de um sistema de recomendação de Issues para contribuições em projetos Open Source

Douglas Diehl Lutz[1], Gustavo Stangherlin Cantarelli[1]

[1]Curso de Ciência da Computação – Universidade Franciscana (UFN)
Caixa Postal 97010-032 – Santa Maria – RS – Brazil

{douglas.lutz,gus.cant}@ufn.edu.br

Abstract. *This article presents a proposal of a web application development to help developers in contributing in open-source projects of GitHub's repository, developers will be able to receive issue recommendations on projects that need help. The recommendations system will be using the user's information obtained in its profile and also information about the open-source projects obtained in GitHub's API. For the development of this project, the adopted agile development methodology was Feature-Driven Development. The application was developed with the Elixir programming language and Phoenix web framework, and was also based on automated tests with the Test-Driven Development standard.*

Resumo. *Este trabalho apresenta o desenvolvimento de uma aplicação web para auxiliar desenvolvedores na contribuição em projetos open-source da plataforma GitHub, onde eles podem receber recomendações de issues em projetos que precisam de ajuda. O sistema faz recomendações com base no perfil do usuário que se inscrever na plataforma e com base nas informações obtidas sobre projetos presentes no GitHub a partir de sua API. Para o desenvolvimento desse projeto, foi adotada a metodologia de desenvolvimento ágil Feature-Driven Development. A aplicação foi desenvolvida na linguagem funcional Elixir com auxílio de seu framework web Phoenix e com o desenvolvimento baseado em testes automatizados com o padrão Test-Driven Development.*

1. Introdução

Ao longo das duas primeiras décadas do século XXI, o desenvolvimento de software *open-source* (código aberto) foi atraindo cada vez mais atenção. A rápida evolução de projetos de software *open-source* (OSS), como o sistema operacional Linux, *browser* Mozilla, IDE Eclipse, entre tantos outros, faz com que empresas tradicionais procurem meios de aproveitar o potencial do desenvolvimento de *open-source* em suas atividades de desenvolvimento de software.

O ecossistema de projetos *open-source* está mudando a forma que projetos de software funcionam. Os benefícios que o *open-source* traz vêm de diversas formas, facilita a tarefa de integrar novas tecnologias mantendo sistemas sempre atualizados, promove uma colaboração mais eficaz tanto dentro das empresas quanto nas

comunidades de desenvolvimento e, também, fornece agilidade na hora de acrescentar novas funcionalidades a um sistema.

Os desenvolvedores que contribuem para OSS também ganham muito com essa prática. É possível aumentar exponencialmente o seu conhecimento por expô-lo a um projeto maduro e que foi feito por vários desenvolvedores experientes, como também vêm sendo cada vez mais comum as empresas pedirem o link de perfil do GitHub para verificarem projetos pessoais e contribuições em outros projetos.

A contribuição da comunidade para projetos *open-source* é fundamental para a existência deles. A plataforma GitHub fornece um lugar para o armazenamento desses projetos, enquanto as solicitações de modificações e melhorias são feitas por meio de *issues*[1] na plataforma [GitHub 2020a]. Dessa maneira, cabe ao programador buscar projetos e *issues* para contribuir, tarefa que muitas vezes não é tão simples, visto que existe uma quantidade muito grande deles.

1.1. Justificativa

Percebeu-se a inexistência de uma ferramenta que facilite a vida do programador na hora da escolha de uma *issue* para contribuir e que também aumente a visibilidade de projetos de código aberto. Constatou-se que os sistemas atuais não suprimem a necessidade dos desenvolvedores de software com grande desatualização no aspecto tecnológico das ferramentas existentes, assim como ausência de funcionalidades pertinentes ao assunto. Com base nisso, o presente trabalho se propõe a desenvolver um sistema que ofereça ao usuário uma filtragem e recomendação de *issues* no GitHub, para que desenvolvedores não percam tempo e esforço nesta busca, otimizando, então, seu tempo e esforço apenas na resolução de *issues*.

1.2. Objetivo Geral

O objetivo deste trabalho foi o desenvolvimento de um sistema web que facilite o processo de seleção feito por desenvolvedores para escolher funcionalidades e melhorias a serem implementadas em projetos de código aberto. O sistema fará o carregamento de *issues* ativas no GitHub e, com base tanto no perfil de cada usuário quanto nas informações de cada uma das *issues* e, com a utilização de um algoritmo de recomendação, apresentará ao usuário as *issues* mais apropriadas.

1.3. Objetivos Específicos

Os objetivos específicos deste trabalho foram:

- Pesquisar, estudar e utilizar algoritmos e métodos de recomendação;
- Estudar e utilizar a API do GitHub;
- Utilizar o sistema de versionamento de código Git para o desenvolvimento do sistema;

---

[1] Uma *issue* é uma postagem contida no *Issue Tracker* do GitHub. *Issues* são usadas como uma forma de delegação de tarefas e informação de bugs. Possui um autor, título, número e descrição e pode ser atribuída a um dos desenvolvedores daquele repositório.

- Utilizar a linguagem Elixir juntamente com o seu *framework* web Phoenix e o SGBD (Sistema de Gerenciamento de Banco de Dados) PostgreSQL;
- Utilizar a metodologia FDD (*Feature Driven Development*) no desenvolvimento do projeto e a técnica de desenvolvimento TDD (*Test Driven Development*) no desenvolvimento do sistema.

2. Revisão Bibliográfica

Esta seção apresentará os conceitos necessários para o desenvolvimento deste trabalho, bem como quais as tecnologias que foram utilizadas para realizar o que foi proposto. Serão apresentadas as concepções de sistemas de recomendações e suas técnicas, GitHub e sua API, Elixir, Phoenix e PostgreSQL, sendo, assim, possível ter uma melhor compreensão das tecnologias e técnicas utilizadas para a elaboração da aplicação.

2.1. Sistemas de Recomendação

Pessoas geralmente confiam nas recomendações fornecidas por outros na tomada de decisões rotineiras e diárias. Contudo, com o crescimento explosivo e a variedade de informações disponíveis na web e a rápida introdução de novos serviços de *e-business*, os usuários acabam sobrecarregados de opções, levando-os a tomar más decisões. Dessa forma, na tentativa de solucionar esses problemas, os primeiros Sistemas de Recomendação (SR) aplicaram algoritmos para aproveitar as recomendações já existentes de usuários a fim de fornecer novas recomendações a um outro usuário [Ricci *et al.* 2011].

Sistemas de Recomendação são definidos como ferramentas e técnicas de software que fornecem sugestões de itens que podem ser usados por um usuário. As sugestões estão relacionadas a vários processos de tomada de decisão, como quais itens comprar, que música ouvir ou quais notícias ler on-line. Normalmente, um SR se concentra em apenas um tipo específico de item e, consequentemente, seu design, sua interface gráfica e a principal técnica de recomendação usada para gerar as recomendações são todos personalizados para fornecer sugestões úteis e eficazes para esse item específico [Ricci *et al.* 2011].

2.1.1. Técnicas de Recomendação

Os Sistemas de Recomendação funcionam com a aplicação de técnicas específicas. Algumas podem usar dados simples, como avaliações de usuários em itens, e outras, como ontologias, são mais dependentes do conhecimento e, por isso, mais complexas [Ricci *et al.* 2011].

Algumas técnicas são mais consolidadas, dentre elas pode-se citar a recomendação baseada em conteúdo, a filtragem colaborativa, a recomendação demográfica, a recomendação baseada em conhecimento, entre outras [Ricci *et al.* 2011].

Este trabalho propõe a combinação de mais de um método de recomendação no sistema, sendo necessário para complementar técnicas que podem apresentar problemas em determinadas situações ou para oferecer resultados melhores aos usuários. A

combinação de filtragem de informações e do método baseado em conhecimento fornece ferramentas que se complementam e solucionam o problema proposto.

2.1.2. Filtragem de informação

A filtragem de informação é o nome utilizado para descrever uma variedade de processos que envolvem a entrega de informação para as pessoas que realmente necessitam delas. Essa abordagem mantém um perfil dos interesses do usuário e, com esse perfil, é possível descartar itens não relacionados. Essa filtragem deve ser aplicada a cada novo item adicionado, procurando verificar se este atende ao usuário [Reategui e Cazella 2005].

2.1.3. Método baseado em Conhecimento

Sistemas de recomendação baseados em conhecimento recomendam itens com base no conhecimento específico do domínio sobre como determinados atributos do item atendem às necessidades e preferências dos usuários e, finalmente, quanto o item é útil para o usuário. Nesses sistemas, uma função de similaridade é estimada, a qual pode ser diretamente interpretada como a utilidade da recomendação para o usuário [Ricci *et al.* 2011].

Os sistemas baseados em conhecimento tendem a funcionar melhor que outros no início de sua implementação, no entanto, se não estiverem equipados com componentes de aprendizado, podem ser superados por outros métodos que podem explorar o histórico das recomendações [Ricci *et al.* 2011].

2.2. GitHub

O GitHub é uma plataforma de desenvolvimento onde é possível fazer a hospedagem de código em repositórios abertos ou privados, possibilitando, com isso, que desenvolvedores trabalharem em conjunto mantendo o código em um só lugar. O site oferece suporte para projetos pessoais, projetos *open-source* e projetos empresariais com ferramentas para cada tipo [GitHub 2020a].

Issue é uma publicação que pode ser criada em um repositório do GitHub. *Issues* são utilizadas para sugerir uma nova funcionalidade ou reportar um erro no código, deixando organizada, então, a lista de tarefas que serão implementadas no projeto [GitHub 2020a].

2.2.1. GitHub API

O GitHub fornece uma *Application Programming Interface* (API), uma interface da aplicação que funciona como meio de comunicação com outros softwares externos, onde são disponibilizadas informações sobre repositórios abertos, como contribuidores, *issues*, *forks*, número de estrelas, etc. [GitHub 2020b].

2.3. Tecnologias

Esta seção apresentará as linguagens de programação, *frameworks* e bibliotecas que serão utilizadas para a implementação do software proposto.

2.3.1. Elixir

Elixir é uma linguagem de programação funcional, é utilizada para a construção de aplicações escaláveis e de fácil manutenção. O Elixir é executado na máquina virtual Erlang, dando aos desenvolvedores o acesso completo a todo o ecossistema Erlang, que é conhecido por executar sistemas de baixa latência, distribuídos e tolerantes a falhas. Ao mesmo tempo é muito utilizado em desenvolvimento web, softwares embarcados e processamento de multimídia [Elixir, 2020].

2.3.2. Phoenix

O Phoenix é um framework web que implementa a arquitetura MVC (Model, View, Controller) e fornece estruturas padrão para o banco de dados da aplicação e suas páginas da web [Phoenix, 2020a].

Muitos de seus componentes e conceitos são semelhantes aos que são utilizados em outros frameworks web como Ruby on Rails ou Django do Python. O Phoenix busca oferecer alta produtividade do desenvolvedor e alto desempenho para a aplicação [Phoenix, 2020b].

2.3.3. Padrão MVC (*Model, View, Controller*)

MVC é uma arquitetura de software que faz com que o acoplamento entre classes seja reduzido, aumentando, assim, a coesão delas. Essa arquitetura torna o código mais independente para que possa ser facilmente reutilizado, economizando tempo e esforço em desenvolvimentos futuros [Carrillo *et al*. 2005].

O MVC tem sido amplamente implementado nas aplicações web para separar a entidade responsável por mostrar as informações (*view*), a responsável por armazená-las (*model*) e a responsável por receber os eventos do usuário (*controller*) [Carrillo *et al*. 2005].

O *controller* é responsável por controlar todas as interações com o usuário. Sempre que a aplicação recebe uma requisição (*request*) do usuário, o controlador precisa decidir o que fazer em seguida e qual será a resposta (*response*) ao usuário, essa é a parte que gerencia o controle de fluxo global do aplicativo. Enquanto isso, os *models* são as classes da aplicação responsáveis pela manutenção dos dados e interação com o banco de dados. E, a *view* é responsável pela interface da aplicação, nesse caso, é o código HTML e JavaScript que é enviado ao cliente pelo *controller* [Carrillo *et al*. 2005].

2.3.4. *Front-end*

Front-end trata-se das interfaces com que o usuário interage na aplicação. Para a construção do mesmo, serão utilizadas as linguagens HTML, CSS e JavaScript com o auxílio da biblioteca Bootstrap.

HTML é uma linguagem de marcação que define a estrutura de um site na internet, o navegador interpreta o conteúdo de um arquivo HTML e mostra essa informação para o usuário [WHATWG 2020]. Todavia, o HTML possui limitações como a estilização, possibilitada apenas com CSS, que é o mecanismo de estilização utilizado nos sites da internet [W3C 2020].

O Bootstrap é uma biblioteca *open-source* criada pelos desenvolvedores do Twitter. Tem como objetivo trazer diversas estilizações e comportamentos para componentes a fim de deixar as interfaces da aplicação responsivas e com melhor aspecto visual [Bootstrap 2020].

2.3.5. PostgreSQL

O PostgreSQL é um SGBD (Sistema Gerenciador de Banco de Dados) relacional *open-source*. Pelo fato de ser relacional, faz o armazenamento dos dados em tabelas separadas, proporcionando boa performance e flexibilidade. Além do mais, faz o uso da linguagem padrão SQL (*Structured Query Language* -- Linguagem de Consulta Estruturada), geralmente utilizada para criar tabelas, acessar e manipular conteúdos armazenados no banco de dados [PostgreSQL 2020].

O PostgreSQL ganhou uma forte reputação por sua arquitetura comprovada, confiabilidade, integridade de dados, conjunto robusto de recursos, extensibilidade e dedicação da comunidade *open-source* por trás do software para fornecer consistentemente soluções inovadoras e de alto desempenho. Devido a isso, se tornou o banco de dados relacional preferido por muitas pessoas e organizações [PostgreSQL 2020].

2.4. *Feature-Driven Development*

Feature-Driven Development é uma metodologia de desenvolvimento de software que contém características de processos ágeis, como foco na programação, interação constante com o cliente e entrega frequente de versão do produto, assim como inclui alguns benefícios de processos rigorosos, como modelagem, planejamento prévio e controle do projeto. Também prevê práticas apenas para o desenvolvimento de software em si, não se preocupando com outros fatores, como a escolha de tecnologias e ferramentas [Silva et al. 2009].

A metodologia propõe alguns processos para o desenvolvimento de uma aplicação, são eles: desenvolver um modelo abrangente, construir uma lista de funcionalidades, planejar através de funcionalidades, projetar através de funcionalidades e construir através de funcionalidades. Cada processo tem sua importância no momento do desenvolvimento e a metodologia dá suporte e abrange todos esses cinco processos [Silva et al. 2009].

2.5. *Test-Driven Development* (TDD)

Test-Driven Development é uma forma de conduzir a programação de um software com testes automatizados, desse modo, apenas duas regras são necessárias para produzir um software com código limpo, que são:

- Escrever código novo apenas se um teste falhar;
- Eliminar duplicação.

Essas regras resultam em diversos benefícios para o software e desenvolvedores que estão trabalhando no mesmo. O TDD faz com que o código fique coeso, sem duplicação, com o mínimo possível de acoplação entre componentes e o ambiente de

desenvolvimento necessita fornecer resposta rápida a pequenas mudanças para verificar se nenhum teste falha [Beck 2003].

3. Trabalhos Correlatos

Nesta seção, são apresentados os trabalhos correlatos que foram utilizados como base para o desenvolvimento deste trabalho. Eles foram escolhidos utilizando como critério a data de produção e semelhança com o presente trabalho. Os trabalhos, em questão, mostram como funcionam e quais são as principais técnicas de recomendação, assim como uma proposta semelhante de software que já existe no mercado e, por fim, a comparação entre as duas arquiteturas em que o GitHub possui suas APIs.

3.1. Técnicas de Recomendação para usuários de Bibliotecas Digitais

O trabalho de Furlan et al. (2018) apresenta o desenvolvimento de um sistema de recomendação de artigos científicos para pesquisadores. O trabalho faz a combinação das técnicas de recomendação baseada em conhecimento e colaborativa, e a busca de produções científicas é feita com a utilização do conceito de web Crawler no motor de buscas Google Acadêmico.

3.2. Software CodeTriage: Envio de *issues* ativas para usuários

CodeTriage [Schneeman 2020] é um software de código aberto que possui uma proposta semelhante à do presente trabalho. O software faz o envio de *issues* diariamente para os usuários, pegando-as de repositórios em que o usuário é inscrito na plataforma. O software foi desenvolvido com o *framework* Ruby on Rails e utiliza a API do GitHub para fazer o carregamento de *issues* ativas para o banco de dados do sistema.

3.3. *Data Collection and Analysis of GitHub Repositories and Users*

O trabalho de Chatziasimidis e Stamelos (2015) apresenta um estudo sobre as características dos usuários e os fatores de sucesso de projetos do GitHub. Foram utilizados dados de 100 mil projetos e 10 mil usuários que foram obtidos através da API da plataforma. Com esses dados, foram aplicados algoritmos de mineração de dados para atingir os objetivos do trabalho. O trabalho conseguiu identificar diversas características que influenciam no sucesso de um projeto *open-source* no GitHub.

3.4. Considerações sobre os Trabalhos Correlatos

Todos os trabalhos relacionados aqui mencionados auxiliaram no desenvolvimento deste projeto, dado que cada um se encaixa e agrega de uma forma diferente. Buscou-se um trabalho para cada ponto principal do presente projeto; com isso, cada trabalho propôs algo diferente, buscando uma maior gama de informações como base de estudo.

As técnicas de recomendação utilizadas por Furlan serviram como base para a seleção de quais serão úteis para o presente trabalho, descartando a técnica de recomendação colaborativa que, apesar de apresentar as tecnologias mais maduras, possui limitações, como a exigência de uma avaliação inicial para todos os itens a serem recomendados.

Foi utilizada a forma com que o software CodeTriage faz a pré-seleção de quais projetos serão disponibilizados na plataforma, fazendo com que a quantidade de total de *issues* seja limitada.

Foram estabelecidas quais informações podem ser obtidas de um projeto e de um usuário do GitHub utilizando a própria API da plataforma; assim sendo, foi possível definir quais informações serão relevantes para o presente trabalho.

4. Metodologia

Para o desenvolvimento do trabalho, foram utilizadas boas práticas da metodologia *Feature-Driven Development* (FDD) e, também a técnica de desenvolvimento *Test-Driven Development* (TDD). A FDD é a metodologia de desenvolvimento ágil mais completa e robusta; a utilização de boas práticas da mesma faz com que seja possível definir claramente qual será o escopo do trabalho, enquanto isso, o TDD faz com que o desenvolvimento ágil tenha garantia de segurança e confiabilidade com testes automatizados de toda a aplicação.

4.1. Desenvolver um Modelo Abrangente

A Figura 1 apresenta o diagrama de atividades, que compreende o fluxo de rotinas que serão desenvolvidas para a interação do usuário com o sistema. Em um primeiro momento, o usuário acessa o website, onde ele poderá fazer um cadastro ou fazer o login - caso já possua um cadastro. Ao fazer um novo cadastro, é necessário preencher as informações necessárias do perfil, como nível de programação e linguagens dominadas. Com essas informações, é possível fazer a solicitação de *issues*; o sistema busca *issues* com o uso da API do GitHub. Após a obtenção da lista de *issues*, é feita a pontuação e predição das *issues* e, com isso, é apresentado ao usuário uma lista com as mais próximas dos interesses do usuário.

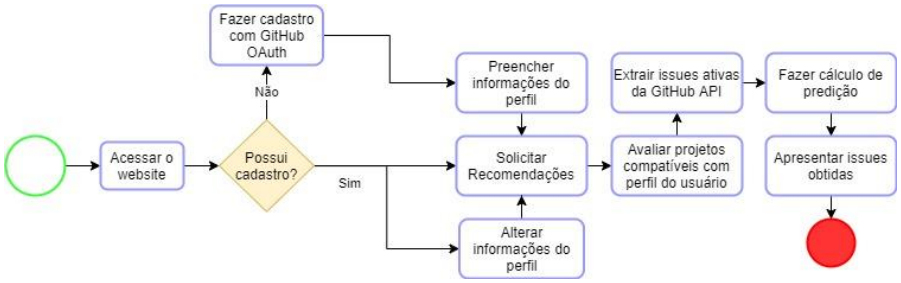


Figura 1. Modelo abrangente representado por Diagrama de Atividades

4.2. Construir a Lista de Funcionalidades

Construir uma Lista de Funcionalidades é o segundo passo do desenvolvimento proposto pela metodologia FDD. Nessa etapa, todas as funcionalidades necessárias ao cumprimento das necessidades do cliente são levantadas. Dessa maneira, tais funcionalidades podem ser agrupadas em requisitos que irão auxiliar no processo de desenvolvimento do projeto [Silva et al. 2009]. Os requisitos deste projeto são apresentados nas Tabelas 1 e 2; a Tabela 1 apresenta os Requisitos Funcionais (RF), enquanto a Tabela 2 apresenta os Requisitos Não Funcionais (RNF).

Tabela 1. Requisitos Funcionais

RF01 - Controle de Usuário: O sistema deverá gerenciar funções de cadastro e exclusão de usuário.	
Complexidade: Baixa	Relevância: Essencial
RF02 - Criação de perfil: O sistema deverá permitir o usuário a fazer o gerenciamento de seu perfil.	
Complexidade: Baixa	Relevância: Essencial
RF03 - Cadastro de projetos: O sistema deverá gerenciar funções de cadastro e exclusão de projetos opensource.	
Complexidade: Baixa	Relevância: Essencial
RF04 - Filtragem de projetos: O sistema deverá fazer a filtragem de projetos baseado no perfil do usuário.	
Complexidade: Média	Relevância: Essencial
RF05 - Aquisição de issues: O sistema deverá obter issues a partir da API do GitHub.	
Complexidade: Alta	Relevância: Essencial
RF06 - Cálculo de similaridade: O sistema deverá fazer o cálculo de similaridade das issues obtidas com o perfil do usuário.	
Complexidade: Alta	Relevância: Essencial
RF07 - Recomendação de issues: O sistema deverá fazer a recomendação de issues com o resultado da função de similaridade.	
Complexidade: Baixa	Relevância: Essencial

Tabela 2. Requisitos Não Funcionais

RNF01 - Linguagem de programação: O sistema será desenvolvido com o uso das linguagens Elixir, HTML, CSS e Javascript.
RNF02 - Banco de dados: O banco de dados a ser utilizado é PostgreSQL.
RNF03 - Framework para o servidor: O sistema deve utilizar o framework Phoenix, e aplicar a arquitetura MVC.
RNF04 - Framework para as interfaces: O layout do sistema deve utilizar o Bootstrap, a fim de manter a exibição responsiva.

4.3. Planejar por Funcionalidade

A terceira etapa da metodologia FDD é planejar por funcionalidade; dessa forma, a ordem que será seguida e o tempo dedicado a cada funcionalidade que será implementada são definidos (Apêndice A).

O Diagrama de Casos de Uso deste projeto é apresentado na Figura 3. Tal diagrama é responsável por mostrar o conjunto de atores e casos de uso e os seus relacionamentos, o que ajuda a organizar os comportamentos do sistema, auxiliando os responsáveis pelo projeto a terem uma visão mais completa das interações que ocorrem na aplicação que será desenvolvida. Este diagrama é importante principalmente para a organização e modelagem dos comportamentos de um sistema.

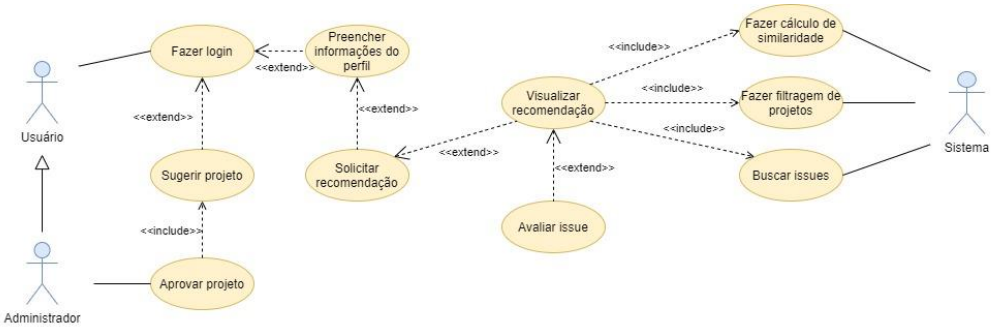


Figura 3. Diagrama de Casos de Uso

Juntamente com o Diagrama de Casos de Uso também foram construídos descritivos de caso de uso para os principais fluxos da aplicação (Apêndice B).

4.4. Projetar por Funcionalidade

Essa é a fase da metodologia em que o modelo da interface do usuário pode ser esboçado, assim como os diagramas de sequência, de atividades e de classes também podem ser apresentados [Silva et al. 2009].

As principais funcionalidades e o fluxo geral da aplicação, ambos na perspectiva do usuário, podem ser observados através do diagrama de atividades, o qual foi apresentado na Figura 2, que se encontra na seção 4.2 do presente trabalho.

No Modelo Entidade-Relacionamento (MER) é possível visualizar todas as tabelas nas quais os dados serão armazenados, bem como seus atributos e suas relações entre si (Figura 5).

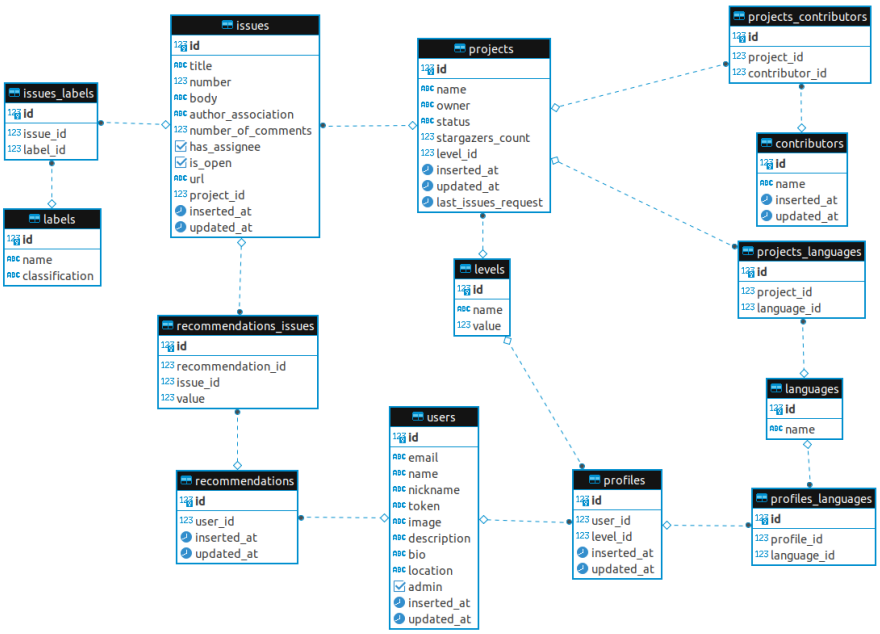


Figura 5. Modelo Entidade-Relacionamento

4.5. Construir por Funcionalidade

Nessa etapa da metodologia foi realizado o desenvolvimento do projeto, onde foram utilizadas todas as tecnologias propostas. Dentre os diversos módulos e funções que foram criados para a implementação da aplicação, foram selecionados alguns que foram considerados os mais relevantes para o funcionamento da aplicação.

Para solicitar uma recomendação o sistema verifica se as *issues* que possuí são atuais, caso não sejam o sistema invalida as *issues* cadastradas e faz a busca das *issues* ativas dos projetos que estarão na recomendação, isso é feito pela função *update_project_issues_from_github,* conforme mostrado na Figura 6.

```elixir
def update_project_issues_from_github(project) do
  current_issues = get_project_issues(project)
  set_issues_closed(current_issues)

  with {:ok, issues_params} <- IssueSeeker.Http.Issue.get(project),
       {:ok, updated_issues} <- create_or_update_issues(project, issues_params) do
    update_last_issue_request(project)
    {:ok, updated_issues}
  else
    error -> error
  end
end
```

Figura 6. Trecho de código que faz a invalidação das *issues* atuais e busca novas

A função faz o uso do módulo *IssueSeeker.Http.Issue,* o qual é o módulo responsável por fazer a requisição para a API do GitHub e obter as *issues* ativas daquele projeto. A função *get* deste módulo é mostrada na Figura 7. A requisição é feita utilizando a biblioteca HTTPoison, a qual faz requisições por meio do protocolo HTTP.

```elixir
def get(owner, name) do
  case url(owner, name) |> HTTPoison.get() do
    {:ok, %HTTPoison.Response{} = response} ->
      {:ok, format_response(response)}
    {:error, error} ->
      {:error, error}
  end
end
```

Figura 7. Trecho de código que faz a requisição das *issues* na API do GitHub

Neste caso, a url da requisição é definida dinamicamente pela função *url* que recebe os parâmetros do projeto no qual fará a requisição, isso é mostrado na Figura 8.

```elixir
defp url(owner, name) do
  "#{base_url()}/repos/#{owner}/#{name}/issues?state=open"
end
```

Figura 8. Trecho de código que monta a url dinamicamente para a busca das *issues*

Após ser feita a requisição das issues para a API do GitHub, a função *format_response,* que é mostrada na Figura 9, recebe a resposta dessa requisição e extraí os parâmetros necessários para salvar na base de dados, as informações extraídas de uma *issue* são: Número, título, descrição, associação do autor, data de atualização, data de criação, indicação se está aberta, número de comentários e url de detalhes do GitHub.

```elixir
defp filter_issues(%{"pull_request" => _}, acc), do: acc
defp filter_issues(issue, acc) do

  issue_params =
    issue
    |> Map.take([
      "number",
      "title",
      "body",
      "author_association",
      "updated_at"
    ])
    |> Map.merge(%{
      "is_open" => (Map.get(issue, "state") == "open"),
      "number_of_comments" => Map.get(issue, "comments"),
      "has_assignee" => (Map.get(issue, "assignee") != nil),
      "labels" => (
        issue
        |> Map.get("labels")
        |> Enum.map(&(Map.get(&1, "name")))
      ),
      "url" => Map.get(issue, "html_url"),
      "inserted_at" => Map.get(issue, "created_at"),
    })

  acc ++ [issue_params]
end
```

Figura 9. Trecho de código que extrai as informações de uma resposta da API do GitHub

5. Resultados

A proposta geral deste trabalho foi o desenvolvimento de uma aplicação web que obtenha *issues* a partir da API do GitHub e faça recomendações aos usuários, visando auxiliar tanto os desenvolvedores quanto os projetos *open-source*. Para facilitar o uso da aplicação, ela foi disponibilizada na internet para o acesso por meio de qualquer navegador.

Devido ao grande número de interfaces elaboradas para esta aplicação, serão apresentadas apenas as interfaces com maior nível de relevância para o projeto, ou seja, as interfaces que mostram as recomendações e *issues* do sistema.

A Figura 10 mostra o menu superior da aplicação, neste menu que podem ser acessadas todas as ações que serão feitas na aplicação.

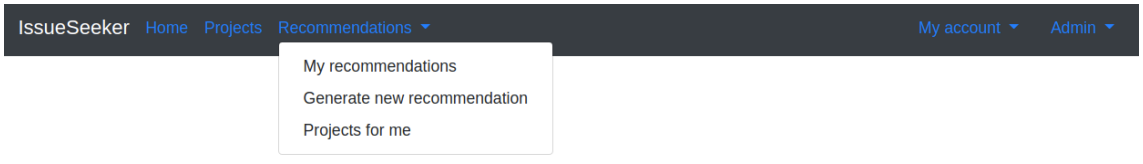
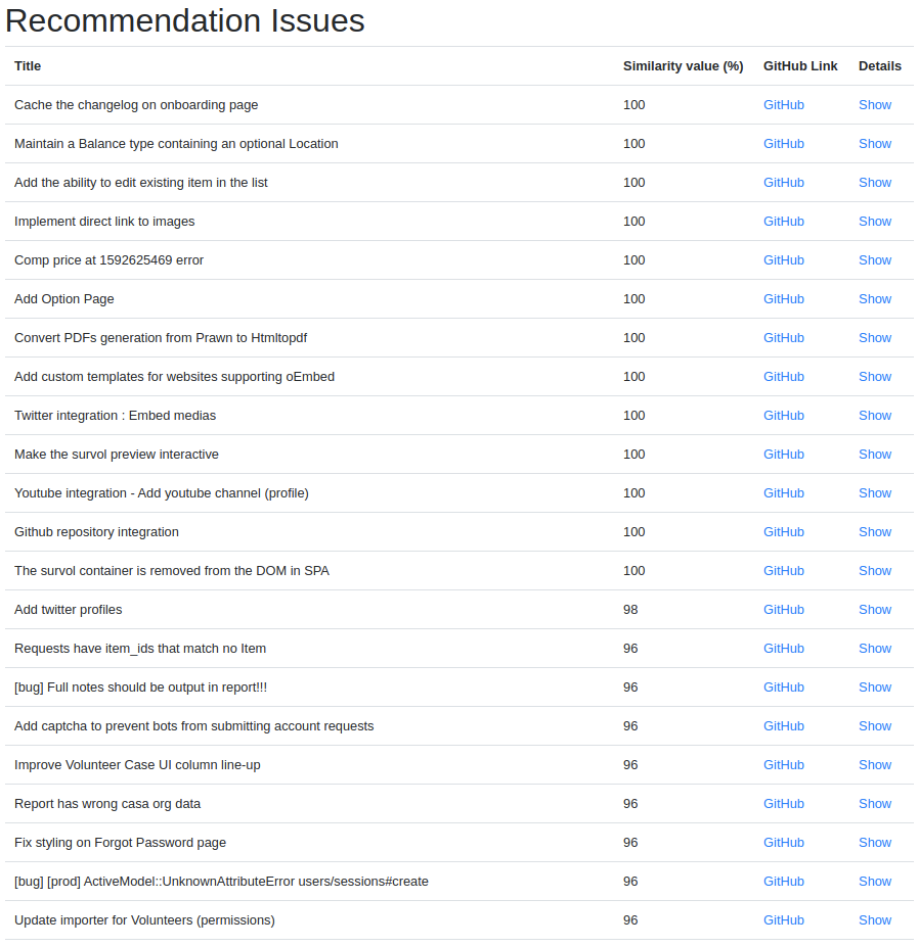


Figura 10. Barra de menu da aplicação

Com o perfil preenchido com as informações necessárias, é possível acessar a opção *Generate new recommendation* que cria uma nova recomendação ao usuário. Ao gerar uma nova recomendação, o sistema apresenta a interface de recomendação que pode ser visualizada na Figura 11. Esta interface mostra uma tabela com as principais *issues* que precisam de ajuda no momento, onde são apresentadas informações como título da *issue*, valor da similaridade, link do GitHub e botão de detalhes.



Recommendation Issues

Title	Similarity value (%)	GitHub Link	Details
Cache the changelog on onboarding page	100	GitHub	Show
Maintain a Balance type containing an optional Location	100	GitHub	Show
Add the ability to edit existing item in the list	100	GitHub	Show
Implement direct link to images	100	GitHub	Show
Comp price at 1592625469 error	100	GitHub	Show
Add Option Page	100	GitHub	Show
Convert PDFs generation from Prawn to Htmltopdf	100	GitHub	Show
Add custom templates for websites supporting oEmbed	100	GitHub	Show
Twitter integration : Embed medias	100	GitHub	Show
Make the survol preview interactive	100	GitHub	Show
Youtube integration - Add youtube channel (profile)	100	GitHub	Show
Github repository integration	100	GitHub	Show
The survol container is removed from the DOM in SPA	100	GitHub	Show
Add twitter profiles	98	GitHub	Show
Requests have item_ids that match no Item	96	GitHub	Show
[bug] Full notes should be output in report!!!	96	GitHub	Show
Add captcha to prevent bots from submitting account requests	96	GitHub	Show
Improve Volunteer Case UI column line-up	96	GitHub	Show
Report has wrong casa org data	96	GitHub	Show
Fix styling on Forgot Password page	96	GitHub	Show
[bug] [prod] ActiveModel::UnknownAttributeError users/sessions#create	96	GitHub	Show
Update importer for Volunteers (permissions)	96	GitHub	Show

Figura 11. Interface da lista de *issues* de uma recomendação

O valor de similaridade de uma *issue* é obtido utilizando-se diferentes pesos para as variáveis que são levadas em consideração. Foram testadas diversas combinações de pesos para se chegar a um resultado mais otimizado. As varáveis que são levadas em consideração para o cálculo são:

- Data de criação, com peso de 10% do valor total;
- *Labels,* com peso de 30% do valor total;
- Associação do autor, com peso de 20% do valor total;
- Número de comentários, com peso de 10% do valor total;
- Indicação se já possuí alguém atribuído, com peso de 30% do valor total.

A interface de detalhes de uma *issue* é apresentada na Figura 12, onde apresenta mais informações sobre ela.

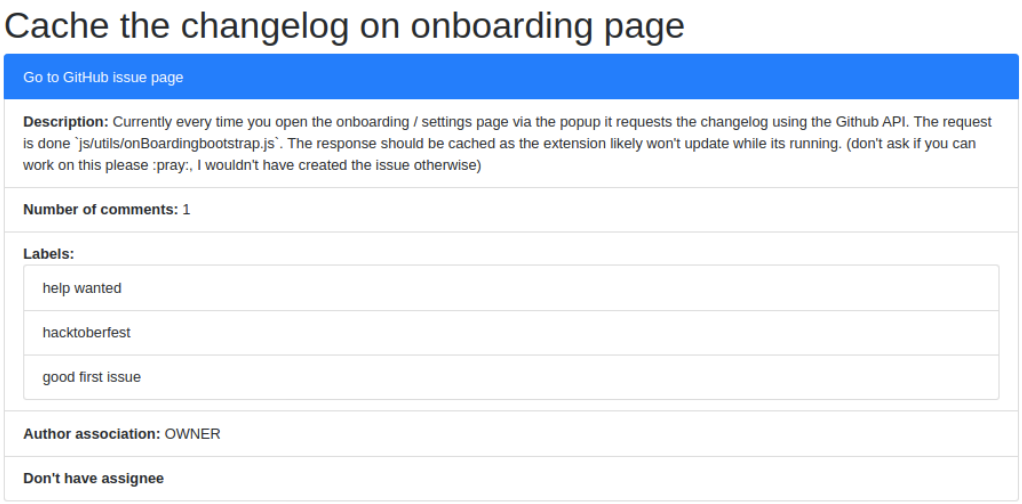


Figura 12. Interface de detalhes de uma *issue*

O trabalho foi publicado no XXIV Simpósio de Ensino, Pesquisa e Extensão - SEPE como trabalho completo no ano de 2020.

6. Testes e Validação

Com a utilização do padrão de desenvolvimento baseado em testes, ao final do desenvolvimento chegou-se ao número de 138 cenários de testes unitários os quais foram divididos em testes para os modelos, contextos e controladores, onde os módulos mais críticos para o funcionamento da aplicação tiveram uma cobertura de mais de 90% do código.

Os testes unitários são essenciais em aplicações modernas pois garantem que nenhuma funcionalidade base da aplicação quebrará no desenvolvimento de novas funcionalidades para a aplicação.

7. Conclusão

Este trabalho propôs o desenvolvimento e implementação de uma aplicação web para recomendação de *issues* de projetos *open-source* para desenvolvedores com a finalidade de beneficiar tanto o programador com agilidade em uma tarefa manual, quanto os projetos, fornecendo visibilidade aos mesmos.

Baseado nas pesquisas relacionadas, foi possível constatar a carência de uma aplicação com a mesma finalidade. As aplicações encontradas na internet não apresentam uma proposta igual, limitando-se apenas a apresentar os projetos *open-source* e suas *issues*, sem qualquer tipo de filtragem ou recomendação. Todavia, encontrar projetos semelhantes foi de suma importância, visto que, com eles, foi possível identificar os principais requisitos e carências na área.

Destaca-se a importância do GitHub, que foi uma peça fundamental no desenvolvimento deste trabalho, uma vez que, sem sua existência, tanto como plataforma quanto como API, seria impossível a criação de uma aplicação como esta.

Para trabalhos futuros, é sugerido que seja criado um outro projeto para o *front-end* da aplicação utilizando tecnologias como Javascript e React, enquanto o projeto atual ficará responsável somente pelo *back-end* em forma de uma API, o que faria com que a experiência do usuário que fosse utilizar a aplicação fosse melhorada e para que a estruturação do código fique mais organizada.

Referências

Beck, K. (2003) "Test-driven Development: By Example", In: Addison-Wesley Professional, Boston, Massachusetts.

Bootstrap (2020). "Bootstrap". Disponível em: https://getbootstrap.com/. Acesso em: maio de 2020.

Carrillo, M. G., Alcaide, I. G., and Herranz, S. M. (2005). "Applying hierarchical mvc architecture to high interactive web application", 3rd International Conference on Information research, Applications and Education – ITECH, pages 110–115.

Chatziasimidis, F. and Stamelos, L. (2015). "Data collection and analysis of github repositories and users", 6th International Conference on Information, Intelligence, Systems and Applications (IISA).

Elixir (2020). "The Elixir Programming Language". Disponível em: https://elixir-lang.org/. Acesso em: outubro de 2020.

Furlan, L. A. R., Zamberlan, A. O., Vieira, S. A. G., and Canal, A. P. (2018). "Desenvolvimento de um sistema de recomendação para bibliotecas digitais", DisciplinarumScientia – Naturais e Tecnológicas, 19(1):87–103.

GitHub, I. (2020a). "Github". Disponível em: https://github.com/. Acesso em: maio de 2020.

GitHub, I. (2020b). "Graphql api v4". Disponível em: https://developer.github.com/. Acesso em: maio de 2020.

Phoenix (2020a). "Phoenix Framework". Disponível em: https://www.phoenixframework.org/. Acesso em: outubro de 2020.

Phoenix (2020b). "Documentação do Phoenix". Disponível em: https://hexdocs.pm/phoenix/overview.html. Acesso em: outubro de 2020.

PostgreSQL (2020). "Postgresql". Disponível em: https://www.postgresql.org/about/. Acesso em: maio de 2020.

Reategui, E. B. and Cazella, S. C. (2005). "Sistemas de recomendação", XXV Congresso da Sociedade Brasileira de Computação.

Ricci, F., Rokach, L., and Shapira, B. (2011). "Introduction to recommender systems hand-book", Recommender Systems Handbook, (1):1–35. Springer, Boston, MA.

Schneeman, R. (2020). “Codetriage”. Disponível em: https://www.codetriage.com/. Acesso em: maio de 2020.

Silva, F. G., Hoentsch, S. C. P., and Silva, L. (2009). “Uma análise das metodologias ágeis fdd e scrum sob a perspectiva do modelo de qualidade mps.br”, Scientia Plena, 5(12).

W3C (2020). “Cascading style sheets”. Disponível em: https://www.w3.org/Style/CSS/. Acesso em: maio de 2020.

WHATWG (2020). “Html”. Disponível em: https://html.spec.whatwg.org/. Acesso em: maio de 2020.

Apêndice A. Estimativa do tempo das funcionalidades

O planejamento foi feito para cada funcionalidade e foram levados em consideração múltiplos fatores, os principais foram o nível de dificuldade para implementação e a relevância geral da funcionalidade. Além disso, para fazer os cálculos, foram consideradas métricas como:

- 15 horas semanais para implementação da aplicação;
- 20 semanas no total para o desenvolvimento, ou seja, 300 horas;
- Levando em consideração a utilização do TDD, pelo menos 40% das horas de desenvolvimento serão dedicadas à criação de testes automatizados.

Tabela 3. Estimativa do tempo de implementação das funcionalidades

Funcionalidade	Porcentagem do tempo de trabalho (%)	Horas de trabalho
RF01 - Controle de Usuário	20	60
RF02 - Criação de perfil	5	15
RF03 - Cadastro de projetos	5	15
RF04 - Filtragem de projetos	5	15
RF05 - Aquisição de issues	25	75
RF06 - Cálculo de similaridade	30	90
RF07 - Recomendação de issues	10	30
TOTAL	**100**	**300**

Apêndice B. Descritivos de Caso de Uso

Tabela 4. Descritivo do Caso de Uso Preencher Informações do Perfil

Identificação	UC1
Caso de uso	Preencher informações do perfil
Descrição	Esse caso de uso é responsável por fazer o preenchimento das informações do usuário no seu perfil.
Ator Principal	Usuário
Atores Secundários	Sistema
Pré-condições	
PC1	O usuário deve ter feito login com sua conta do GitHub.
Pós-condições	
PO1	A página principal é apresentada.
Fluxo Principal	
FP1	1.1. O usuário acessa a interface de configuração do perfil. 1.2. O sistema exibe as opções a serem configuradas. 1.3. O usuário preenche seu nível de programação no menu de seleção, que pode ser Junior, Pleno ou Sênior. 1.4. O usuário preenche as linguagens em que sabe programar nas caixas de seleção. 1.5. O usuário clica no botão "Salvar Informações". 1.6. O sistema valida os dados informados pelo usuário [FE1]. 1.7. O sistema informa ao usuário que as configurações foram aceitas. 1.8. Fim do caso de uso.
Fluxo de Exceções	
FE1	1.1. Os dados preenchidos estão inválidos. 1.2. O sistema apresenta uma mensagem de erro informando quais campos estão inválidos. 1.3. O sistema retorna ao [FP1.2].

Tabela 5. Descritivo do Caso de Uso Solicitar Recomendação

Identificação	UC2
Caso de uso	Solicitar Recomendação
Descrição	Esse caso de uso é responsável por solicitar uma recomendação na aplicação, sendo que a mesma é executada apenas quando as informações do usuário já foram preenchidas.
Ator Principal	Usuário
Atores Secundários	Sistema
Pré-condições	
PC1	O usuário deve ter preenchido as informações de seu perfil.
Pós-condições	
PO1	Recomendação é apresentada
Fluxo Principal	
FP1	1.1. O usuário seleciona a opção "Receber Recomendação" na Tela Principal. 1.2. O sistema seleciona quais projetos podem ser incluídos na recomendação com base no perfil do usuário. 1.3. O sistema busca todas issues ativas dos projetos selecionados fazendo uma requisição na API do GitHub. 1.4. O sistema faz o cálculo de similaridade de cada issue obtida com o perfil do usuário. 1.5. O sistema apresenta o resultado do cálculo para o usuário. 1.6. Fim do caso de uso.