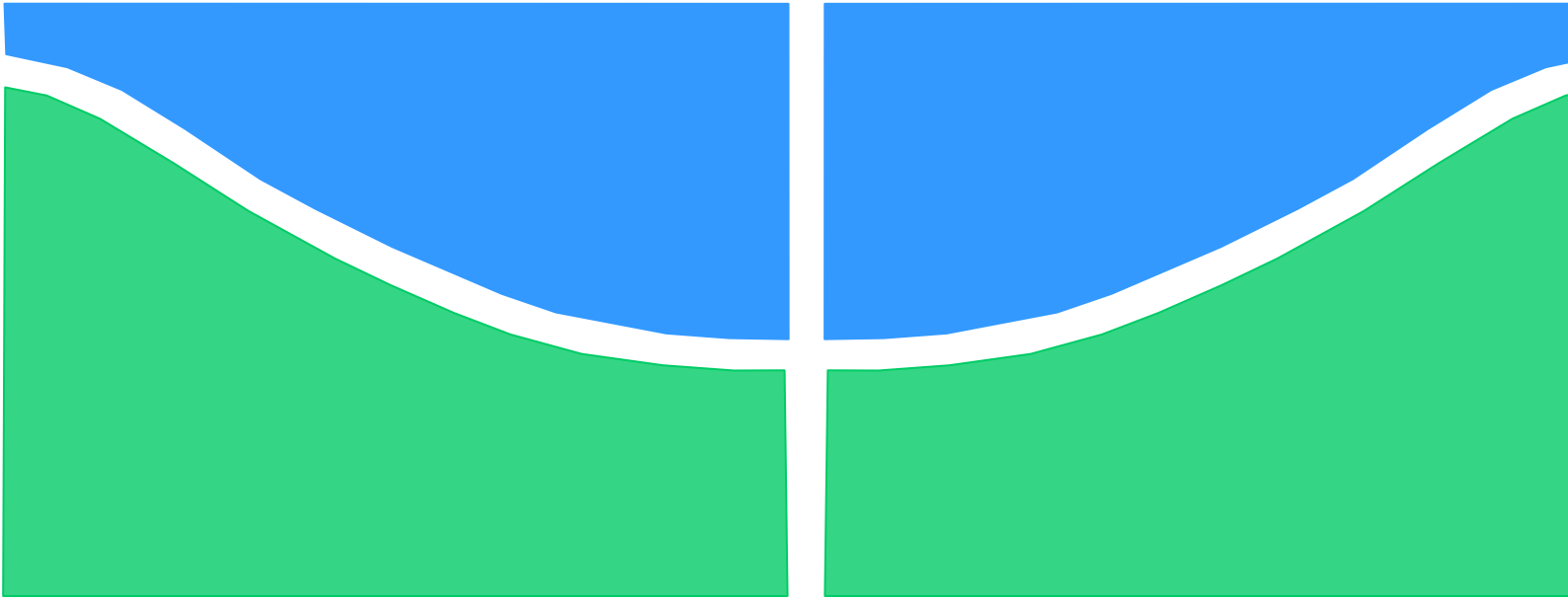
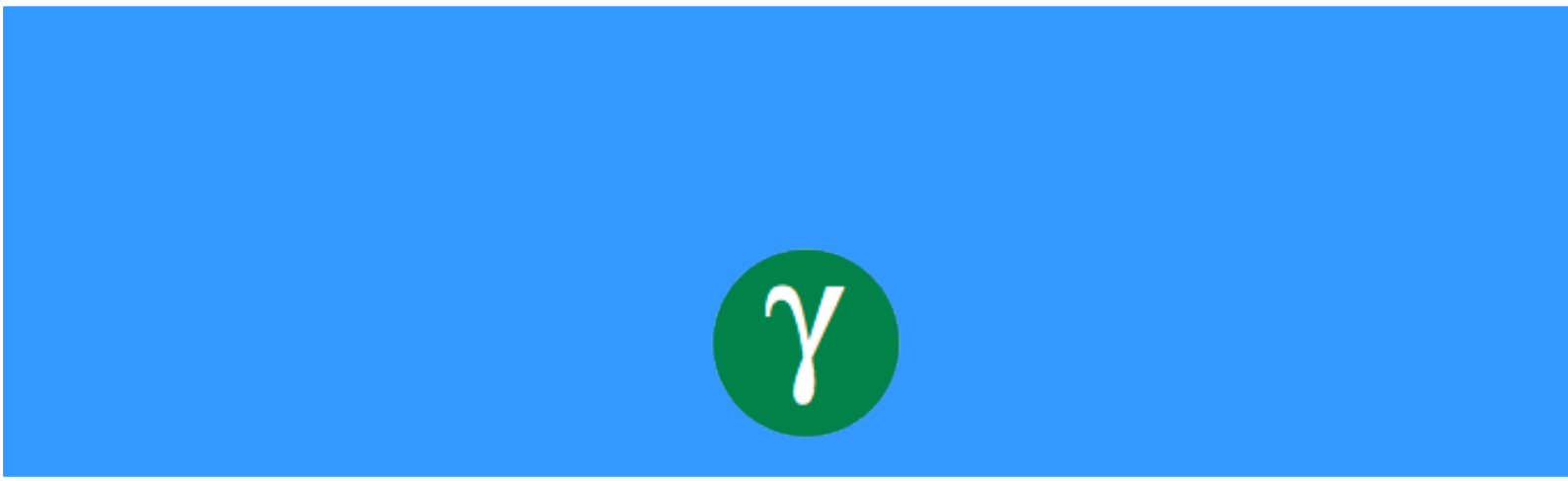


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia de Software

APLICATIVO PARA DISPOSITIVOS MÓVEIS INTEGRADO COM A FERRAMENTA URI ONLINE JUDGE

Autor: Lucas dos Santos Ribeiro Leite
Orientador: Dr. Vandor Roberto Vilardi Rissoli

Brasília, DF
2018



LUCAS DOS SANTOS RIBEIRO LEITE

APLICATIVO PARA DISPOSITIVOS MÓVEIS INTEGRADO COM A FERRAMENTA URI ONLINE JUDGE

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Orientador: Dr. Vandor R. V. Rissoli

Brasília, DF
2018

CIP – Catalogação Internacional da Publicação

Leite, Lucas dos Santos Ribeiro.
Aplicativo para dispositivos móveis integrado com a ferramenta URI Online Judge / Lucas dos Santos Ribeiro Leite (em ordem normal). Brasília: UnB, 2018. 103 p. : il. ; 29,5 cm.

Monografia (Graduação) – Universidade de Brasília Faculdade do Gama, Brasília, 2018. Orientação: Vandor R. V. Rissoli.

1. Dispositivos Móveis. 2.Juíz online. 3. Aplicativo móvel. 4. Prototipação. 5. Testes de Usabilidade. I. Rissoli, Vandor Roberto Vilardi. II. Aplicativo Para Dispositivos Móveis Integrado Com A Ferramenta URI Online Judge.

CDU Classificação

REGULAMENTO E NORMA PARA REDAÇÃO DE RELATÓRIOS DE PROJETOS DE GRADUAÇÃO FACULDADE DO GAMA - FGA

Lucas dos Santos Ribeiro Leite

Monografia submetida como requisito parcial para obtenção do título de Bacharel em Engenharia de Software da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 04/07/2018 apresentada e aprovada pela banca examinadora abaixo assinada:

Prof. Dr. Vandor Roberto Vilardi Rissoli, UnB/ FGA
Orientador

Prof. Dr. Edson Alves da Costa Júnior, UnB/ FGA
Membro Convidado

Prof. Dr. André Barros de Sales, UnB/ FGA
Membro Convidado

Brasília, DF
2018

RESUMO

O URI Online Judge é um projeto desenvolvido pelo Departamento de Ciência da Computação da URI (Universidade Regional Integrada do Alto Uruguai e das Missões) que tem o objetivo de promover a prática de programação e o compartilhamento de conhecimento através de problemas de programação de nível iniciante até problemas de nível avançado. A ideia de implementar um aplicativo para dispositivos móveis veio dos próprios desenvolvedores por meio de uma seção no *site* de sugestões para novas funcionalidades. A proposta deste trabalho é implementar um aplicativo para dispositivos móveis com sistema operacional iOS utilizando técnicas de prototipação e testes de usabilidade visando uma interface intuitiva e eficiente. Ao final do trabalho, foi desenvolvida uma solução que permite a visualização de informações atualizadas presentes no *site* do URI *Online Judge*, diretamente na tela de um dispositivo móvel.

Palavras-chave: juiz *online*. dispositivos móveis. aplicativo móvel. iOS. prototipação. testes de usabilidade.

ABSTRACT

The URI Online Judge is a project that is being developed by the Computer Science Department of URI University. The main goal of the project is to provide programming practice and knowledge sharing through programming problems from beginner to advanced levels. The idea of developing an application for mobile devices came from the developers themselves at a section on the site about new functionalities sugestions. The goal of this project is to create an app for mobile devices with iOS using prototyping techniques and usability testing, aiming for an inituitive and efficient interface. By the end of this project, the solution implemented allows users to visualize updated information contained in the URI Online Judge website, using a mobile device.

Keywords: online judge. mobile devices. mobile application. iOS. prototyping. usability testing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

URI - Universidade Regional Integrada do Alto Uruguai e das Missões

iOS - *iPhone Operating System*

MVC - *Model View Controller*

IDE - *Integrated Development Environment*

API - *Application Programming Interface*

JSON - *JavaScript Object Notation*

UML - *Unified Modeling Language*

TCC – Trabalho de Conclusão de Curso

SUMÁRIO

INTRODUÇÃO

Esse capítulo apresenta o contexto em que o trabalho proposto se insere, a questão de pesquisa, o objetivo geral e os objetivos específicos, além de fornecer as orientações de como esse documento está organizado.

CONTEXTUALIZAÇÃO

O URI *Online Judge* (Juiz Disponível ou Juiz Conectado) é um projeto que está sendo desenvolvido pelo Departamento de Ciência da Computação da URI (Universidade Regional Integrada do Alto Uruguai e das Missões). O principal objetivo da ferramenta elaborado por este projeto é promover a prática de programação e o compartilhamento de conhecimento (URIONLINEJUDGE, 2018).

O projeto, chamado de URI Online Judge, é um repositório com quase 2 mil exercícios de programação, divididos em categorias que vão desde o nível iniciante até o avançado. Estes exercícios podem ser usados por docentes ou entusiastas na prática da programação. A plataforma analisa automaticamente as soluções criadas pelos usuários e avisa quando existirem erros ou incoerências para serem revistos. Além disso, cria um ranking de programadores e universidades, de acordo com o sucesso na resolução dos problemas. Atualmente, o sítio virtual possui cerca de 170 mil usuários e mais de 10 milhões de submissões para serem analisadas e/ou executadas. A construção desse projeto foi iniciada pelo professor Neilor Tonin e pelo aluno Jean Bez (REVISTA ABRIL, 2015).

O *site* é utilizado em alguns cursos de programação de computadores na Universidade de Brasília (UnB) do Gama (FGA) como forma de ajudar os discentes a melhorarem suas habilidades em lógica de programação, além de fornecer um tipo de competição entre os próprios estudantes, motivando-os ao estudo.

QUESTÃO DE PESQUISA

Ao navegar pelo *site* do URI *Online Judge* em busca de problemas de programação para impulsionar os estudos, uma limitação existente é a necessidade do interessado estar sempre usando um computador para realizar essa navegação nas pá-

ginas virtuais do ambiente URI. Observando essa necessidade surge a seguinte questão de pesquisa: Como tornar a navegação do *site* URI *Online Judge* mais acessível para os usuários aonde quer que estejam?

No *site* do URI existe uma funcionalidade que possibilita aos seus usuários sugerirem novas funcionalidades que poderiam ser implementadas na ferramenta de auxílio ao estudo de programação. Observando essas sugestões foi identificada a solicitação da disponibilização da ferramenta também em ambiente móvel por meio de um aplicativo. Essa sugestão reforça a questão de pesquisa e sinaliza uma forma de tornar ainda mais acessível os recursos do URI.

Com o objetivo de atender essa sugestão, coerente com a questão de pesquisa, foi elaborada uma análise sobre o desafio de fornecer acesso adequado ao URI através de dispositivos móveis e as possibilidades dos envolvidos neste trabalho. Reconhecendo o conhecimento prévio estabelecido nas experiências dos envolvidos no desenvolvimento para iOS, além da motivação em entregar um aplicativo funcional ao término desse trabalho, foi iniciada as atividades de desenvolvimento para o URI *Online Judge* que estão melhor especificadas nesse Trabalho de Conclusão de Curso (TCC).

OBJETIVO GERAL

O objetivo principal desse trabalho é criar um *software* para dispositivos móveis com sistema operacional iOS baseado na ferramenta URI *Online Judge*, com o intuito de apresentar na tela de um dispositivo móvel informações do próprio *site* do URI.

OBJETIVOS ESPECÍFICOS

Os objetivos específicos desse trabalho são:

- desenhar um protótipo de telas para dar uma visibilidade de como o aplicativo deve se comportar e determinar a viabilidade das funcionalidades propostas;
- realizar testes de usabilidade com possíveis usuários para determinar pontos de evolução na interface do aplicativo antes de começar a implementação;

- implementar o aplicativo baseado no protótipo desenvolvido.

METODOLOGIA

Considerando a importância de uma interface intuitiva e simples de um aplicativo móvel, o modelo ideal para esse projeto é a prototipação. A prototipação é um modelo que tem como objetivo facilitar o entendimento dos requisitos, possibilitando que seja proposta uma solução adequada para o cliente (CAMARINI, 2013).

No ciclo de vida da prototipação a definição do sistema ocorre de forma gradual e evolutiva por parte do usuário e do desenvolvedor. A Figura 1 ilustra a sequência de fases do ciclo de vida da prototipação (ALVES, VANALLE, 2001).

O sistema resultante da prototipação é apenas um protótipo que não pode ser considerado como um produto final, mas que pode ser considerado um investimento inicial ao projeto, demonstrando a viabilidade do mesmo (NEPOMUCENO, 2012).

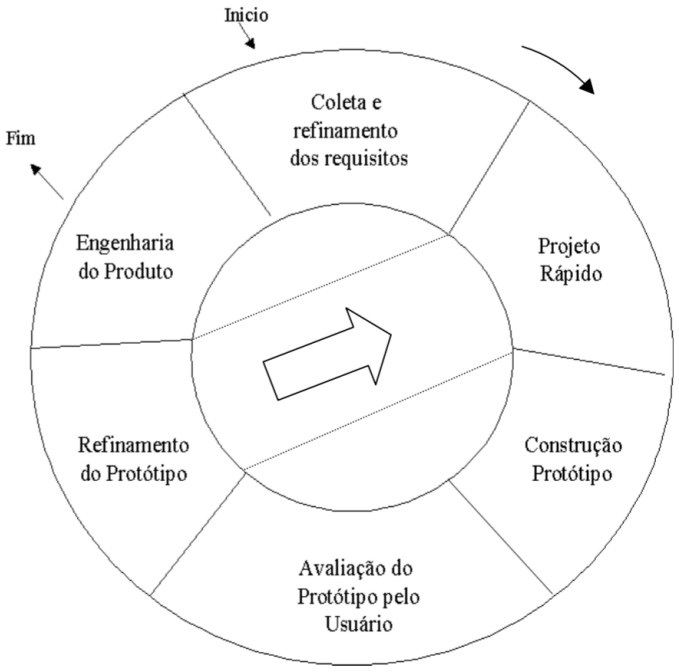


Figura 1 - Ciclo de Vida da Prototipação (ALVES, VANALLE, 2001).

CONTATO COM EQUIPE *URI ONLINE JUDGE*

Com o objetivo de desenvolver um produto final que tenha um nível de qualidade digno do ambiente URI, mantendo os padrões de interface, imagens e linguajar utilizados, foi necessário estabelecer o contato com a equipe responsável por manter o *site*. Desde a concepção da ideia desse trabalho, uma comunicação foi mantida com os criadores do projeto *URI Online Judge,* Neilor Tonin e Jean Bez, por meio de troca de e-mails e chamadas telefônicas. Essa comunicação foi de extrema importância durante todo o processo de desenvolvimento do projeto, pois foram discutidas novas ideias e sugestões de mudanças que ajudaram a deixar o novo aplicativo com a visão condizente com o *site* do URI.

Após o encerramento desse trabalho e a disponibilização do aplicativo, será importante manter o contato com a equipe do URI, a fim de que o processo de manutenção do aplicativo possa estar condizente com as alterações corretivas e/ou evolutivas do *site*, além do atendimento de outras demandas sugeridas pelos usuários do URI.

ESTRUTURA DO TRABALHO

Esse trabalho está dividido em capítulos. O capítulo 1 apresenta o Referencial Teórico do trabalho, onde são detalhados os conceitos que serviram de base para a sua elaboração. O capítulo 2 mostra o Suporte Tecnológico que detalha as ferramentas que foram usadas ao longo de todo o processo de desenvolvimento. O capítulo 3 intitulado Proposta, são descritas as funcionalidades presentes na nova aplicação, relacionando cada uma com sua respectiva tela do protótipo inicial. O Capítulo 4 mostrado os resultados obtidos, tanto com os testes de usabilidade realizados, quanto com as telas finais do aplicativo desenvolvido. O Capítulo 5 possui as considerações finais a respeito do trabalho e as ideias futuras que poderão incentivar a sua continuação.

1 REFERENCIAL TEÓRICO

Esse capítulo apresenta os principais conceitos envolvidos na elaboração desse trabalho (TCC). A aplicação dos conceitos apresentados nesse capítulo no desenvolvimento do projeto referente a esse trabalho, permitirá o desenvolvimento de um produto final com a qualidade esperada.

1.1 ARQUITETURA DO PROJETO (MVC - *MODEL VIEW CONTROLLER*)

MVC é um padrão arquitetural no qual a aplicação é dividida em três camadas distintas. Com isso, é possível agrupar as estruturas que compõem a aplicação de acordo com suas responsabilidades, preservando a independência de cada camada. Essas camadas consistem em Modelo (*Model*), Visão (*View*) *e* Controle (*Controller*). A camada Modelo é a abstração em *software* do domínio em questão, que pode ser desde um número simples de um contador, o texto de um editor de textos, até um objeto complexo. A camada Visão lida com a parte de apresentação, mostrando os dados obtidos da camada modelo de forma apropriada para o usuário. A camada Controle realiza a interface entre as camadas Visão e Modelo, além de capturar ações de dispositivos de entrada e passar essas ações para as outras camadas de acordo com a necessidade (KRASNER, POPE, 1988).

Um ciclo padrão da arquitetura MVC começa com uma ação do usuário detectada pela Controladora que notifica a Camada Modelo para se modificar de forma apropriada. A Visão pode, então, determinar se deve atualizar os dados apresentados a partir do novo estado da Modelo. As interações entre as camadas do MVC são representadas na Figura 2.

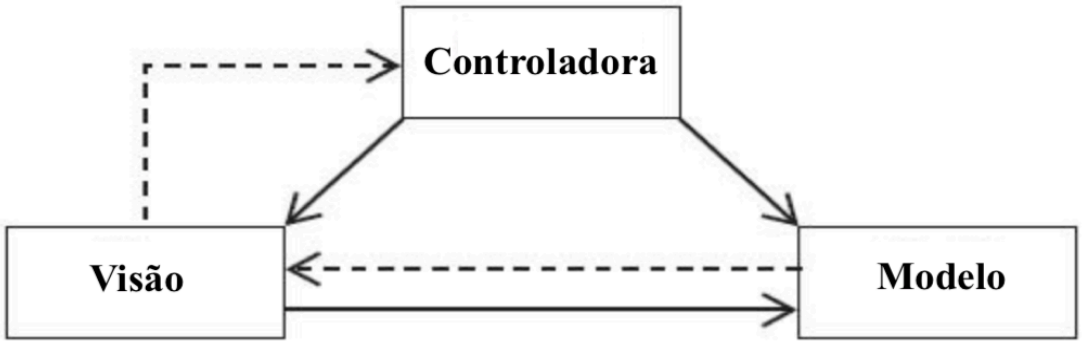


Figura 2 - Interação entre as camadas do MVC (KRASNER, POPE, 1988).

Para atingir os objetivos desse projeto, será necessário conhecer os conceitos abordados nessa seção, considerando que o padrão de arquitetura MVC foi o escolhido para ser seguido na implementação do aplicativo móvel. A escolha desse padrão de arquitetura para o projeto em questão se deu devido a simplicidade da separação de suas camadas baseado na responsabilidade de cada elemento, além de ser o padrão recomendado pela *Apple* a ser seguido no desenvolvimento de aplicações para dispositivos móveis com *iOS*. A arquitetura do projeto desenvolvido será melhor especificada no capítulo 4 denominado Proposta.

1.2 USABILIDADE EM DISPOSITIVOS MÓVEIS

O uso de dispositivos móveis tem crescido e se tornado extremamente popular nos últimos anos. Uma interface bem definida e com boa usabilidade é muito importante em aplicações para dispositivos móveis, considerando o tamanho reduzido da tela nesses aparelhos. Jun Gong e Peter Tarasewich (2010) dizem que apesar de existirem regras de design para aplicações *desktop,* como as 8 regras de ouro do design de interface, não existem regras similares para dispositivos móveis. Entretanto, 4 destas regras de ouro podem ser aplicadas, sem alterações, para aplicações voltadas para dispositivos móveis.

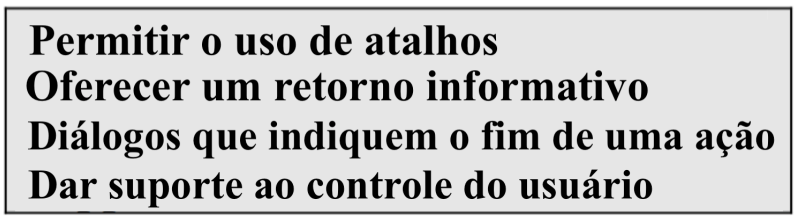


Figura 3 - Regras interessantes para as aplicações móveis (GONG, TARASEWICH, 2010).

Essas 4 regras são apresentadas na Figura 3 e melhor esclarecidas a seguir:

- **permitir o uso de atalhos:** com o aumento do número de usuários, surge a necessidade, por parte do usuário, de reduzir o número de interações para realizar uma determinada operação;

- **oferecer um retorno informativo (*feedback*):** para cada ação do usuário, deve haver um *feedback* do sistema. Para ações frequentes e de menor importância, esse *feedback* pode ser modesto, enquanto que para ações mais importantes, a resposta deve ser mais substancial (AGNI, 2015);
- **diálogos que indiquem o fim de uma ação:** sequências de ações devem ser organizadas em grupos com um começo, meio e fim. Informação de *feedback* após a conclusão de um conjunto de ações dá aos usuários o sentimento de satisfação na realização de uma tarefa (AGNI, 2015).
- **dar suporte ao controle do usuário:** usuários experientes querem ter a sensação de que estão no comando da interface e que ela responda às suas ações. Eles não querem surpresas ou mudanças no comportamento familiar, e ficam incomodados com sequências de entrada de dados, dificuldade na obtenção de informações importantes e incapacidade de produzir o resultado esperado (SCHNEIDERMAN, 2010).

Jun Gong e Peter Tarasewich (2010) dizem que as outras quatro regras de ouro de Ben Schneiderman necessitam de alterações para se adequarem as aplicações para dispositivos móveis, sendo estas:

- **consistência:** usuários de dispositivos móveis podem precisar alternar com frequência entre as aplicações móveis e *desktop*. Nesses casos, a consistência dos dados deve ser mantida entre os dois ambientes;
- **permitir a fácil reversão de ações:** isso pode ser mais difícil para aplicações em dispositivos móveis devido a um poder de processamento e capacidade de memória menores, se comparados a um *desktop*. Se a aplicação precisar enviar dados para um servidor remoto, essa transferência deve ser feita com pouca frequência;
- **prevenção de erros:** em dispositivos móveis pequenos, a proximidade dos botões acaba tornando a ocorrência de erros mais frequente se comparado a uma interface para *desktop*. Por isso, em dispositivos móveis, prevenir a ocorrência de ações acidentais por parte do usuário torna-se ainda mais importante;

- **reduzir a carga de memória de curta duração:** um bom design de interface evita que o usuário tenha que memorizar informações de uma tela para que tenha que usá-las em outra (SCHNEIDERMAN, 2010). Utilizar modos de interação alternativos pode ser benéfico para qualquer tipo de aplicação (GONG, TARASEWICH, 2010).

Como um dos focos desse trabalho era definir uma usabilidade intuitiva para o aplicativo que estava sendo desenvolvido, tornou-se importante aprofundar o conhecimento nesse tema (usabilidade) para os dispositivos móveis. Por isso os principais conceitos norteadores para este trabalho foram abordados e serviram de diretrizes relevantes na elaboração das telas do aplicativo, a fim de montar sua interface intuitiva e de fácil utilização para o usuário.

1.2.1 Heurísticas de usabilidade

Nielsen (1995) aponta 10 heurísticas gerais para design de interfaces de usuários. Esses princípios são aplicáveis a praticamente qualquer tipo de interface de usuário:

- **visibilidade do estado do sistema:** o sistema sempre deve manter o usuário informado sobre o que está acontecendo através de *feedbacks* apropriados em tempo real;
- **correspondência entre o sistema e o mundo real:** o sistema deve "falar" a língua do usuário, através de termos e frases familiares que o usuário utiliza para se comunicar;
- **liberdade e controle do usuário:** o usuário deve sempre ter a opção de desfazer algo que tenha feito no sistema sem ter que passar por um processo extenso;
- **consistência e padrões:** é importante que o sistema mantenha um padrão em palavras, cores, desenhos e sons, pois o usuário não deve ter que se preocupar com elementos diferentes que significam a mesma coisa;
- **prevenção de erros:** um design que previne que algum problema ocorra é ainda melhor que mensagens de erro explicativas. Um bom design

elimina condições propícias ao erro ou solicita a confirmação ao usuário antes de realizar determinada ação;

- **reconhecimento ao invés de memorização:** o usuário não deve precisar decorar o caminho feito para realizar determinada ação. Por isso, o sistema deve tornar as opções disponíveis bem visíveis ou facilmente acessíveis;
- **flexibilidade e eficiência de uso:** o sistema deve ser capaz de ter uma boa experiência para usuários leigos e para os mais experientes, oferecendo atalhos para ações que são executadas com mais frequência.
- **estética e design minimalista:** os diálogos não devem possuir informações irrelevantes ou desnecessárias. A interface deve ser simples e objetiva;
- **ajude o usuário a reconhecer, diagnosticar e recuperar-se de erros:** as mensagens de erro devem ser claras, indicando o problema com precisão e sugerindo uma solução. Além disso, devem estar posicionadas próximas do conteúdo ou ação que causou a ocorrência do erro;
- **ajuda e documentação:** apesar de ser melhor se o sistema possa ser usado sem a necessidade de uma documentação, ela pode ser necessária. Nesse caso, deve ser fácil para o usuário procurar informações e o sistema deve listar passos a serem seguidos sem tornar a documentação muito grande (NIELSEN, 1995).

O conhecimento sobre as heurísticas de usabilidade foi essencial na criação das telas do aplicativo que estava sendo desenvolvido. O desenho do protótipo inicial da aplicação considerou todos os tópicos envolvidos nesse estudo, visando elaborar um esboço inicial das telas que os respeitasse. Entretanto, para ter uma interface intuitiva, é importante ter um conhecimento de como os usuários considerados como público-alvo do projeto lidariam com as telas desenhadas inicialmente. A próxima seção irá mostrar como a preocupação com possíveis problemas de usabilidade deverá ser abordada no contexto desse trabalho.

1.2.2 Testes de usabilidade

O termo "Teste de Usabilidade" é usado, geralmente, para definir qualquer técnica usada para avaliar um produto ou sistema. É um processo no qual participantes avaliam o grau em que um produto se encontra em relação a critérios específicos de usabilidade (RUBIN, CHISNELL, 1994).

A necessidade de realizar testes de usabilidade está interiorizada em todos aqueles que desenvolvem software, pois a usabilidade pode permitir que os usuários o usem com satisfação, eficácia e eficiência na realização de tarefas. Um software pode estar bem concebido em termos de funcionalidades, mas se sua usabilidade não for intuitiva o bastante, o usuário pode rejeitá-lo (CARVALHO, 2002).

Realizar testes de usabilidade com usuários reais é fundamental e imprescindível já que providencia informações diretas sobre como as pessoas utilizam o sistema e quais problemas elas encontram na interface sendo testada (NIELSEN, 1994).

Com a realização de testes de usabilidade, pode-se registrar os melhores resultados obtidos para futuras realizações levando à minimização do custo do serviço de suporte aos usuários, crescimento de vendas e facilitar o lançamento de produtos mais competitivos e com menos problemas de usabilidade (RUBIN,CHISNELL, 1994).

Nielsen (1994) aponta a importância de se preocupar, em qualquer tipo de teste, com problemas relacionados à confiabilidade e validade dos testes. Confiabilidade é a questão de que, se os testes forem realizados repetidamente, os resultados devem ser idênticos, enquanto que a validade questiona se os testes realmente refletem os problemas na usabilidade da interface do sistema que está sendo testado.

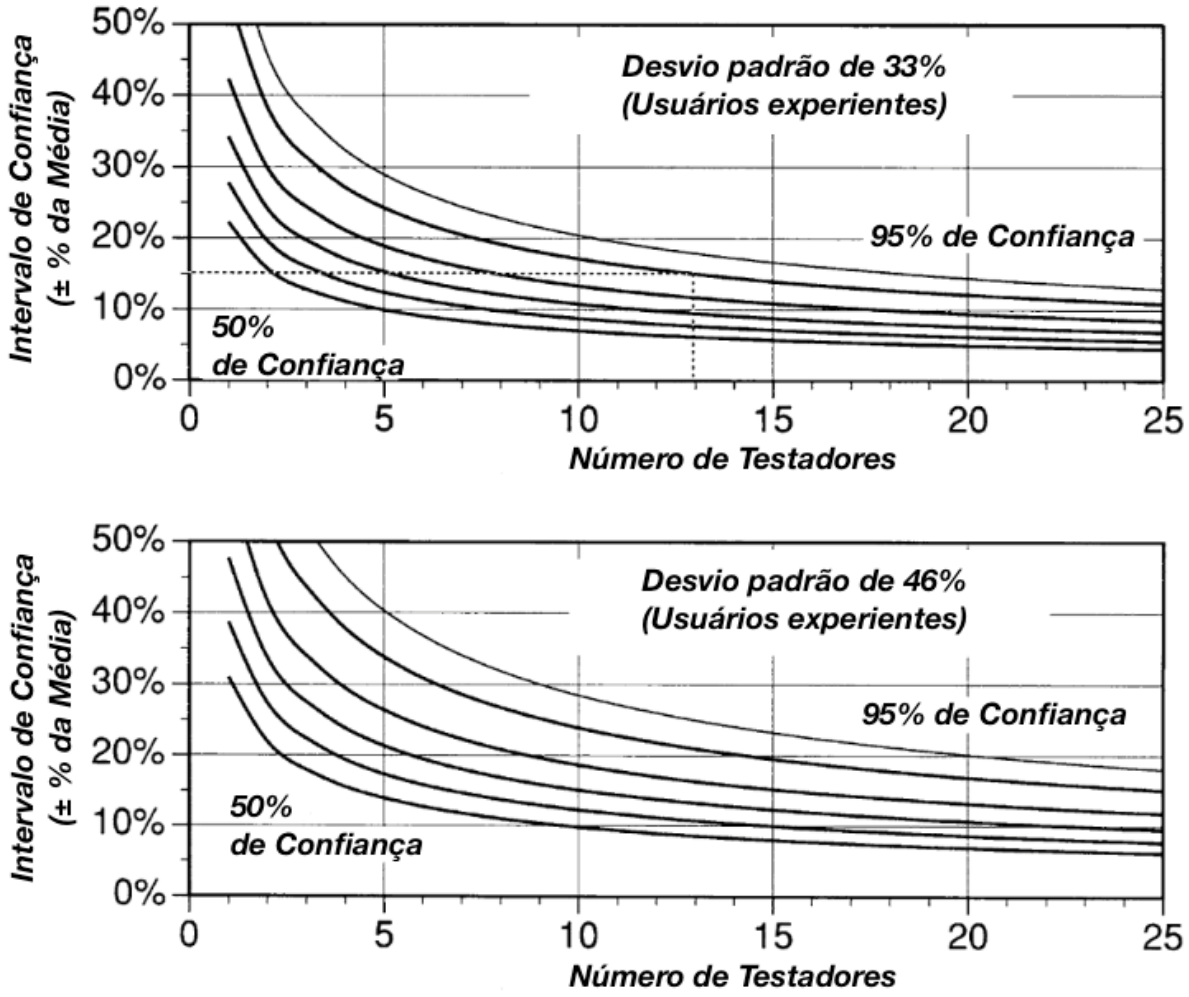


Figura 4 - Confiabilidade de testes de usabilidade (NIELSEN, 1994).

A Figura 4 mostra dois gráficos com intervalos de confiabilidade dos testes de usabilidade por número de usuários participantes dos testes. O primeiro gráfico considera usuários de nível experiente, enquanto que o segundo considera usuários sem experiência (NIELSEN, 1994).

Nielsen (2000) ainda comprova em seu texto *"Why you only need to test with 5 users"* que, em se tratando de testes de usabilidade, os melhores resultados podem ser obtidos ao se realizar testes com não mais que cinco usuários. Isso pode ser ex-

plicado pelo fato de que quanto mais usuários realizarem os mesmos testes na aplicação repetidas vezes, menos problemas inéditos serão encontrados, ou seja, uma falha de usabilidade detectada durante um teste, tem grande chance de ter sido detectada anteriormente em um teste com um usuário diferente. A relação entre a porcentagem total de problemas de usabilidade encontrados e o número de usuários envolvidos no teste de usabilidade pode ser visualizada no gráfico da Figura 5.

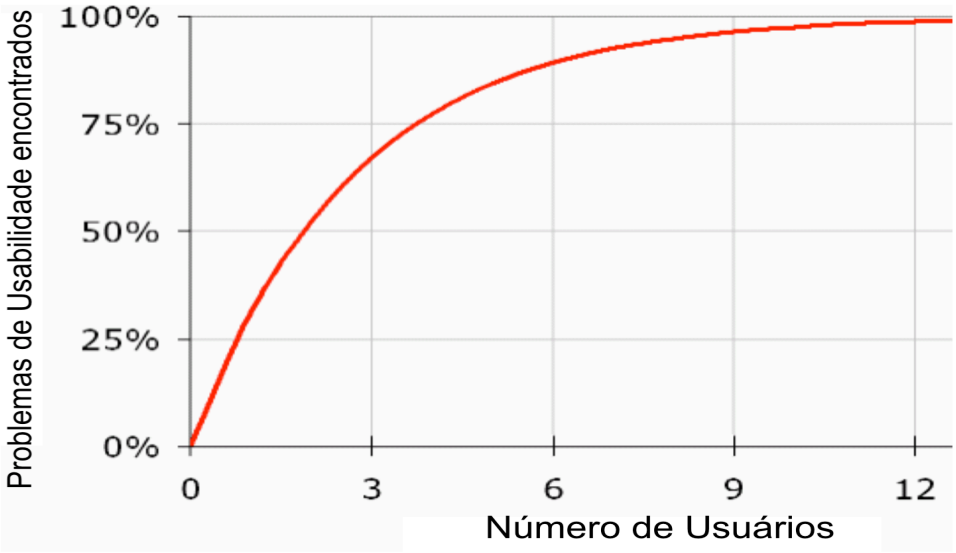


Figura 5 - Problemas de Usabilidade/Número de Usuários (NIELSEN, 2000).

Com o intuito de determinar possíveis problemas com a usabilidade nas telas do protótipo criado inicialmente, serão aplicados os conceitos vistos até aqui a respeito de testes de usabilidade. Desta forma, será possível alterar e melhorar a usabilidade do aplicativo móvel sendo criado, antes mesmo de ser iniciada a implementação da aplicação.

1.3 *WEB SCRAPING*

Uma área de grande importância na computação é a extração de dados em sistemas web, processo também conhecido como *Web Scraping* ou *Data Mining*. Essa técnica faz uso de processos automatizados para copiar informações específicas de um sistema *web,* para que alguns desses dados coletados sejam recuperados e analisados posteriormente (MITCHELL, 2015).

Essa técnica pode ser considerada muito útil por profissionais de *marketing* por exemplo, por facilitar a manipulação, busca e análise de dados para a otimização de vendas e personalização do atendimento a clientes, tanto que diversos negócios a utilizam com essa finalidade (WESTCON, 2017).

Ryan Mitchell (2015) lista uma série de razões que justificam a utilização de *Web Scraping* para recuperar dados de um sistema *web*:

- os dados a serem coletados estão armazenados em uma coleção de *sites* que não possuem uma API (*Application Programming Interface*) correspondente;
- os dados a serem coletados são pequenos ou não estão inclusos em uma API fornecida;
- A fonte fornecedora dos dados não possui infraestrutura ou habilidade técnica para construir uma API.

O conceito de *Web Scraping* será de extrema importância para o trabalho, pois no momento de desenvolvimento desse projeto, não existe uma API disponível para acesso aos dados do sistema *URI Online Judge*. Por esse motivo, foi considerada a ideia de utilizar essa técnica para servir como meio de obtenção dos dados que serão utilizados pelo aplicativo sendo desenvolvido.

2 SUPORTE TECNOLÓGICO

Neste capítulo, serão descritas as ferramentas que foram utilizadas no decorrer do processo de elaboração do trabalho, juntamente com informações a respeito de cada uma e justificativas para a escolha das mesmas no contexto do trabalho.

2.1 *PAGES*

Pages é um aplicativo de edição de textos incluído no pacote *iWork* da Apple, que corresponde a um conjunto de ferramentas exclusivas para Mac OS e iOS. O *Pages* consegue importar alguns documentos do Microsoft Word e, além disso, permite o usuário iniciar um documento a partir de uma série de *templates* pré-definidos.

Para a escrita do documento desse TCC, foi escolhida essa ferramenta devido a facilidade de formatação do texto e capacidade de exportação do documento em diversos formatos suportados. Como durante o processo de escrita do documento, foi surgindo a necessidade de exportação do arquivo em formatos como PDF e documento do *Word*, essa ferramenta se provou bastante útil para realizar tais funções.

2.2 *GIT*

Git é um sistema de controle de versão livre e *open source*, inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do *kernel* Linux, mas, atualmente, muito utilizado em projetos de pequeno e grande porte por sua eficiência, rapidez e facilidade de realizar versionamentos.

Com o objetivo de ter um controle de todos os arquivos criados, tanto no documento de TCC, quanto nas aplicações desenvolvidas, o *git* foi escolhido como ferramenta para esse propósito devido a capacidade de se manter um histórico de todas as alterações realizadas, além de permitir uma visualização de versões anteriores de cada arquivo de acordo com a necessidade.

2.3 *BITBUCKET*

Bitbucket é um serviço de hospedagem de projetos controlados através do Mercurial, que é um sistema de controle de versões distribuído, além de também suportar repositórios que usam *git*. Possui um serviço grátis e outro comercial e foi implementado usando a linguagem Python. O serviço grátis do Bitbucket permite hospedar repositórios privados com até cinco colaboradores.

Com o intuito de permitir o acesso aos arquivos do trabalho a partir de qualquer computador, enquanto surgir a necessidade, essa ferramenta foi definida por ter essa capacidade e por permitir a criação de repositórios privados sem a necessidade de um plano pago.

2.4 *XCODE*

Xcode é uma IDE (*Integrated Development Environment*) que contém várias ferramentas que auxiliam o desenvolvimento de aplicações para *MacOS*, *iOS*, *WatchOS* e *tvOS*. Lançado pela primeira vez em 2003 pela *Apple*, e com a versão mais atual lançada em maio de 2018, está disponível na *App Store* gratuitamente para usuários do Sistema Operacional OS X *El Capitan e macOS Sierra*.

A escolha da IDE para desenvolvimento do aplicativo foi limitada pois essa é a ferramenta padrão para desenvolvimento de aplicações para dispositivos *iOS*. No entanto, o *Xcode* é uma ótima ferramenta que possui diversas funcionalidades que auxiliam no desenvolvimento de aplicativos. Além disso, possui integração com o *git*, o que facilitou todo o processo de implementação do projeto.

Para a realização deste trabalho, foi utilizada a última versão disponível da ferramenta, que é a 9.4.1. Essa versão possui suporte à versão 4.2 da linguagem de programação *Swift*, que conta com novas funções e melhorias em comparação com versões anteriores.

2.5 *JUSTINMIND*

Justinmind é uma ferramenta de prototipação para aplicações web e móveis, que permite criação de telas de maneira intuitiva, permitindo edição em *drag-drop*, ajuste de tamanhos, exportar e importar *widgets*. Além disso, a ferramenta possui um aplicativo móvel que permite testar protótipos de aplicativos diretamente no celular. Essa é uma ferramenta que necessita pagamento mensal para ser utilizada, mas que possui um período de testes grátis que foi suficiente para a realização desse trabalho.

Essa ferramenta foi escolhida para criar os protótipos de telas iniciais do projeto devido a facilidade e rapidez para desenvolver as telas, além de facilitar a realização dos testes de usabilidade com seu aplicativo móvel que permite a interação com os protótipos direto pelo celular.

2.6 *HEROKU*

O *Heroku* é uma plataforma em nuvem que permite a hospedagem e monitoramento de apps de maneira fácil e acessível. Foi desenvolvida em 2007 e suporta diversas linguagens de programação como *Ruby, Java, PHP, Go, Python*, entre outras. Essa ferramenta conta com alguns planos pagos com benefícios adicionais. Entretanto, a assinatura do plano gratuito foi suficiente para a hospedagem e manutenção do serviço implementado nesse trabalho.

Com o objetivo de publicar os serviços implementados ao longo do processo de desenvolvimento desse projeto de maneira simples e rápida, o *heroku* foi definido como ferramenta para realizar essa função. A forma como o *heroku* funciona integrado com o *git*, tornou o fluxo de publicação de novas versões rápido e intuitivo, o que justifica a utilização da ferramenta.

2.7 *ASTAH PROFESSIONAL*

Astah Professional é um *software* para modelagem UML (*Unified Modeling Language*) desenvolvido pela *Change Vision Inc*. Permite a modelagem de diagramas de classe, casos de uso, sequência, entre outros. Essa ferramenta requer uma assinatura periódica para ser utilizada, mas conta com uma versão de testes gratuita por 30 dias, que foi o suficiente para a criação dos diagramas presentes nesse trabalho.

Os diagramas de classes, casos de uso e de sequência mostrados ao longo desse documento foram criados através do *Astah Professional*. Considerando a variedade de diagramas produzidos ao longo do projeto, essa ferramenta foi definida por atender essa necessidade, e por permitir a criação de tais diagramas de maneira simples e rápida.

3 PROPOSTA

O aplicativo proposto foi desenvolvido para dispositivos móveis com Sistema Operacional iOS, com o qual o usuário consegue acessar o URI usando sua conta cadastrada previamente no *site* do URI Online Judge. Além disso, o usuário poderá visualizar suas informações de perfil e submissões, consultar e detalhar problemas, visualizar lista de rankings e alterar configurações. A Figura 6 representa um diagrama de casos de uso das funcionalidades propostas para o aplicativo novo.

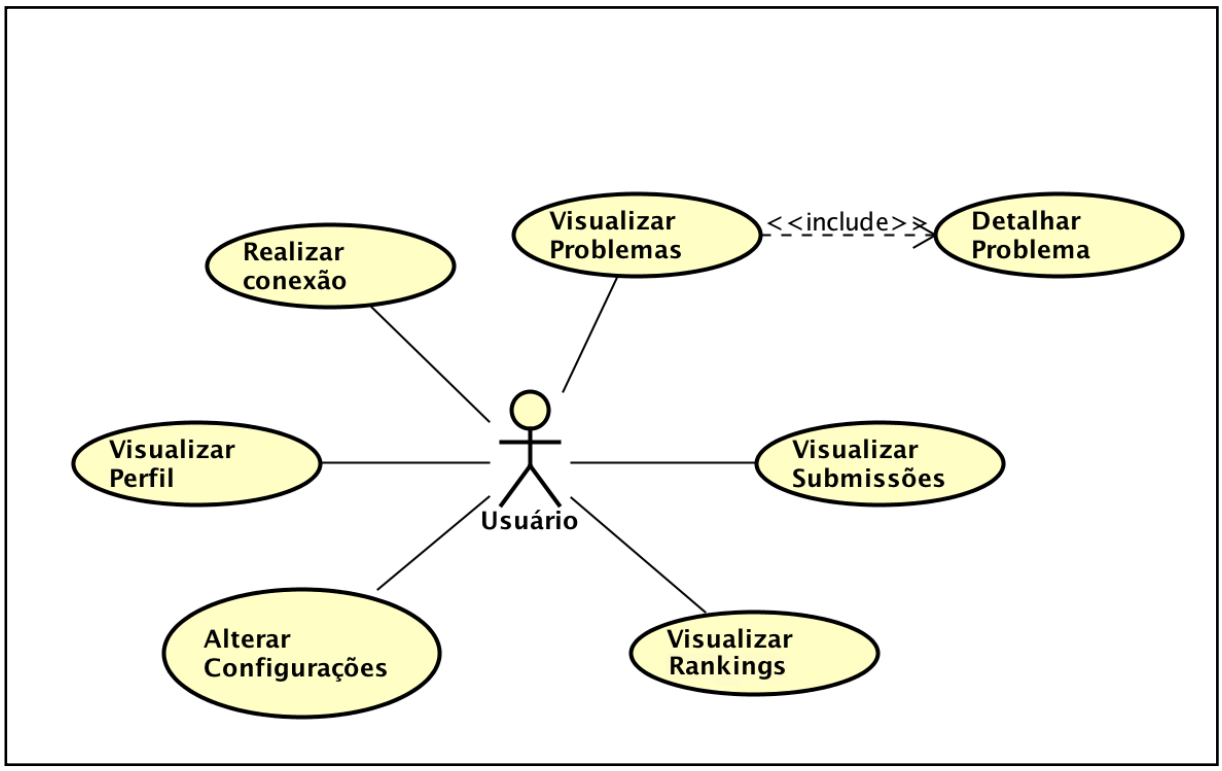


Figura 6 - Diagrama de casos de uso do aplicativo.

3.1 DESCRIÇÃO DAS FUNCIONALIDADES

Nessa seção serão descritas as funcionalidades propostas que serão implementadas nesse trabalho, baseando-se no Diagrama de Casos de Uso da Figura 6.

3.1.1 Realizar conexão

Tabela 1 – Descrição da Funcionalidade 'Realizar conexão'

Ator	Usuário
Descrição	Autentica o usuário por meio do e-mail e senha informados.
Pré-condições	O usuário deve possuir cadastro no ambiente.
Pós-condições	Acesso concedido no aplicativo
Prioridade	Alta
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário abre o aplicativo. P2 - O aplicativo mostra a tela de conexão com os campos de e-mail e senha vazios conforme a Figura 7. P3 - O usuário preenche o campo de e-mail com seu e-mail cadastrado. P4 - O usuário preenche o campo de senha com a sua senha cadastrada. P5 - O usuário pressiona o botão "ENTRAR". P6 - O aplicativo valida as informações preenchidas nos campos e autentica a conexão do usuário.
Fluxo Alternativo	A1 - Conexão pelo *Facebook* A1.1 - O usuário pressiona o botão "Facebook" A1.2 - O aplicativo apresenta uma tela onde o usuário deve entrar com seus dados cadastrados no Facebook. A1.3 - O usuário deve autorizar o acesso do app às suas informações. A1.4 - O aplicativo retorna para a tela de conexão e realiza a autenticação do usuário. A2 - Conexão pelo Google A2.1 - O usuário pressiona o botão "Google" A2.2 - O aplicativo apresenta uma tela onde o usuário deve entrar com seu e-mail e senha da Google. A2.3 - O usuário deve autorizar o acesso às informações do e-mail informado. A2.4 - O aplicativo retorna para a tela de conexão e autentica o usuário.

Fluxo de exceção	E1 - Dados inválidos E1.1 - O usuário preenche os campos email e senha com dados inválidos. E1.2 - O usuário pressiona o botão “Entrar” E1.3 - O aplicativo apresenta um alerta informando que os dados informados são inválidos com um botão “OK”. E1.4 - Ao pressionar o botão “OK”, o aplicativo apresenta a tela de conexão para o usuário preencher os campos novamente.

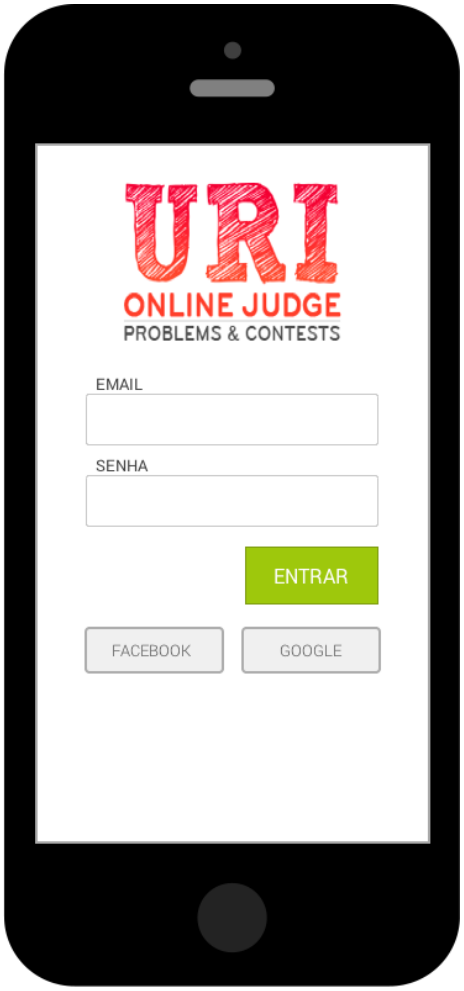


Figura 7 - Protótipo da tela de conexão.

3.1.2 Visualizar perfil

Tabela 2 – Descrição da Funcionalidade 'Visualizar perfil'

Ator	Usuário
Descrição	Visualizar informações de perfil do usuário
Pré-condições	O usuário deve ter efetuado a conexão no aplicativo
Pós-condições	Tela com informações de perfil do usuário
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário efetua conexão no aplicativo. P2 - O usuário pressiona o botão "Perfil" na barra inferior da tela principal do aplicativo. P3 - O aplicativo apresenta uma tela contendo as informações de perfil do usuário conforme a Figura 8. P4 - O caso de uso é encerrado.
Fluxo Alternativo	Nenhum
Fluxo de exceção	Nenhum

Figura 8 - Protótipo da tela de visualização de perfil.

3.1.3 Visualizar problemas

Tabela 3 – Descrição da Funcionalidade ‘Visualizar problemas

Ator	Usuário
Descrição	Visualizar lista de problemas
Pré-condições	O usuário deve ter efetuado conexão no aplicativo
Pós-condições	Tela com lista de problemas do URI Online Judge
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário efetua a conexão no aplicativo. P2 - O usuário pressiona o botão “Problemas" na barra inferior da tela principal do aplicativo. P3 - O aplicativo apresenta uma lista de categorias de problemas conforme a Figura 9. P4 - O usuário seleciona a categoria desejada. P5 - O aplicativo apresenta uma tela com a lista de problemas da categoria selecionada. P6 - O caso de uso é encerrado.
Fluxo Alternativo	Nenhum
Fluxo de exceção	Nenhum

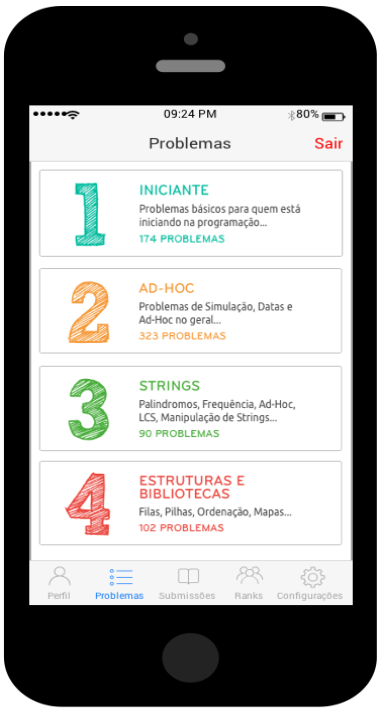


Figura 9 - Protótipo da tela de lista de categorias.

3.1.4 Detalhar problema

Tabela 4 – Descrição da Funcionalidade 'Detalhar problema'

Ator	Usuário
Descrição	Visualizar descrição de um problema
Pré-condições	O usuário deve ter efetuado conexão no aplicativo
Pós-condições	Tela com descrição de um problema
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário visualiza uma lista de problemas conforma a Figura 10. P2 - O usuário seleciona um problema da lista sendo mostrada. P3 - O aplicativo carrega os detalhes do problema selecionado e mostra em uma tela de descrição conforma a Figura 11. P4 - O caso de uso é encerrado.
Fluxo Alternativo	Nenhum
Fluxo de exceção	E1 - Erro no carregamento E1.1 - Ocorre um erro ao carregar os detalhes do problema selecionado pelo usuário. E1.2 - O aplicativo apresenta um alerta informando a causa do erro com um botão "OK". E1.3 - Ao tocar no botão "OK", o aplicativo volta para a tela com a lista de problemas.

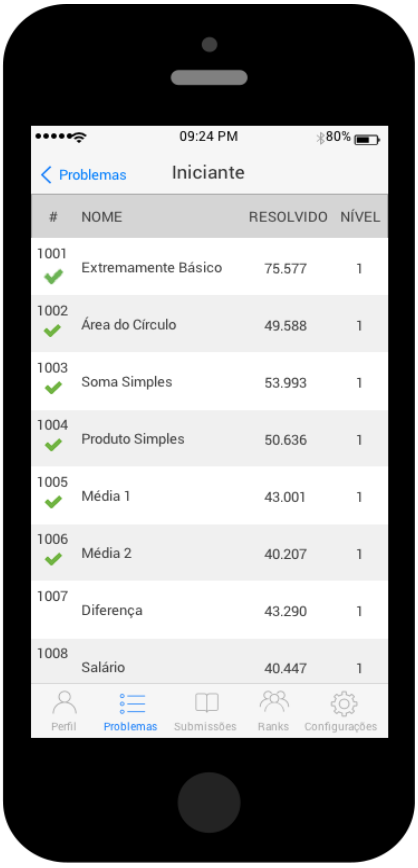


Figura 10 - Protótipo da tela de lista de problemas.

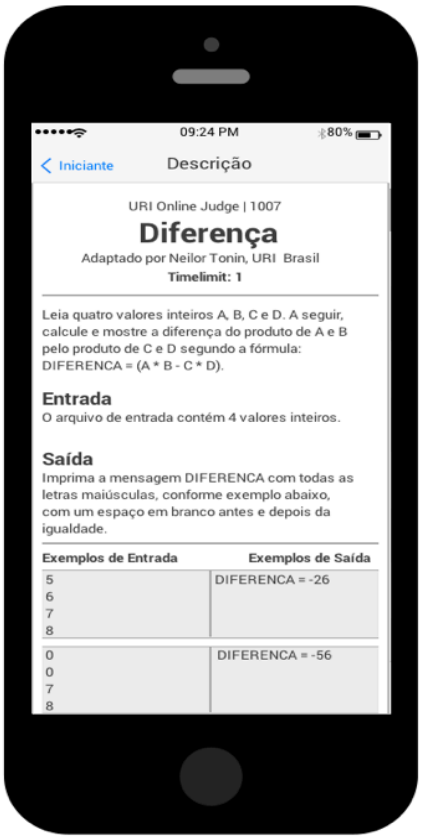


Figura 11 - Protótipo da tela de detalhamento de problema.

3.1.5 Visualizar submissões

Tabela 5 – Descrição da Funcionalidade 'Visualizar submissões'

Ator	Usuário
Descrição	Visualizar submissões feitas no site do URI Online Judge
Pré-condições	O usuário deve ter efetuado conexão no aplicativo
Pós-condições	Tela com lista de submissões
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário efetua a conexão no aplicativo. P2 - O usuário pressiona o botão "Submissões" na barra inferior da tela principal do aplicativo. P3 - O aplicativo mostra uma tela, conforme a Figura 12, com a lista de submissões feitas no site do URI Online Judge pela conta do usuário que efetuou a conexão. P4 - O caso de uso é encerrado.
Fluxo Alternativo	A1 - Mostrar mais informações A1.1 - O usuário rotaciona o dispositivo na orientação *landscape.* A1.2 - O Aplicativo reorganiza as colunas da tabela para mostrar mais informações na tela, aproveitando o espaço horizontal adicional.
Fluxo de exceção	Nenhum

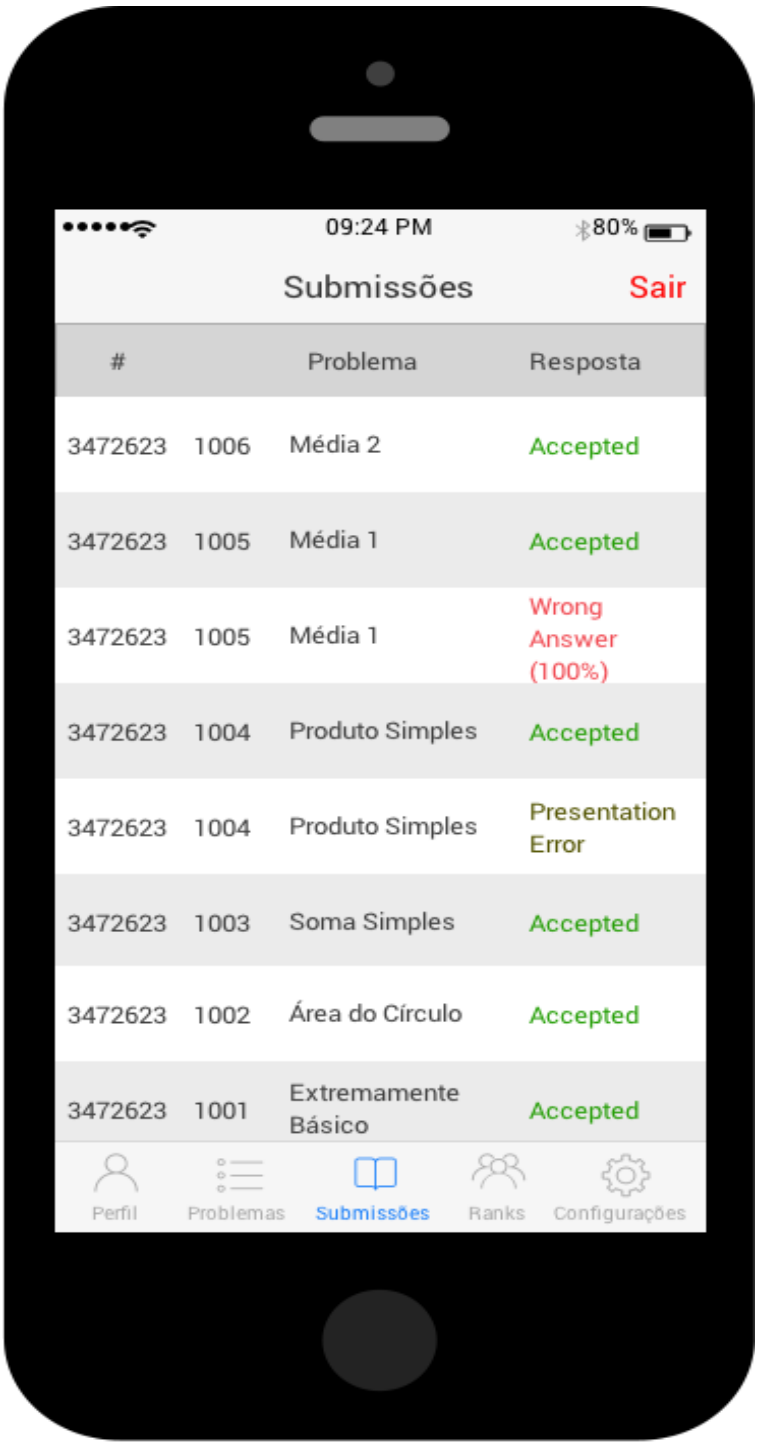


Figura 12 - Protótipo da tela de lista de submissões.

3.1.6 Visualizar rankings

Tabela 6 – Descrição da Funcionalidade 'Visualizar rankings'

Ator	Usuário
Descrição	Visualizar rankings do URI Online Judge
Pré-condições	O usuário deve ter efetuado conexão no aplicativo
Pós-condições	Tela com lista de rankings
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário efetua a conexão no aplicativo. P2 - O usuário pressiona o botão "*Ranks*" na barra inferior da tela principal do aplicativo. P3 - O aplicativo mostra uma tela com a lista de rankings conforme a Figura 13. P4 - O usuário seleciona o tipo de ranking que deseja visualizar. P4 - O caso de uso é encerrado.
Fluxo Alternativo	A1 - Ranking por Universidades A1.1 - O aplicativo mostra uma lista com o ranking por universidades conforma a Figura 14. A2 - Ranking por País A2.1 - O aplicativo mostra uma lista com o ranking por país. A3 - Ranking por Usuários A3.1 - O aplicativo mostra uma lista com o ranking de usuários. A4 - Encontre-me no ranking A4.1 - O aplicativo apresenta a lista de ranking de usuários mostrando o usuário logado atualmente na lista.
Fluxo de exceção	Nenhum

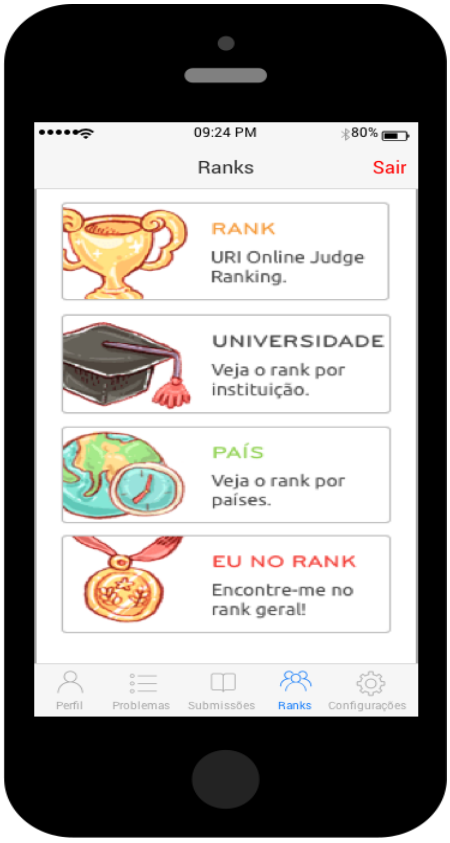


Figura 13 - Protótipo da tela de lista de rankings.

Figura 14 - Protótipo da tela de ranking por universidades.

3.1.7 Alterar configurações

Tabela 7 – Descrição da Funcionalidade ‘Alterar configurações’

Ator	Usuário
Descrição	Visualizar e alterar informações gerais
Pré-condições	O usuário deve ter efetuado conexão no aplicativo
Pós-condições	Tela de Configurações com dados alterados exibidos.
Prioridade	Média
Fluxo Principal	P1 - O caso de uso é iniciado quando o usuário efetua a conexão no aplicativo. P2 - O usuário pressiona o botão “Configurações" na barra inferior da tela principal do aplicativo. P3 - O aplicativo mostra uma tela com dados de perfil do usuário logado, assim como informações de configuração do aplicativo. P4 - O caso de uso é encerrado.
Fluxo Alternativo	A1 - Alterar informações A1.1 - O usuário edita as informações mostradas na tela da forma que preferir. A1.2 - O usuário pressiona o botão “Salvar”. A1.3 - O aplicativo salva as informações atualizadas e apresenta um alerta informando o usuário do sucesso da operação.
Fluxo de exceção	E1 - Erro no salvamento das informações E1.1 - Ocorre um erro no processo de salvar as informações preenchidas pelo usuário. E1.2 - O sistema apresenta um alerta informando o usuário a causa do erro com um botão “OK”. E1.3 - Ao tocar no botão “OK” o aplicativo fecha o alerta.

Figura 15 - Protótipo da tela de configurações.

3.2 IMPLEMENTAÇÃO DAS FUNCIONALIDADES

Nessa seção será descrita, por meio de diagramas de classe, cada funcionalidade elaborada por este trabalho, além da indicação dos recursos de implementação usados. O padrão de arquitetura adotado foi o MVC, esclarecido anteriormente, sendo empregada a linguagem de programação *Swift (*APPLEDEVELOPER*, 2016)*. A modelagem das camadas principais do projeto está representado no diagrama de componentes da Figura 16.

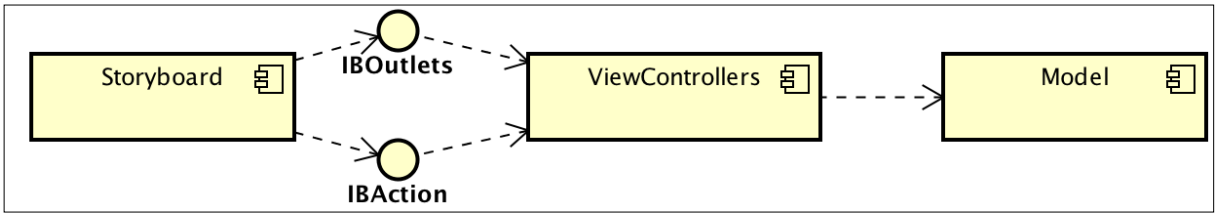


Figura 16 - Diagrama de Componentes do Projeto.

3.2.1 Camada Modelo

A camada Modelo, que corresponde a *Model* da arquitetura MVC, corresponde a abstração em *software* de um problema do mundo real. As classes que fazem da implementação do projeto nessa camada, assim como o relacionamento entre elas, estão representadas no diagrama da Figura 17, considerando as funcionalidades propostas para esse trabalho.

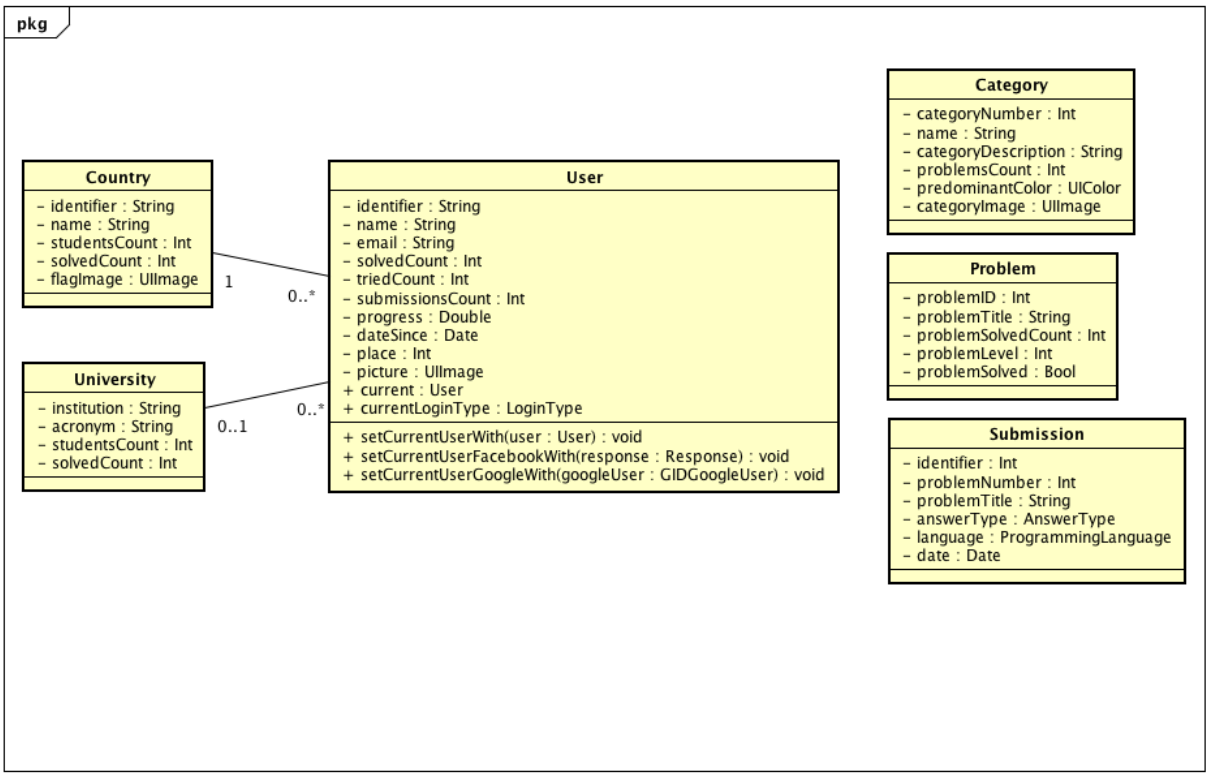


Figura 17 - Diagrama de Classes representado a camada Modelo (*Model*).

3.2.2 Perfil de usuário

O diagrama de classes que representa a funcionalidade de 'Visualizar Perfil' está representado na Figura 18.

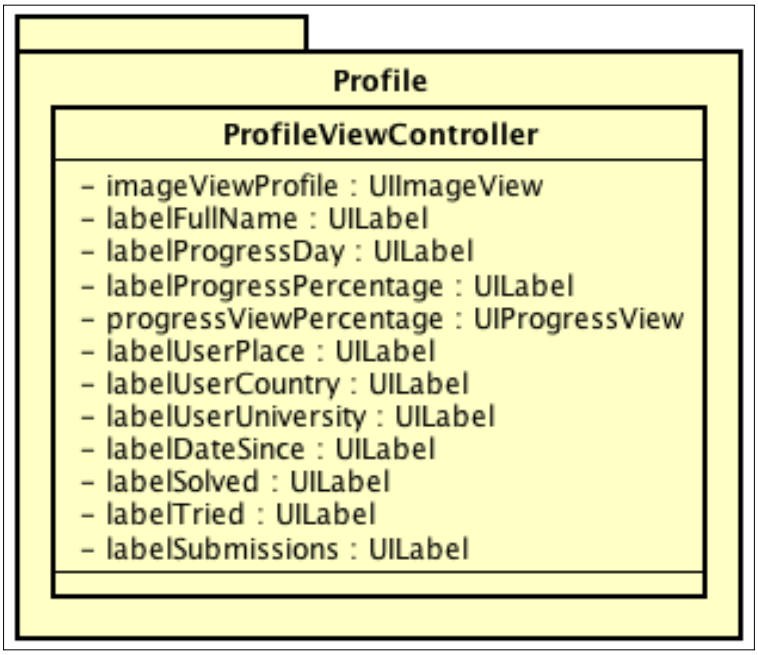


Figura 18 - Diagrama da classe Perfil de usuário (*ProfileViewController*).

3.2.3 Visualizar categorias e problemas

A Figura 19 mostra o diagrama de classes referente à funcionalidade de Visualizar categorias de problemas, além de demonstrar o relacionamento com a classe da camada Modelo relativa.

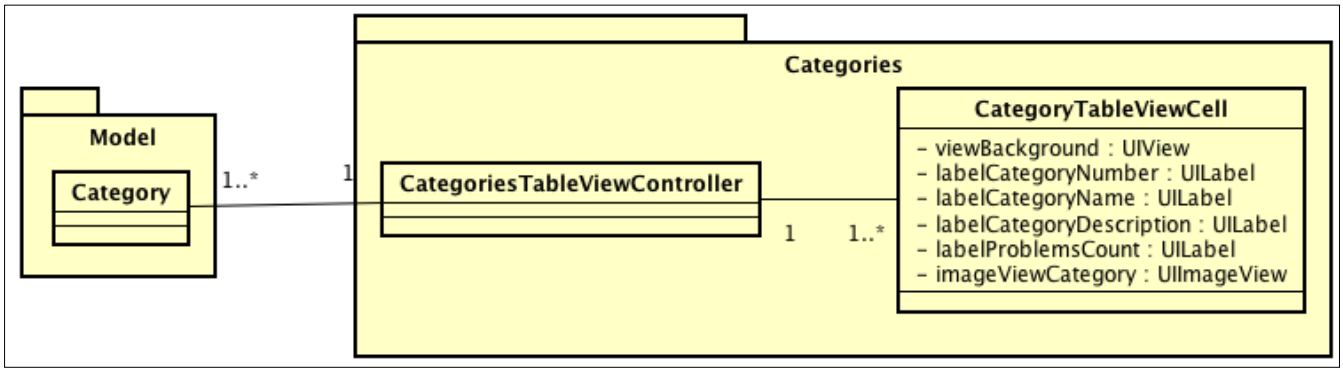


Figura 19 - Diagrama da classe Visualizar Categorias.

Na Figura 20 são mostradas as classes que pertencem à funcionalidade ‘Visualizar Problemas’ de acordo com a categoria de problemas referente. Além disso, a classe '*ProblemDetailViewController*' é responsável pela funcionalidade de detalhar determinado problema de acordo com o caso de uso ‘Detalhar Problema’.

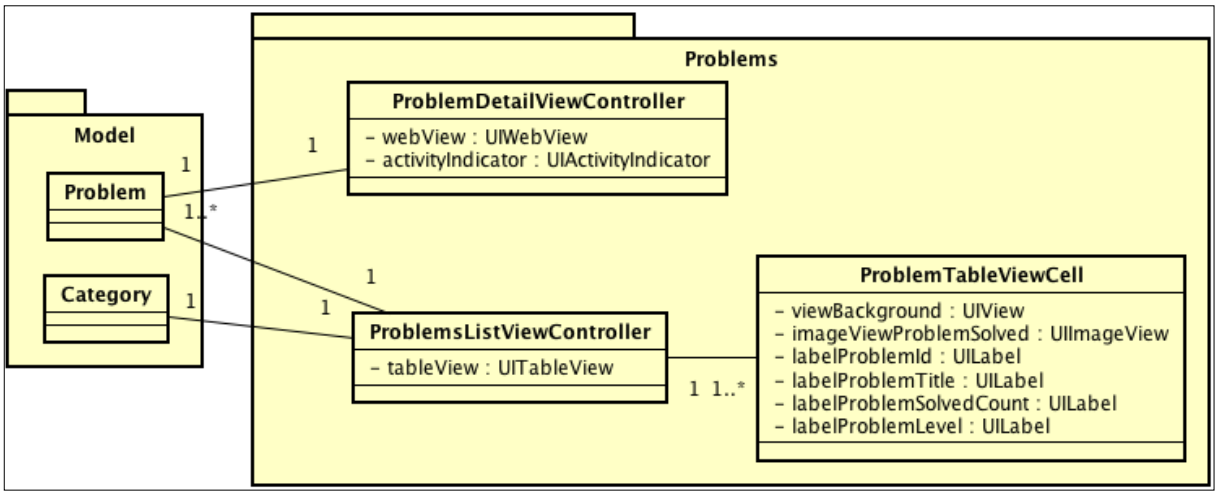


Figura 20 - Diagrama da classe Visualizar Problemas.

3.2.4 Visualizar submissões

A Figura 21 representa o diagrama de classes para a funcionalidade ‘Visualizar Submissões’ com seu relacionamento com as classes da camada Modelo da arquitetura. Nesse diagrama é mostrada a classe ‘*DynamicOrientationViewController’* que

implementa a capacidade do aplicativo de suportar a orientação *landscape* do dispositivo com o intuito de aumentar o número de informações disponíveis na tela para o usuário.

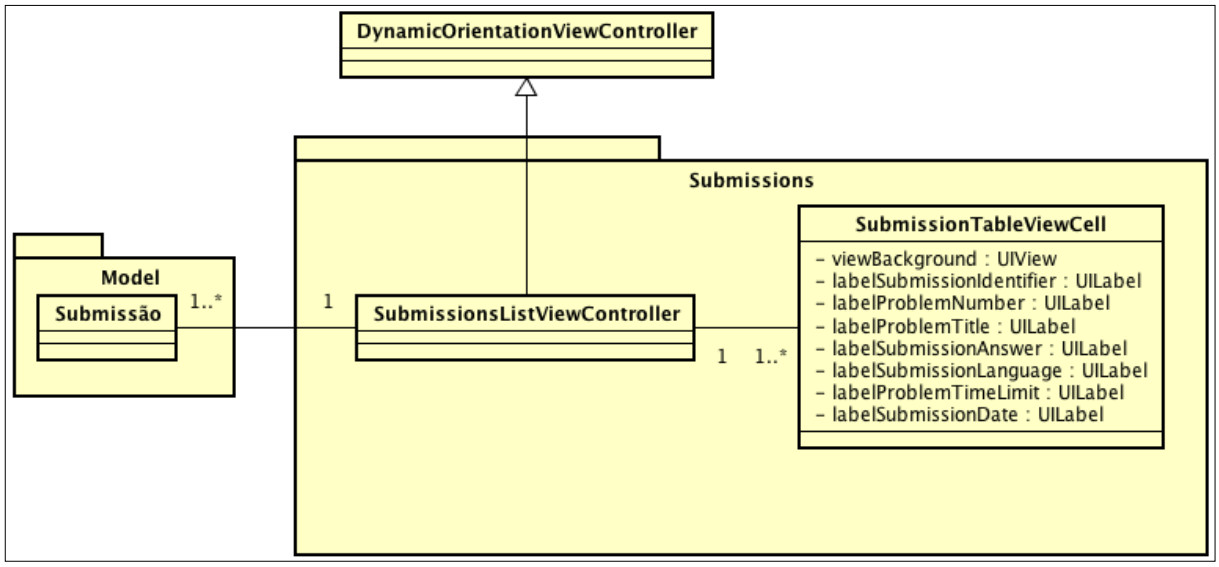


Figura 21 - Diagrama da classe Visualizar Submissões.

3.2.5 Visualizar rankings

A Figura 22 mostra o diagrama de classes referente à funcionalidade de visualizar lista de tipos de rankings, além de mostrar o relacionamento entre as classes de rankings suportadas pelo aplicativo.

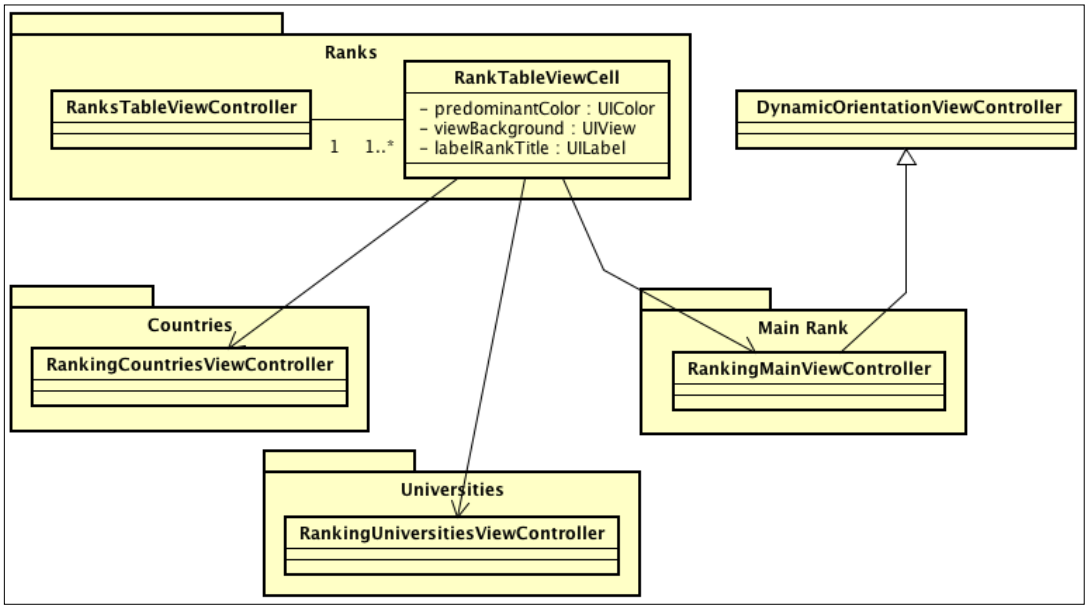


Figura 22 - Diagrama da classe Visualizar Rankings.

3.2.6 *Storyboard* do projeto

A *Storyboard* é uma representação das telas do projeto e o relacionamento entre elas, mostrando uma visão completa do fluxo da aplicação (APPLE, 2016). Nesse projeto, as *storyboards* representam a camada de Visão (View) da arquitetura MVC utilizada.

Na Figura 23 é apresentada a *storyboard* referente às telas da funcionalidade de visualizar e detalhar problemas. É possível verificar a navegação que se inicia na listagem das categorias de problemas, em seguida a listagem de problemas referentes a categoria selecionada, e finalmente, o detalhamento do problema escolhido.

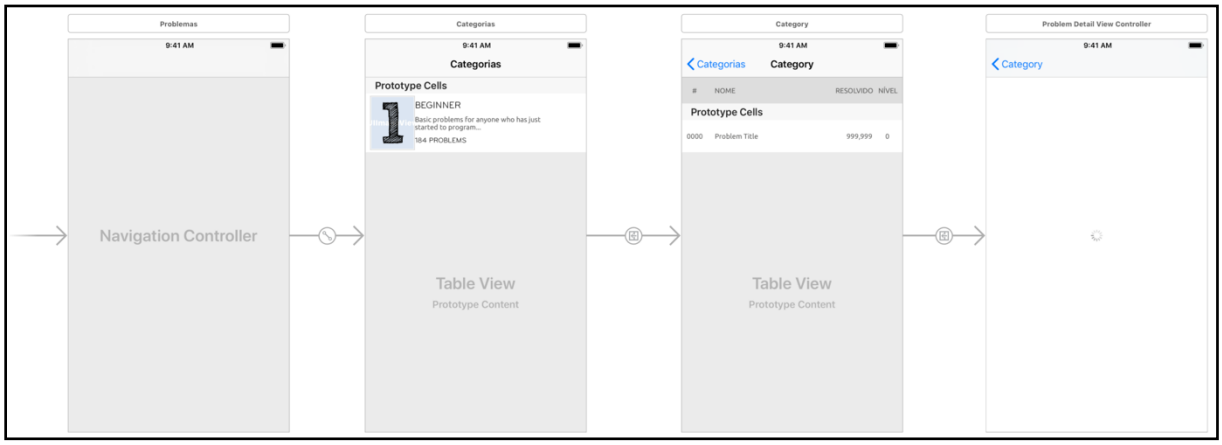


Figura 23 – *Storyboard:* Visualizar problemas.

Na Figura 24 é mostrada a *storyboard* referente às telas da funcionalidade de listar rankings. É possível perceber a separação das diferentes listagens de rankings de acordo com a opção selecionada pelo usuário. A navegação é diferente para cada tipo de ranking suportado (países, universidades e usuários), e sempre se inicia com a lista inicial, onde o usuário escolhe qual ranking deseja visualizar.

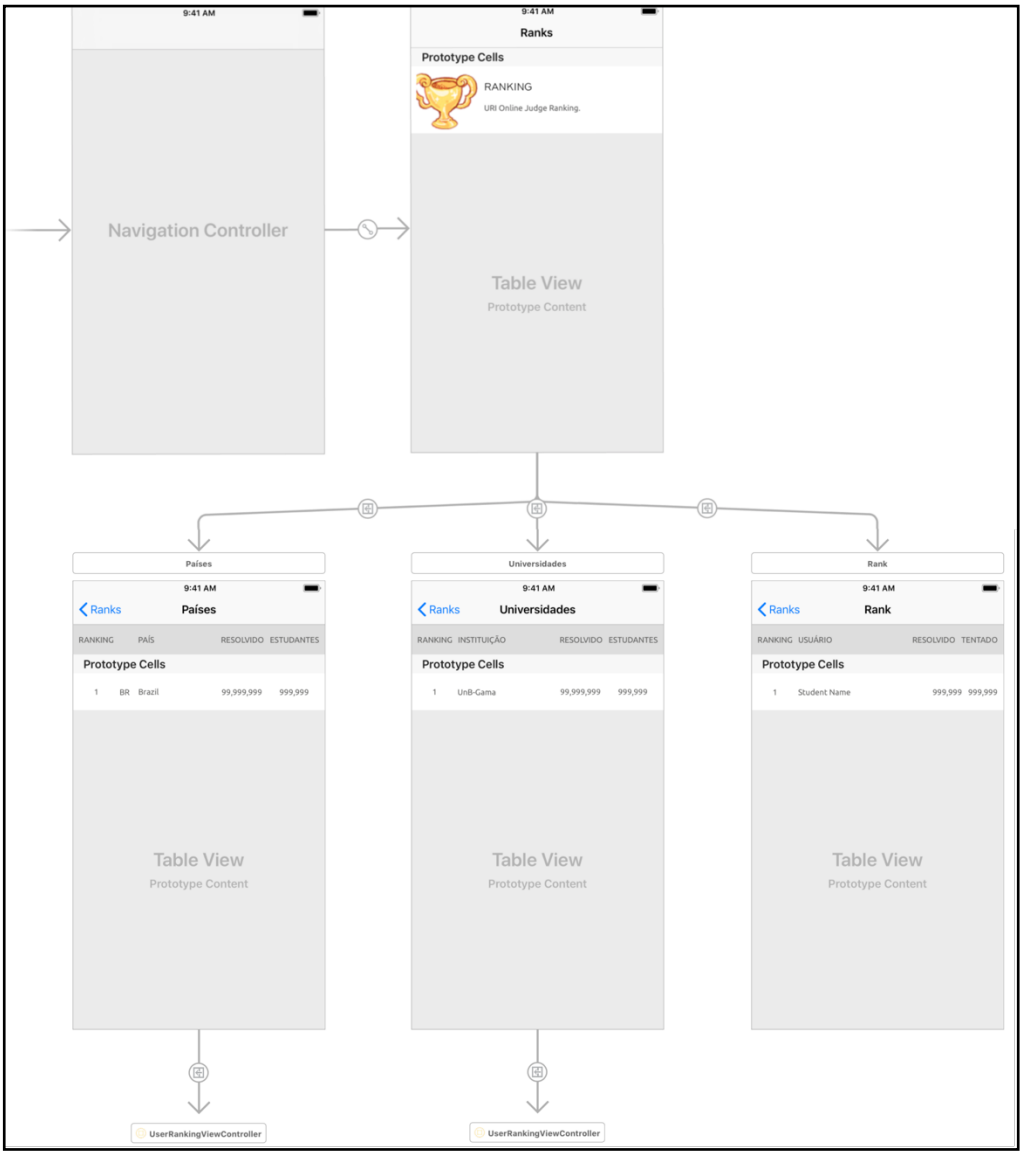


Figura 24 – *Storyboard:* Lista de rankings.

3.3 API PARA CONSULTA DE DADOS

Para a disponibilização de uma versão inicial do aplicativo é necessária a implementação de um sistema que realiza a busca de informações diretamente no *site* do URI *Online Judge*, utilizando a técnica de *Web Scraping*. Essa necessidade surge devido a inexistência de uma API fornecida pelos criadores do URI. As informações obtidas dessa forma serão disponibilizadas por meio de uma API que deve ser consultada pelo aplicativo.

Existem duas abordagens possíveis a serem consideradas para o desenvolvimento dessa solução com API. A primeira abordagem seria implementar um sistema

que realiza a busca em todas as páginas do *site* do *URI Online Judge* de forma periódica, por exemplo uma vez por dia, e armazena todas as informações obtidas em um banco de dados para consultas posteriores. O problema dessa abordagem é a possibilidade do banco de dados criado conter dados desatualizados, já que existe a chance de um novo elemento ser adicionado ao sistema do URI a qualquer momento, mas que só será acrescentado à nova base quando uma nova busca (atualização) for realizada.

A implementação de um sistema que realiza a busca por dados sob demanda seria a segunda abordagem, na qual haveria a requisição solicitada pelo aplicativo e o novo sistema buscaria a página correspondente no *site* do URI e retornaria as informações respectivas para essa requisição. A desvantagem nessa abordagem aconteceria porque em cada requisição feita pelo aplicativo uma nova requisição deverá ser feita pela API ao *site* do URI, o que deixará todo o processo mais lento.

Considerando a complexidade da implementação e a confiabilidade dos dados fornecidos pela API a ser desenvolvida, a segunda abordagem foi escolhida para ser seguida na implementação dessa API. Embora possa ser mais lento o processamento, esta escolha prioriza a segurança e o sincronismo com o URI.

4 RESULTADOS OBTIDOS

4.1 TESTES REALIZADOS

Para a realização dos testes de usabilidade para o protótipo inicial desenvolvido, os usuários foram orientados a simular a realização de uma série de tarefas pré-definidas com o intuito de determinar o tempo levado para executar cada tarefa, possibilitando, assim, a descoberta de pontos de melhoria na interface gráfica do aplicativo.

Considerando o gráfico de confiabilidade dos testes por número de testadores e o número ideal de testadores necessários para garantir um melhor aproveitamento dos testes realizados, proposto por Nielsen (1994 e 2000), esses testes foram realizados com cinco usuários. As tarefas executadas por eles, juntamente com o tempo médio levado para sua execução, são mostradas na Tabela 8.

Tabela 8 – Tarefas e tempos médios de execução

Nº	TAREFA	TEMPO MÉDIO PARA EXECUÇÃO	DESVIO PADRÃO
1	Faça conexão no aplicativo usando **e-mail** e **senha**.	21s	6,59s
2	Verifique as suas submissões.	10s	1,58s
3	Visualize o ranking por instituições.	13s	2,73s
4	Saia da sua conta e, em seguida, faça o acesso pelo *Facebook*.	22s	3,67s
5	Verifique o problema com título 'Diferença' na categoria Iniciante.	15s	3,53s
6	Altere o ícone do perfil para o do *Facebook* e o idioma do aplicativo para inglês.	29s	4,94s
7	Saia da sua conta e, em seguida, entre pelo *Google+*.	6s	1,41s

4.2 API INTERMEDIÁRIA

A aplicação denominada de API intermediária serve como uma interface entre o aplicativo móvel e a base de dados do URI *Online Judge.* Essa API é responsável por receber requisições do aplicativo móvel e retornar um objeto no formato JSON (*JavaScript Object Notation)* correspondente.

O fluxo seguido a cada requisição feita pelo aplicativo para a API intermediária é mostrado no diagrama de sequência na Figura 25.

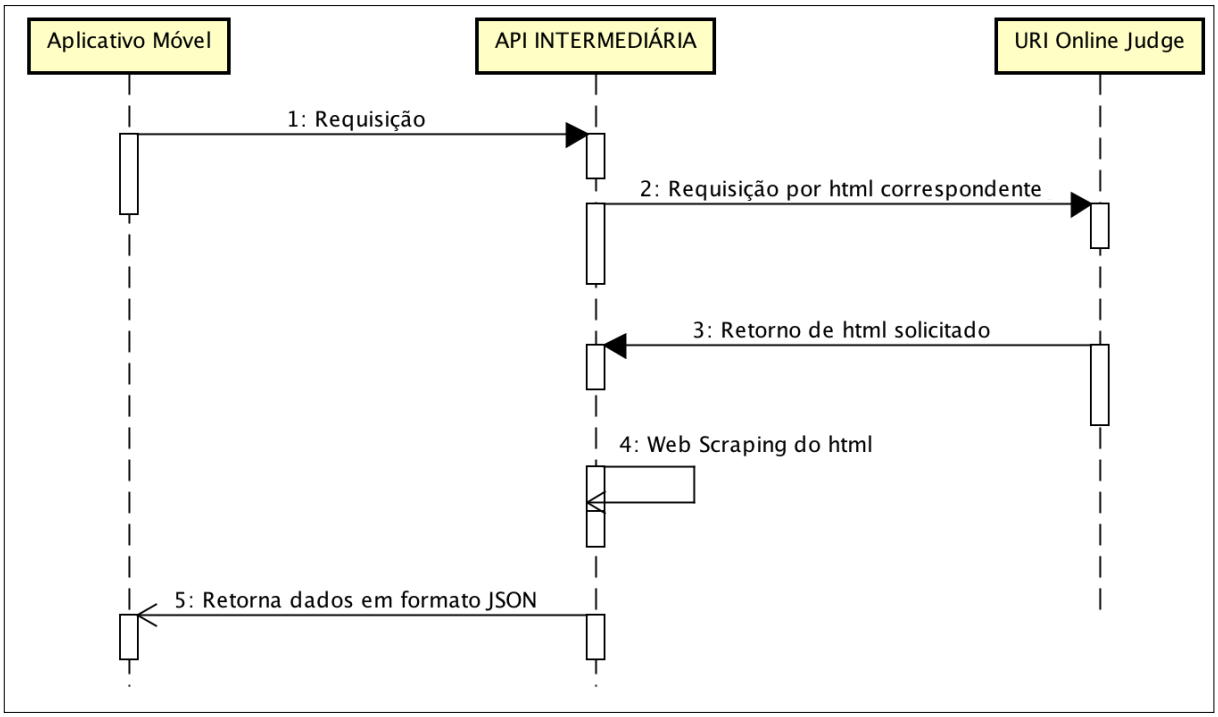


Figura 25 – Diagrama representando a *API Intermediária.*

A API intermediária foi implementada utilizando a linguagem de programação *Ruby* e foi hospedada através da ferramenta *Heroku*. A lista com todos os *endpoints* suportados por essa API, bem como exemplos dos dados retornados, está disponível no Apêndice A desse trabalho.

4.3 APLICATIVO MÓVEL

A implementação do aplicativo móvel com todas as funcionalidades propostas nesse trabalho no período de tempo determinado inicialmente, tornou-se tecnicamente inviável devido a inexistência de uma API para integração com a base de dados do sistema URI *Online Judge*. Entretanto, uma versão considerada inicial, foi implementada utilizando, como fonte de dados, a API intermediária descrita anteriormente.

Devido a limitação nos dados que podem ser fornecidos por essa API, as funcionalidades disponíveis nessa versão do aplicativo também foram limitadas. A Figura 26 mostra o diagrama de casos de uso com as funcionalidades que estão presentes na versão inicial do aplicativo.

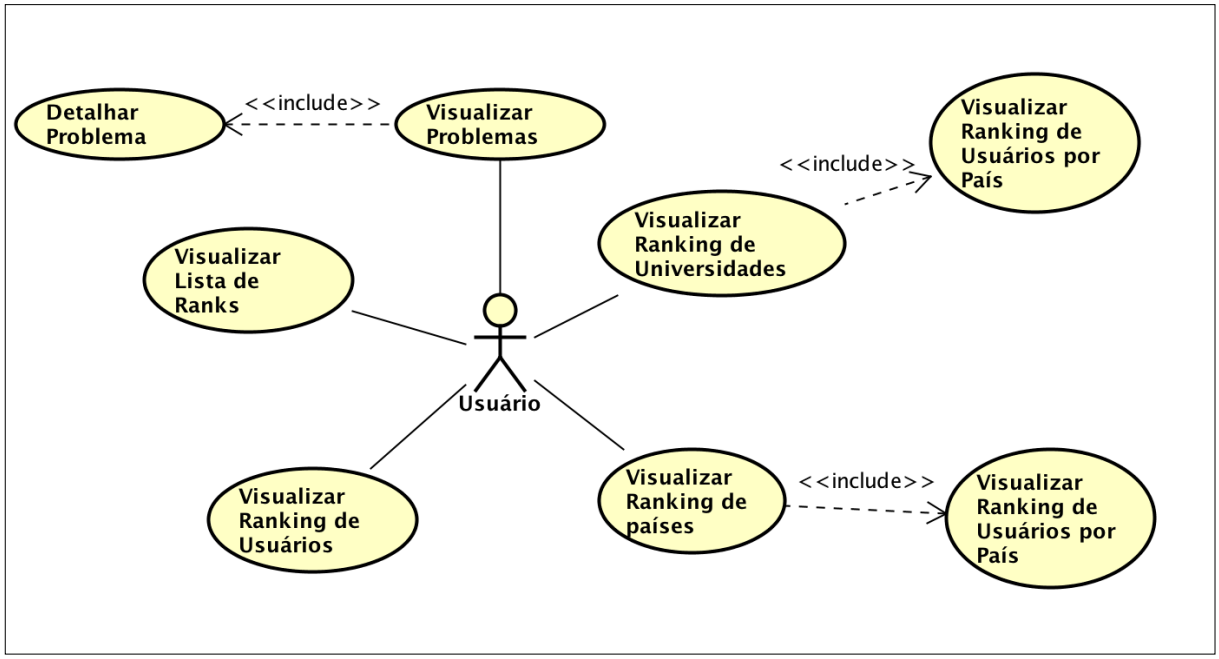


Figura 26 - Diagrama de casos de uso de versão inicial do aplicativo.

A fim de evitar requisições repetidas e desnecessárias à API intermediária foi implementada uma lógica de *cache* local dos dados obtidos pelas requisições, ou seja, para cada requisição bem-sucedida para a API, o aplicativo salva no próprio dispositivo os dados recebidos para que sejam consultados posteriormente sem que seja necessária uma nova requisição.

As Figuras de 27 até 31 mostram algumas telas presentes no aplicativo móvel ao término do processo de desenvolvimento.

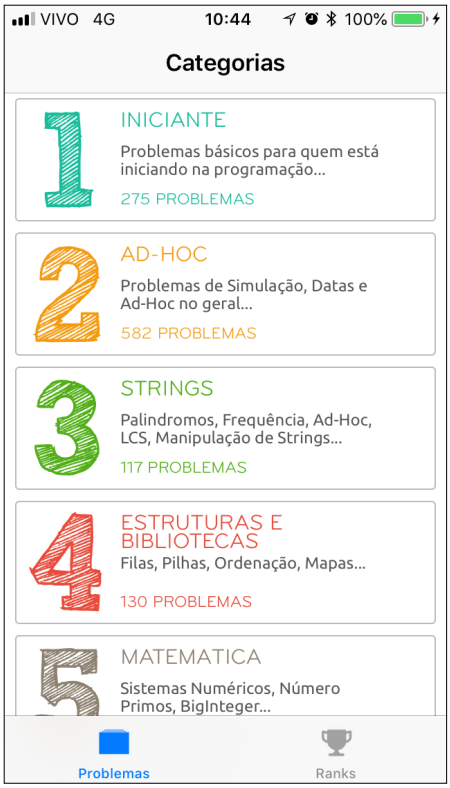


Figura 27 - Lista de Categorias.

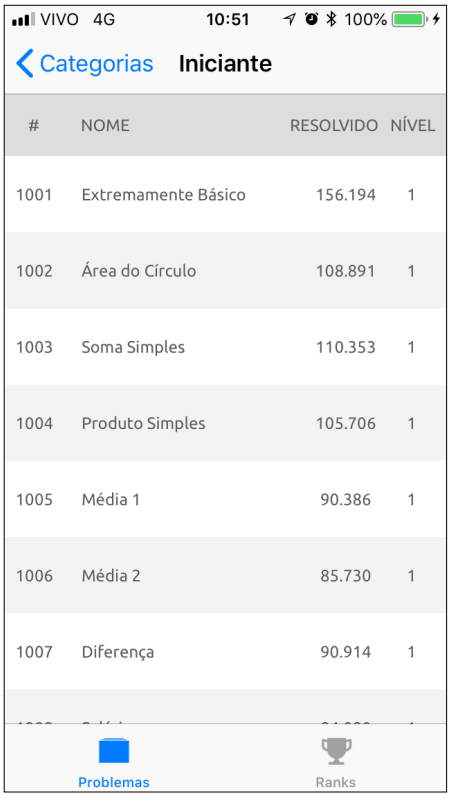


Figura 28 – Apresentação da Lista de Problemas.

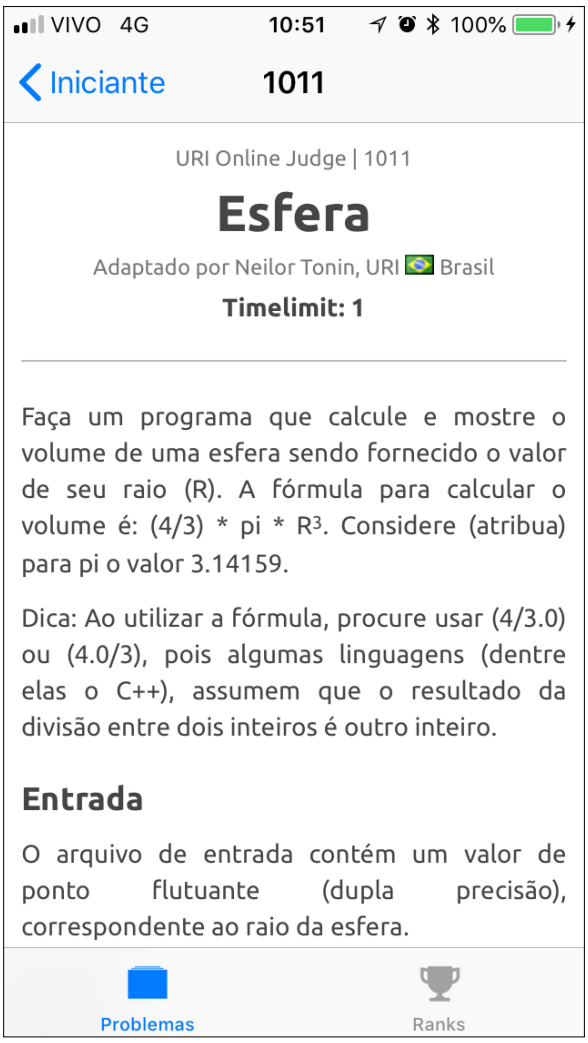


Figura 29 - Detalhamento de Problema.

A Figura 29 mostra a tela de detalhamento de um problema contendo como título da tela, o número do problema. Este número é único para cada problema e previne que problemas com títulos muito grandes não caibam na barra de título. Esse problema aconteceria com a tela como havia sido planejada inicialmente (com o título da tela sendo o título do problema), mas foi prevenido com essa alteração.

As Figuras 30 e 31 mostram as telas com a lista de tipos de rankings suportados no aplicativo e rankings por usuários, respectivamente. Cada tipo aparece com sua imagem respectiva como acontece no *site* URI *Online Judge.*

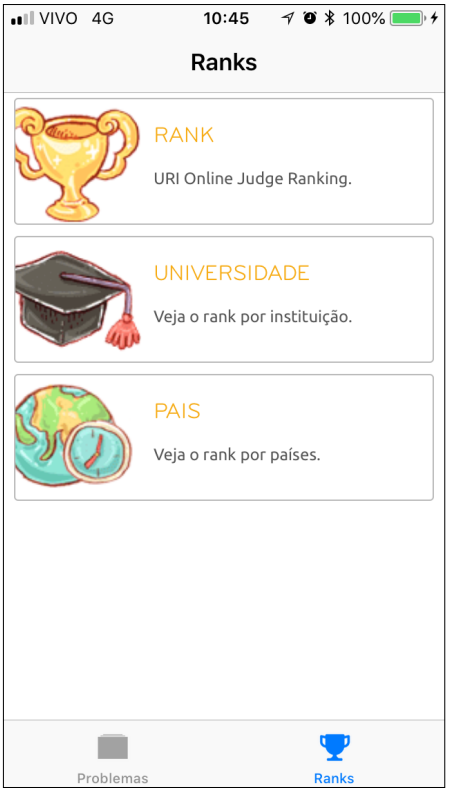


Figura 30 - Lista de Rankings.

Figura 31 - Ranking de Usuários.

A versão inicial do aplicativo móvel implementada e especificada nesse capítulo servirá como uma forma de determinar o nível de aceitação dos usuários a respeito do projeto, e incentivará a evolução contínua do aplicativo, com possíveis sugestões de melhorias e propostas para novas funcionalidades.

As funcionalidades propostas inicialmente que não foram disponibilizadas na versão inicial, como por exemplo a possibilidade de entrar no aplicativo utilizando uma conta criada no *site* do URI *Online Judge* ou de visualizar a lista de submissões do usuário conectado, foram implementadas parcialmente com dados fictícios sendo mostrados para o usuário. Por esse motivo, essas telas foram removidas dessa versão inicial, mas serão adicionadas novamente assim que a integração com os dados do URI *Online Judge* for possível. Como as telas referentes a essas funcionalidades estão parcialmente prontas, a integração com dados reais da ferramenta depende apenas da autorização e disponibilização de uma forma de acesso aos dados necessários por parte da equipe responsável por manter o projeto URI *Online Judge.*

O contato com os autores do projeto em questão foi mantido para este fim, mas até o momento de finalização desse projeto, o acesso necessário a esses dados ainda não foi possível devido a ausência de uma API disponibilizada para esse propósito. Portanto, a continuação do projeto descrito inicialmente, com a implementação de todas as funcionalidades propostas, será executada como um trabalho futuro, como estará melhor descrito no capítulo final desse trabalho.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Ao longo do trabalho, foi possível aplicar o conhecimento adquirido ao longo do curso de Engenharia de Software para planejar e executar o processo de desenvolvimento das soluções apresentadas nesse documento. A metodologia definida foi seguida de modo que os objetivos especificados inicialmente foram atendidos ao final de todo o processo.

Esse trabalho teve como objetivo a criação de um protótipo de telas e a submissão de tal protótipo a uma série de testes de usabilidade com possíveis usuários, visando a detecção de problemas de usabilidade nas telas desenhadas. Tudo isso para desenvolver um aplicativo móvel para dispositivos iOS com uma usabilidade intuitiva seguindo os padrões de interface definidos pelo URI *Online Judge*. Como evidenciado ao longo do documento, esses objetivos foram alcançados, e o resultado final é um aplicativo móvel para dispositivos iOS que faz uso de dados existentes na própria plataforma do URI *Online Judge*.

A disponibilidade do produto desenvolvido na loja de aplicativos da *Apple* (*App Store*) foi impossibilitada devido à falta de autorização providenciada pelos criadores do URI *Online Judge* para a publicação. Uma solicitação por e-mail foi enviada aos responsáveis, mas até o momento de finalização desse trabalho o pedido não havia sido respondido. A publicação da versão inicial do aplicativo na *App Store* será realizada assim que essa autorização for obtida.

Com a finalidade de disponibilizar a versão inicial do aplicativo para um grupo selecionado de usuários, antes da publicação do mesmo na loja, foi empregada a ferramenta *Testflight*. Essa ferramenta é uma plataforma de testes disponibilizada pela *Apple* que tem o intuito de permitir que um grupo de usuários tenham acesso e testem determinado aplicativo antes que ele seja lançado na *App Store*. Para que um usuário possa participar de testes por essa ferramenta é necessário que ele possua um dispositivo iOS com o aplicativo *Testflight* instalado. Com um convite enviado pelo desenvolvedor é possível instalar o aplicativo no dispositivo e iniciar os testes. Essa abordagem servirá como uma alternativa para a disponibilização do aplicativo desenvolvido para alguns usuários selecionados, até que a publicação do aplicativo na *App Store* seja efetuada.

5.2 TRABALHOS FUTUROS

O aplicativo desenvolvido consiste em uma versão inicial que implementa uma arquitetura de ampliação para interação com o URI *Online Judge* através de dispositivos móveis. Reconhecendo as habilidades e conhecimentos dos envolvidos no ambiente iOS, o projeto propôs e implementou essa arquitetura no protótipo planejado com sucesso, demonstrando mais uma alternativa de expansão ao projeto URI *Online Judge*.

No entanto, a continuidade evolutiva dessa proposta e, possivelmente, corretiva em algum ponto que seja mais adequado a equipe responsável pelo URI poderão estar acontecendo, entre elas o desenvolvimento sobre essa arquitetura para outros ambientes operacionais como para Android e Windows Phone.

Novas funcionalidades poderão ser agregadas a essa arquitetura simples, mas eficiente na ampliação do ambiente atual do URI, oferecendo possibilidades de trabalhos futuros interessantes ao projeto original, além de promover sua evolução contínua de acordo com novas demandas. Entre estas possibilidades algumas indicações são relacionadas abaixo e já foram apresentadas como ideias novas para a equipe responsável pelo URI como possíveis novas funcionalidades para o aplicativo:

- resultados em tempo real de todas as submissões enviadas pelos usuários;
- notificação no dispositivo com o resultado da última submissão do usuário conectado;
- possibilidade de marcação de problemas para serem resolvidos posteriormente;
- integração com o ambiente acadêmico do URI;
- notificações para alunos informando a existência de novas listas de exercícios no ambiente acadêmico do URI;

Essas possibilidades atenderiam melhor e em um número ainda maior os usuários do URI *Online Judge*.

REFERÊNCIAS BIBLIOGRÁFICAS

Agni, Edu, 2015, “As oito regras de ouro do design de interfaces”. Disponível em <http://www.uxdesign.blog.br/design-de-interfaces/oito-regras-de-ouro/>

Alves, Rafael Ferreira e VANALLE, Rosângela Maria, 2001, “Ciclo de Vida de Desenvolvimento de Sistemas - Visão Conceitual dos Modelos Clássico, Espiral e Prototipação”. Disponível em <http://www.abepro.org.br/biblioteca/ENEGEP2001_TR93_0290.pdf>

Apple, 2016, “Interface Builder Built-In”. Disponível em <https://developer.apple.com/xcode/interface-builder/>

Apple Developer, 2016, “Swift 3”. Disponível em <https://developer.apple.com/swift/>

Blog Brasil Westcon, 2017. “O que é Web Scraping?”. Disponível em <http://blogbrasil.westcon.com/o-que-e-web-scraping>

Camarini, Bruno, 2013, “Prototipação e sua Importância no Desenvolvimento de Software”. Disponível em <http://dextra.com.br/pt/prototipacao-e-sua-importancia-no-desenvolvimento-de-software/>

Carvalho, Ana Amélia Amorim, 2002, “Testes de Usabilidade: exigência supérflua ou necessidade?”. Actas do 5º Congresso da Sociedade Portuguesa de Ciências da Educação. Lisboa: Sociedade Portuguesa de Ciências da Educação, 235-242.

Gong, Jun and Tarasewich, Peter, 2007, "Interface Design for Handheld Mobile Devices”. Disponível em <https://pdfs.semanticscholar.org/4927/d3149071d3b6349e73ea6398fa8cf28e3df4.pdf>

InfoExame®, Revista Abril Edição 353, 2015, "URI Online Judge na Revista Abril InfoExame®”. Disponível em <https://www.urionlinejudge.com.br/info-exame/>

Krasner, Glenn E. and Pope, Stephen T., 1988, “A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80”. Disponível em <https://www.lri.fr/~mbl/ENS/FONDIHM/2013/papers/Krasner-JOOP88.pdf>

Mitchell, Ryan, 2015, "Web Scraping with Python: Collecting Data from the Modern Web”. Disponível em <https://books.google.com.br/books?hl=pt-BR&lr=&id=OUDfCQAAQBAJ&oi=fnd&pg=PP1&dq=web+scraping&ots=WJTGtzokM4&sig=aCTSmjPcxPV9VeqDjzomPf6uosw#v=onepage&q=web%20scraping&f=false>

Nepomuceno, Dênys, 2012, “Modelos Incremental, Espiral e de Prototipação”. Disponível em <http://engenhariadesoftwareuesb.blogspot.com.br/2012/12/blog-post.html>

Nielsen, Jakob, 1994 "Usability engineering". Elsevier.

Nielsen, Jakob, 1995 "10 usability heuristics for user interface design." Fremont: Nielsen Norman Group. Disponível na Internet

Nielsen, Jakob, 2010 "Why You Only Need to Test with 5 Users” Disponível em <http://www.mathcs.emory.edu/~cengiz/cs540-485-soft-eng-fa13/resources/5userTesting.pdf>

Rubin, Jeffrey and Chisnell, Dana, 2015, “Handbook of Usability Testing, Second Edition: How to Plan, Design, and Conduct Effective Tests”. Disponível em <http://ccftp.scu.edu.cn:8090/Download/efa2417b-08ba-438a-b814-92db3dde0eb6.pdf>

Schneiderman, Ben, 2010, “The Eight Golden Rules of Interface Design”. Disponível em <https://www.cs.umd.edu/users/ben/goldenrules.html>

URI Online Judge, 2016, “URI Online Judge Problems and Contests” Disponível em <https://www.urionlinejudge.com.br/judge/en/login>

APÊNDICE A – *ENDPOINTS* API INTERMEDIÁRIA

As Figuras 32 e 33 representam a documentação da API intermediária implementada durante o projeto e especificada no capítulo 5. Essa documentação contém a URL correspondente de cada serviço, juntamente com os parâmetros aceitados por cada um.

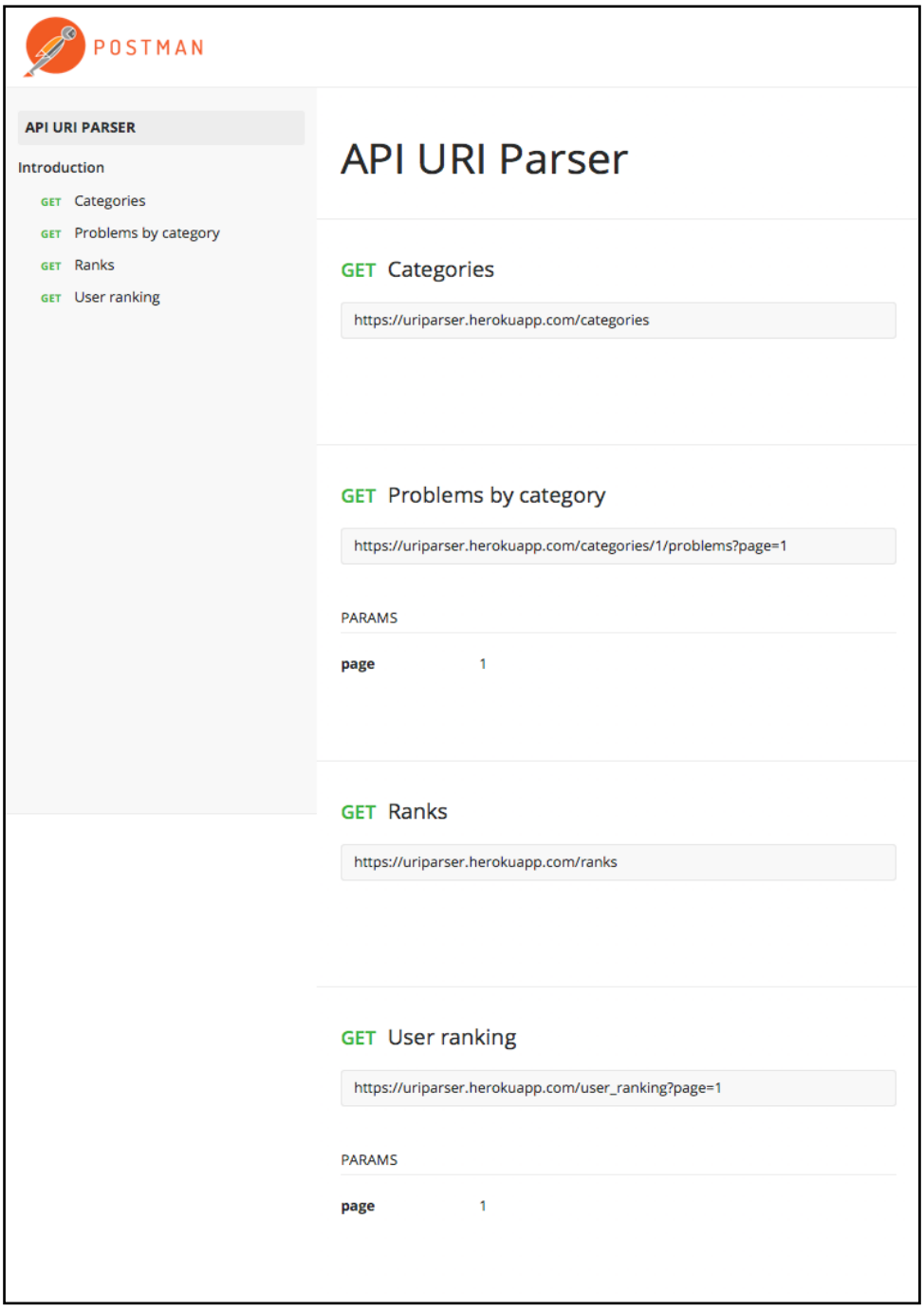


Figura 32 – *Endpoints* API intermediária parte 1.

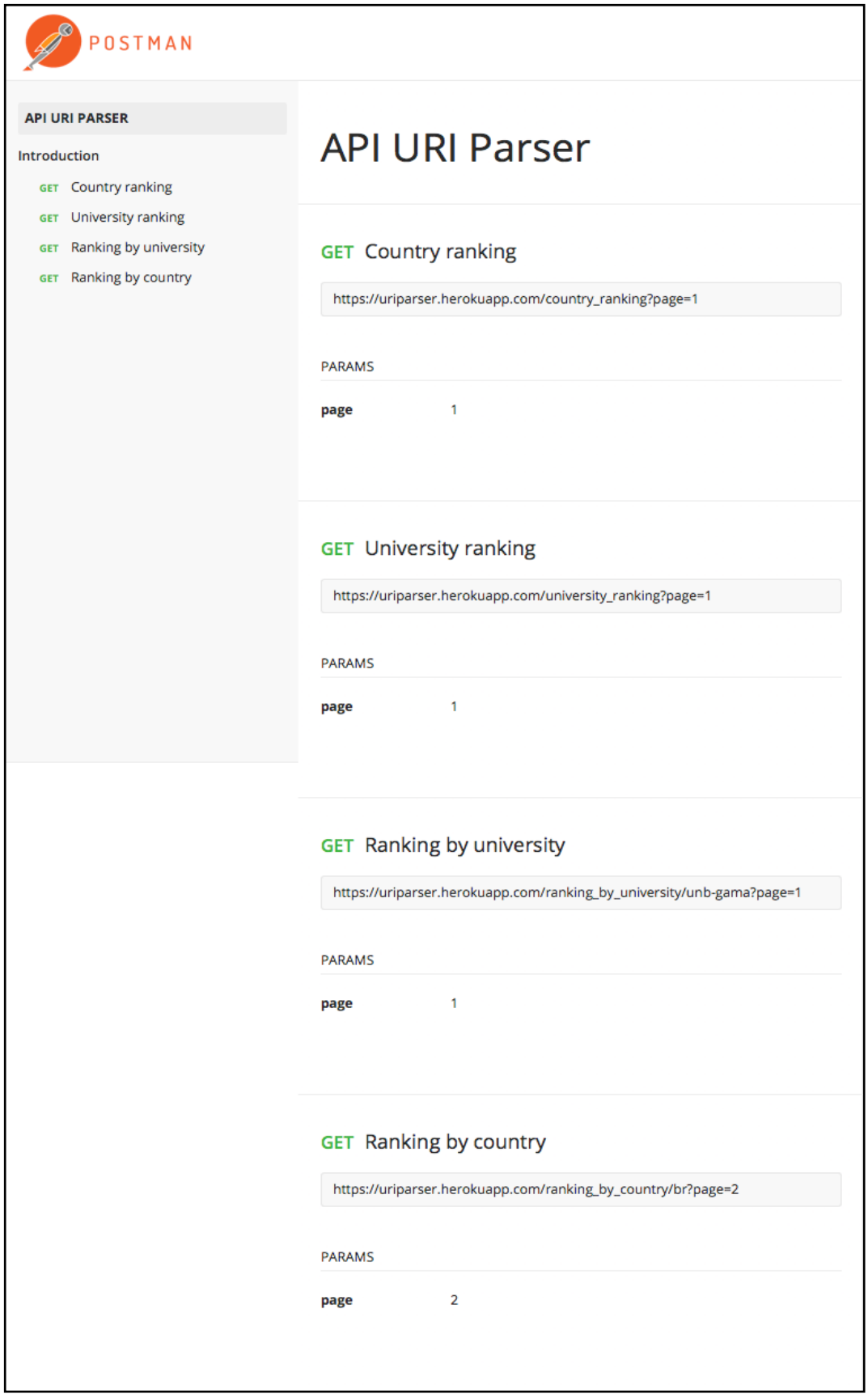


Figura 33 - *Endpoints* API intermediária parte 2.

APÊNDICE B – CÓDIGO-FONTE DA API INTERMEDIÁRIA

As Figuras de 34 até 36 mostram o código-fonte do módulo *Parser* utilizado para realizar o *Web Scraping* das páginas do *site* do URI *Online Judge*. Cada método presente no código-fonte é referente a um *endpoint* suportado pela API e documentado no apêndice A.

```ruby
require 'open-uri'

module Parser
  def self.get_categories(language="pt")
    categories = []

    page = Nokogiri::HTML(open(URL::create(:categories, language)))

    categories_list = page.css('div#category-list li')
    categories_list.each do |html_category|
      number = html_category.css('span').text
      title = html_category.css('div a').text
      description = html_category.css('div p').text
      problems_count = html_category.css('div b').text

      categories << Category.new(number, title, description, problems_count)
    end

    categories
  rescue
    categories
  end

  def self.get_problems(index, parameters, language="pt")
    problems = []

    url = "#{URL::create(:problems, language)}#{index}"
    url = URL::add_parameters(url, parameters)

    page = Nokogiri::HTML(open(url))
    problems_list = page.css('tbody tr')
    problems_list.each do |html_problem|
      begin
        number = html_problem.css('td a')[0].text
        name = html_problem.css('td a')[1].text
        solved = html_problem.css('td')[4].text.strip
        level = html_problem.css('td')[5].text.strip
      rescue
        next
      end

      problem = Problem.new(number, name, solved, level)
      problems << problem
    end
    problems
  rescue
    problems
  end

  def self.get_ranks(language="pt")
    ranks = []

    page = Nokogiri::HTML(open(URL::create(:ranks, language)))
    ranks_list = page.css('ul[class=v-menu] li')
    ranks_list.each do |html_rank|
      identifier = html_rank.css('a')[0]['href'].split("/")[3]
      title = html_rank.css('a').text
      description = html_rank.css('p').text

      rank = Rank.new(identifier, title, description)
      ranks << rank
    end
    ranks
  rescue
    ranks
  end

  def self.get_user_ranking(page=1, language="pt")
    users_ranking = []
```

Figura 34 – Código-fonte API intermediária parte 1.

```
70
71    url = "#{URL::create(:ranks, language)}?page=#{page}"
72    html = Nokogiri::HTML(open(url))
73    users_list = html.css('tbody tr')
74    users_list.each do |user|
75      begin
76        position = user.css('td')[0].text
77        name = user.css('td a')[0].text
78        university = user.css('td a')[1].text
79        solved = user.css('td')[4].text
80        tried = user.css('td')[5].text
81        submissions = user.css('td')[6].text
82      rescue
83        next
84      end
85
86      users_ranking << User.new(position, name, university, solved, tried, submissions)
87    end
88    users_ranking
89  rescue
90    users_ranking
91  end
92
93  def self.get_university_ranking(page=1, language="pt")
94    universities_ranking = []
95
96    url = "#{URL::create(:university_ranking, language)}?page=#{page}"
97    html = Nokogiri::HTML(open(url))
98
99    universities_list = html.css('tbody tr')
100   universities_list.each do |html_university|
101     begin
102       position = html_university.css('td')[0].text
103       acronym = html_university.css('td a')[0].text
104       institution = html_university.css('td a')[1].text
105       solved = html_university.css('td')[4].text
106       students = html_university.css('td')[5].text
107     rescue
108       next
109     end
110
111     universities_ranking << University.new(position, acronym, institution, solved, stude
112   end
113   universities_ranking
114 # rescue
115 #   universities_ranking
116 end
117
118 def self.get_user_ranking_by_university(acronym, page=1, language="pt")
119   users_ranking = []
120
121   url = "#{URL::create(:user_ranking_by_university, language)}#{acronym}?page=#{page}"
122   html = Nokogiri::HTML(open(url))
123
124   users_list = html.css('tbody tr')
125   users_list.each do |html_user|
126     begin
127       position = html_user.css('td')[0].text
128       name = html_user.css('td a')[0].text
129       university = acronym
130       solved = html_user.css('td')[3].text
131       tried = html_user.css('td')[4].text
132       submissions = html_user.css('td')[5].text
133     rescue
134       next
135     end
136
137     users_ranking << User.new(position, name, university, solved, tried, submissions)
138   end
139   users_ranking
```

Figura 35 – Código-fonte API intermediária parte 2.

```ruby
  rescue
    users_ranking
  end

  def self.get_user_ranking_by_country(initials, page=1, language="pt")
    users_ranking = []

    url = "#{URL::create(:user_ranking_by_country, language)}#{initials}?page=#{page}"
    html = Nokogiri::HTML(open(url))

    users_list = html.css('tbody tr')
    users_list.each do |html_user|
      begin
        position = html_user.css('td')[0].text
        name = html_user.css('td a')[0].text
        solved = html_user.css('td')[3].text
        tried = html_user.css('td')[4].text
        submissions = html_user.css('td')[5].text
      rescue
        next
      end

      users_ranking << User.new(position, name, solved, tried, submissions)
    end
    users_ranking
  rescue
    users_ranking
  end

  def self.get_country_ranking(page=1, language="pt")
    countries_ranking = []

    url = "#{URL::create(:country_ranking, language)}?page=#{page}"
    html = Nokogiri::HTML(open(url))

    countries_list = html.css('tbody tr')
    countries_list.each do |html_country|
      begin
        position = html_country.css('td')[0].text
        initials = html_country.css('td a')[0].text
        name = html_country.css('td a')[1].text
        solved = html_country.css('td')[4].text
        students = html_country.css('td')[5].text
      rescue
        next
      end

      countries_ranking << Country.new(position, initials, name, solved, students)
    end
    countries_ranking
  rescue
    countries_ranking
  end
end
```

Figura 36 – Código-fonte API intermediária parte 3.

A Figura 37 mostra o código-fonte do módulo ‘URL’ para montagem das URLs que retornam o html das páginas a serem analisadas pelo módulo ‘*Parser*’.

```ruby
module URL
  URIONLINEJUDGE = "https://www.urionlinejudge.com.br/judge/"

  URLS = {
    categories: "/categories",
    problems: "/problems/index/",
    ranks: "/rank",
    university_ranking: "/universities",
    country_ranking: "/countries",
    user_ranking_by_university: "/users/university/",
    user_ranking_by_country: "/users/country/"
  }

  def self.create(identifier, language="pt")
    "#{URIONLINEJUDGE}#{language}#{URLS[identifier]}"
  end

  def self.add_parameters(path, parameters)
    return path if parameters.empty?

    separator = "?"
    parameters.each do |key, value|
      next if value.nil?
      path << "#{separator}#{key}=#{value}"
      separator = "&"
    end
    path
  end
end
```

Figura 37 – Código-fonte API intermediária parte 4.