**UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO**

**FACULDADE DE CIÊNCIAS**

SISTEMAS DE INFORMAÇÃO

Vittoria Zago

**NIX – FERRAMENTA PARA TESTES DE INTEGRAÇÃO E ACEITAÇÃO**

Trabalho apresentado como exigência parcial

para a conclusão do Curso de Bacharelado

em Sistemas de Informação da Faculdade de

Ciências – UNESP Campus Bauru.

Orientador: Professor Doutor Kelton Augusto Pontara da Costa

Bauru, 2018

**RESUMO**

O uso de APIs (Application Programming Interface) está se difundindo cada dia mais no mercado de desenvolvimento de software do mundo todo. Porém, sua validação e atualização ainda é um desafio para a equipe de desenvolvimento. As ferramentas atuais exigem um trabalho manual e constante, além de um conhecimento técnico muito aprofundado. O teste de uma API, seja de requisitos funcionais, fluxo de navegabilidade ou desempenho, seria realizado da forma ideal se fosse realizado por uma pessoa com conhecimentos profundos no negócio, sem apego ou vícios de teste. Essas dificuldades foram as motivações para este trabalho. Este trabalho apresenta a proposta e implementação do sistema Nix: um sistema que facilite os testes de uma API, que consiga persistir e atualizar de forma automática, liberando assim a equipe de desenvolvimento e aumentando a qualidade dos testes.

**Palavras-chave:** Testes de integração. Testes de APIs.

**ABSTRACT**

The well-known Application Programming Interface (APIs) are dominating the development market all over the world. However, its validation and upgrade testing is still a challenge for the development team. Today's tools require constant manual work and in-depth technical knowledge. The test of an api, whether functional requirements, navigational flow or performance, would be performed ideally if performed by a person with in-depth business knowledge, without attachment or test vices. These difficulties were the motivations for this work. This work presents the proposal and implementation of the Nix system: a system that facilitates the testing of an api, which can persist and update automatically, thus freeing the development team and increasing the quality of the tests.

**Keywords**: Integration tests. Testing APIs.

**Lista de Abreviaturas e Siglas**

|  |  |
| --- | --- |
| API | Interface de Programação de Aplicativos, do inglês *Application Programming Interface* |
| XML | Linguagem de marcação extensível, do inglês Extensable Markup Language |
| REST | Transferência de Estado Representacional, do inglês Representational State Transfer |
| HTML | Linguagem de Marcação de Hipertexto, do inglês HyperText Markup Language |
| HTTP | Protocolo de Transferência de Hipertexto, do inglês Hypertext Transfer Protocol |
| CSV | Valores separados por ponto e vírgula, do inglês Comma-separated values |
|  |  |
|  |  |
|  |  |

**LISTA DE ILUSTRAÇÕES**

[Figura 1: Exemplo de API - Twitter Developers 7](#_Toc517696798)

[Figura 2: Uso de APIs em diferentes áreas 8](#_Toc517696799)

[Figura 3: A pirâmide de testes 9](#_Toc517696800)

[Figura 4: Exemplo da implementação do Rest-Assured 12](#_Toc517696801)

[Figura 5: Interface do JMeter de cadastro de um teste 13](#_Toc517696802)

[Figura 6: Tela de cadastro e envio de solicitação no Postman 14](#_Toc517696803)

[Figura 7: Diagrama de uso 20](#_Toc517696804)

[Figura 8: Diagrama de usuário 21](#_Toc517696805)

[Figura 9: Diagrama de entidades 22](#_Toc517696806)

[Figura 10: Json gerado pelo swagger 24](#_Toc517696807)

[Figura 11: Interface do Swagger para teste 25](#_Toc517696808)

[Figura 12: Interface RoboMongo 27](#_Toc517696809)

[Figura 13: comparação do uso do React em relação a outras tecnologias em 2017 28](#_Toc517696810)

[Figura 14: Tela inicial da importação do cadastro 29](#_Toc517696811)

[Figura 15: Tela inicial com API selecionada 30](#_Toc517696812)

[Figura 16: Exportar arquivo 31](#_Toc517696813)

[Figura 17: Cadastrar requisição 32](#_Toc517696814)

[Figura 18: Cronograma do projeto 35](#_Toc517696815)

**Sumário**

[1. Introdução 7](#_Toc525040940)

[2. Detalhamento do Problema 9](#_Toc525040941)

[3. Soluções Existentes e Objetivos da Proposta 12](#_Toc525040942)

[3.1. RestAssured 12](#_Toc525040943)

[3.2. JMeter 13](#_Toc525040944)

[3.3. Postman 13](#_Toc525040945)

[3.4. Comparativo 14](#_Toc525040946)

[4. Descrição do Projeto Proposto 16](#_Toc525040947)

[4.1. Armazenamento 16](#_Toc525040948)

[4.2. Importação e Exportação 17](#_Toc525040949)

[4.3. Cadastro e atualização manual 18](#_Toc525040950)

[4.4. Variáveis globais e locais 18](#_Toc525040951)

[4.5. Interface do usuário 18](#_Toc525040952)

[4.6. Execução dos testes 18](#_Toc525040953)

[4.7. Diagramas 19](#_Toc525040954)

[5. Tecnologias Pesquisadas e Escolhidas 23](#_Toc525040955)

[5.1. Swagger 23](#_Toc525040956)

[5.2. Java 25](#_Toc525040957)

[5.2.1. Spring 26](#_Toc525040958)

[5.3. MongoDB 26](#_Toc525040959)

[5.3.1 RoboMongo 27](#_Toc525040960)

[5.4. React 27](#_Toc525040961)

[6. Implementação 29](#_Toc525040962)

[7. Procedimento de Teste e Validação do Projeto 33](#_Toc525040963)

[8. Análise dos Riscos 34](#_Toc525040964)

[9. Cronograma do Projeto 35](#_Toc525040965)

[10. Conclusão 36](#_Toc525040966)

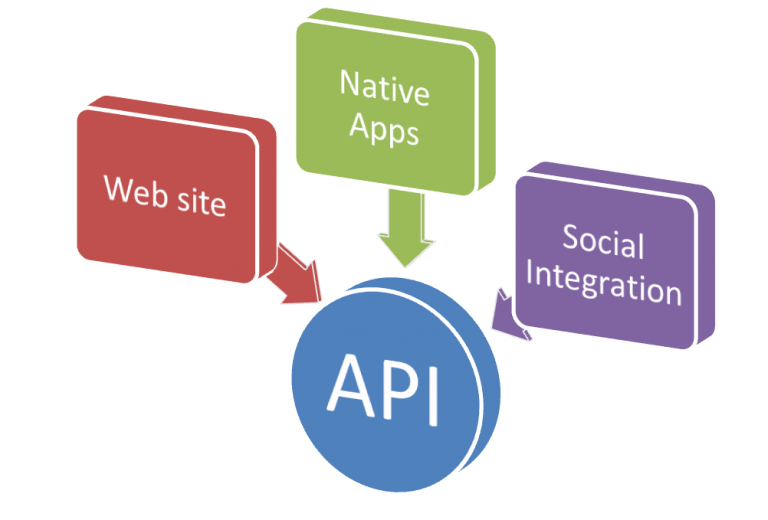
[11. Referências Bibliográficas 37](#_Toc525040967)

# 1. Introdução

O mercado de desenvolvimento de software está focado cada vez mais em desenvolver projetos desacoplados, reutilizáveis e independente de alterações externas, por isso vem crescendo o desenvolvimento de APIs (Application Programming Interface). Como o próprio significado já diz, a API serve para padronizar a interface de comunicação entre dois ou mais sistemas, logo seu uso pode ser feito por diversos sistemas e dispositivos – Figura 1.

A estrutura de uma API pode ser definida como um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação, para que outras aplicações consigam utilizar as funcionalidades desta aplicação, sem precisar conhecer detalhes da implementação do software [1]. Deste sistema também é esperado uma alta velocidade no tempo de resposta.

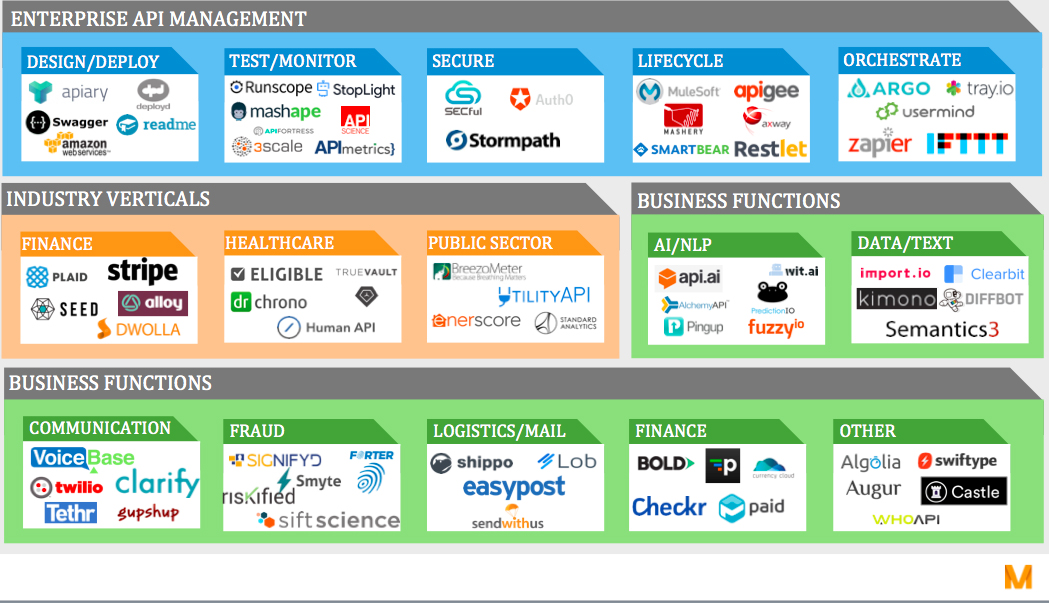
Figura 1: Exemplo de API - Twitter Developers



Fonte: DevMedia

A criação de APIs deixou de ser uma criação para melhoria e reuso interno, para se tornar um produto especializado desenvolvido por uma empresa focada no negócio, que consiga validar da forma certa e evoluir com maior desempenho [2]. Podemos ver na Figura 2 a relação do uso de APIs em diferentes áreas.

Figura 2: Uso de APIs em diferentes áreas



Fonte: TechCrunch

Sabendo que a utilização e implementação dessas APIS estão cada vez maior, cada vez mais tem se procurado formas de validar os métodos disponíveis, o fluxo e as regras de negócio, o desempenho e o nível de segurança da API. Um dos testes mais importante para realizar essas validações, é conhecido como teste de integração.

O teste de integração permite analisar e encontrar erros de interface e erros entre as dependências dos componentes do software. O teste irá verificar o desempenho, requisitos funcionais e a confiabilidade do sistema. Ele é aplicado geralmente depois do teste de unidade, onde cada componente unitário do sistema foi testado de modo isolado. Embora essas unidades possam funcionar corretamente, quando juntas podem ocorrer erros por problemas de compatibilidade ou algum outro motivo [3].

O sistema proposto neste trabalho foi nomeado como Nix, inspirado na Figura representada pela mitologia nórdica, deusa da noite. Ele tem como finalidade o teste integrado da API de forma facilitadora para pessoas sem profundidade técnica em desenvolvimento, de forma a atualizar e armazenar esses testes de forma automática.

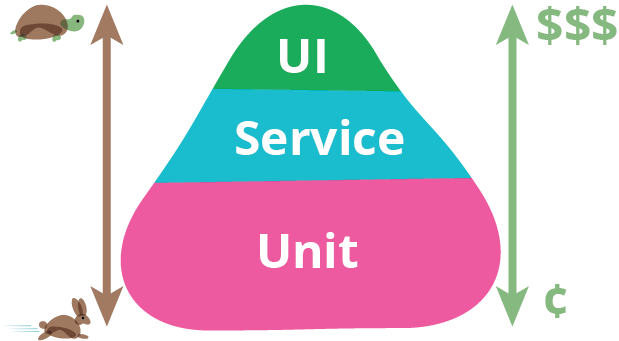
# 2. Detalhamento do Problema

Desenvolver um sistema e mantê-lo em produção é sempre um desafio, pois muitas vezes envolve constantemente evoluir suas funcionalidades para que o sistema permaneça útil para seus usuários o máximo de tempo possível. Devido a esta realidade, é muito fácil ver a importância de se manter um conjunto de testes automatizados em nosso sistema. Por razões como estas, equipes de software muitas vezes possuem indivíduos com perfis bem distintos, esperando que cada perfil contribua de forma diferente para a construção deste software [4]. Normalmente o conhecimento necessário para o desenvolvimento de um software está difundido entre usuários, clientes, analistas de negócios e outros envolvidos que não possuem domínio de programação.

Cada tipo de sistema possuí diferentes possíveis formas e ferramentas para serem testados. Com o crescente uso das APIs no mercado de desenvolvimento de software, vem crescendo também a procura por métodos com melhor resultado na qualidade, e ferramentas que implementem esses métodos.

Na Figura 3, Martin Fowler indica que o custo que o teste pode causar é inversamente proporcional à rapidez de seu desenvolvimento [5].

Figura 3: A pirâmide de testes



Fonte: Martin Fowler

Os testes com a interface do usuário possuem uma complexidade e fragilidade muito maior, podendo quebra-los com pequenas alterações em diversos pontos do sistema. O autor sugere focar nas camadas de teste abaixo em maior quantidade e qualidade, tendo assim maiores chances de sucesso na camada de interface. Os testes de alto nível são uma segunda linha de defesa de testes. Se você obtiver uma falha em um teste de alto nível, não apenas terá um bug em seu código funcional, mas também terá um teste de unidade ausente ou incorreto.

Para entender a finalidade do nosso sistema, é importante entender os diferentes tipos de testes de softwares [6]:

* **Testes de unidade:** têm como objetivo testar a menor unidade de código possível, apenas um método por vez, que deverá representar uma única função. Geralmente não incluem teste com banco de dados, apenas a regra de negócio.
* **Testes automatizados:** estes têm como objetivo testar a interface de interação com o usuário. Há várias ferramentas que conseguem simular o preenchimento de campos, clicks nos botões e toda navegabilidade de uma página.
* **Testes de integração:** testa o comportamento de um conjunto de unidades de código em nosso software trabalhando em conjunto e verifica o resultado com o esperado. O método testado geralmente é um método principal que chama vários métodos, estes já podem ter sido testados unitariamente.
* **Testes de aceitação:** esses testes comparam o retorno dos métodos com o esperado pelo cliente, já especificado na regra de negócio. Pode validar: funcionalidades quanto outros itens não funcionais, como desempenho, acessibilidade, ergonomia, etc.

Os últimos dois testes estão na camada de serviço descrito na pirâmide de Martin Fowler.

Para os testes em API, são utilizadas ferramentas que consigam averiguar os testes de integração e os testes de aceitação. Como já descrito, estes testes validam em sua maioria itens funcionais, os quais necessitam de profissionais com maior conhecimento do negócio para testes mais apurados.

Para escolher a fermenta ideal para teste, deve-se levar em consideração o usuário que criará o caso de teste. A maioria das ferramentas exige que esse usuário seja alguém com conhecimento técnico mais aprofundado, um desenvolvedor de software ou um especialista em testes.

Portanto, o usuário-alvo desses sistemas pode não ser o ideal. Assim como pode ser muito difícil para o especialista do negócio aplicar o teste, também há dificuldade para o programador entender o negócio. Muitos dos problemas que podem acontecer, não cabe ao desenvolvedor saber se está certo ou não. E mesmo os que cabem, nós desenvolvedores somos tendenciosos a desenvolver testes viciados, limitados, por não sermos especialistas no fluxo de negócio.

Por isso, para testes mais precisos, o adequado seria as pessoas que idealizaram o software testar os casos. Porém atualmente, não foi encontrada uma ferramenta de teste para API, que não dependa de muito conhecimento técnico ou ainda que contenha cadastros e atualizações automáticas dos métodos disponíveis da API, exigindo muito trabalho manual e tempo disponível.

# 3. Soluções Existentes e Objetivos da Proposta

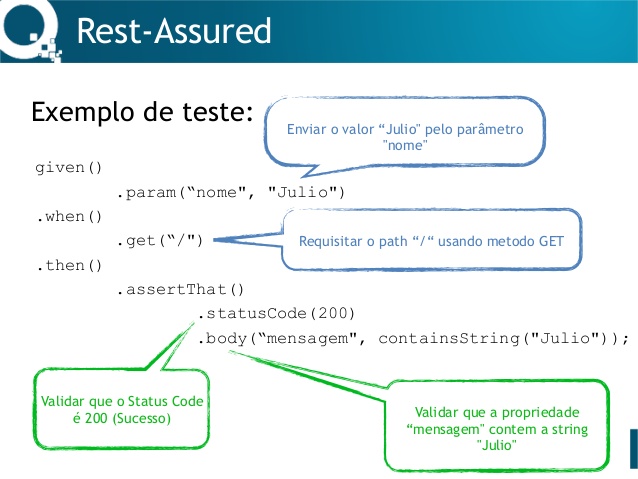
Neste tópico será listada as soluções já existentes para suprir o problema apresentado. Apresentando as principais ferramentas e suas principais características, e logo em seguida o comparativo das mesmas.

## 3.1. RestAssured

Rest-Assured é uma biblioteca desenvolvida em Java, mas que pode ser integrada com qualquer estrutura de automação baseada em Java existente. Tem como finalidade testar serviços REST baseados em HTTP [7]. Os testes têm que ser programados no fonte, e apresenta várias funcionalidades embutidas porém exige um domínio técnico e da linguagem absurdo, como a Figura 4 demonstra.

Como teste é desenvolvido diretamente na fonte, é possível realizar atribuições de variáveis estáticas para serem reutilizadas em testes futuros, e fazer um teste de fluxo completo sem fingir dados.

Figura 4: Exemplo da implementação do Rest-Assured



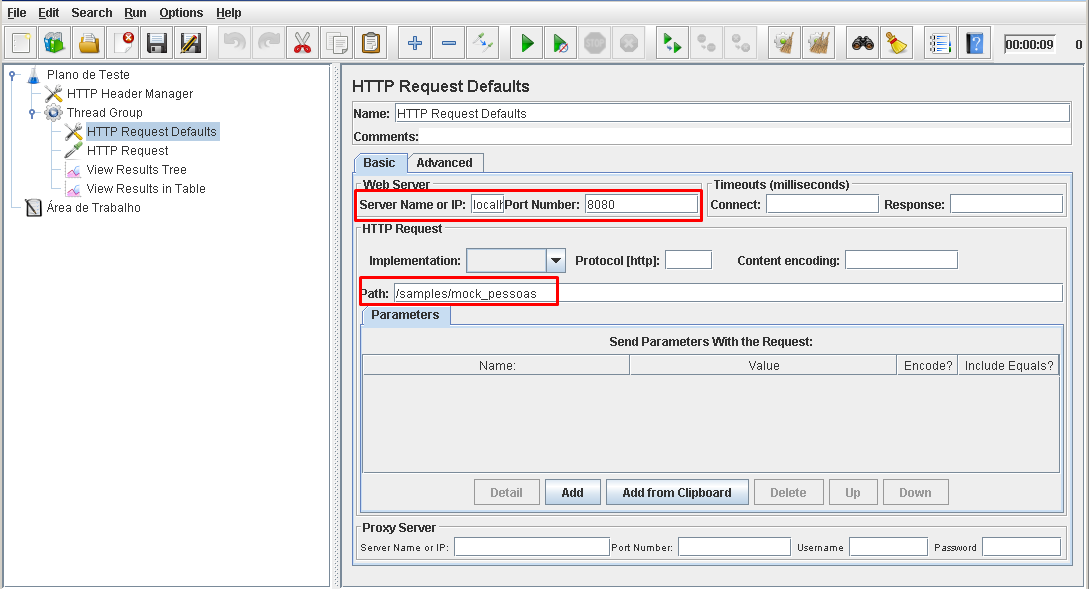
Fonte: TDC

## 3.2. JMeter

O JMeter foi criado principalmente para testes de carga – chamadas em lote para testar desempenho da API. Mas também vêm sido muito utilizado para testes funcionais. Podemos classificar esta ferramenta como a mais completa. O JMeter inclui todas as funcionalidades que você precisa para testar uma API, além de alguns recursos extras. É possível codificar para gerar números, usar variáveis globais, importar arquivos CSV com padrões dados para testes em carga diferentes [8]. Os cadastros dos testes são manuais, e apesar de não ser desenvolvido direto no próprio código fonte, ainda exige uma capacidade técnica muito aprofundada, além de ter uma interface pouco amigável – Figura 5. Todo o teste cadastrado será cadastrado em formato xml.

O JMeter permite o cadastro de variáveis globais e locais, sendo que as locais podem ser atribuídas e reutilizadas em um mesmo conjunto de testes, mas não entre testes distintos.

Figura 5: Interface do JMeter de cadastro de um teste



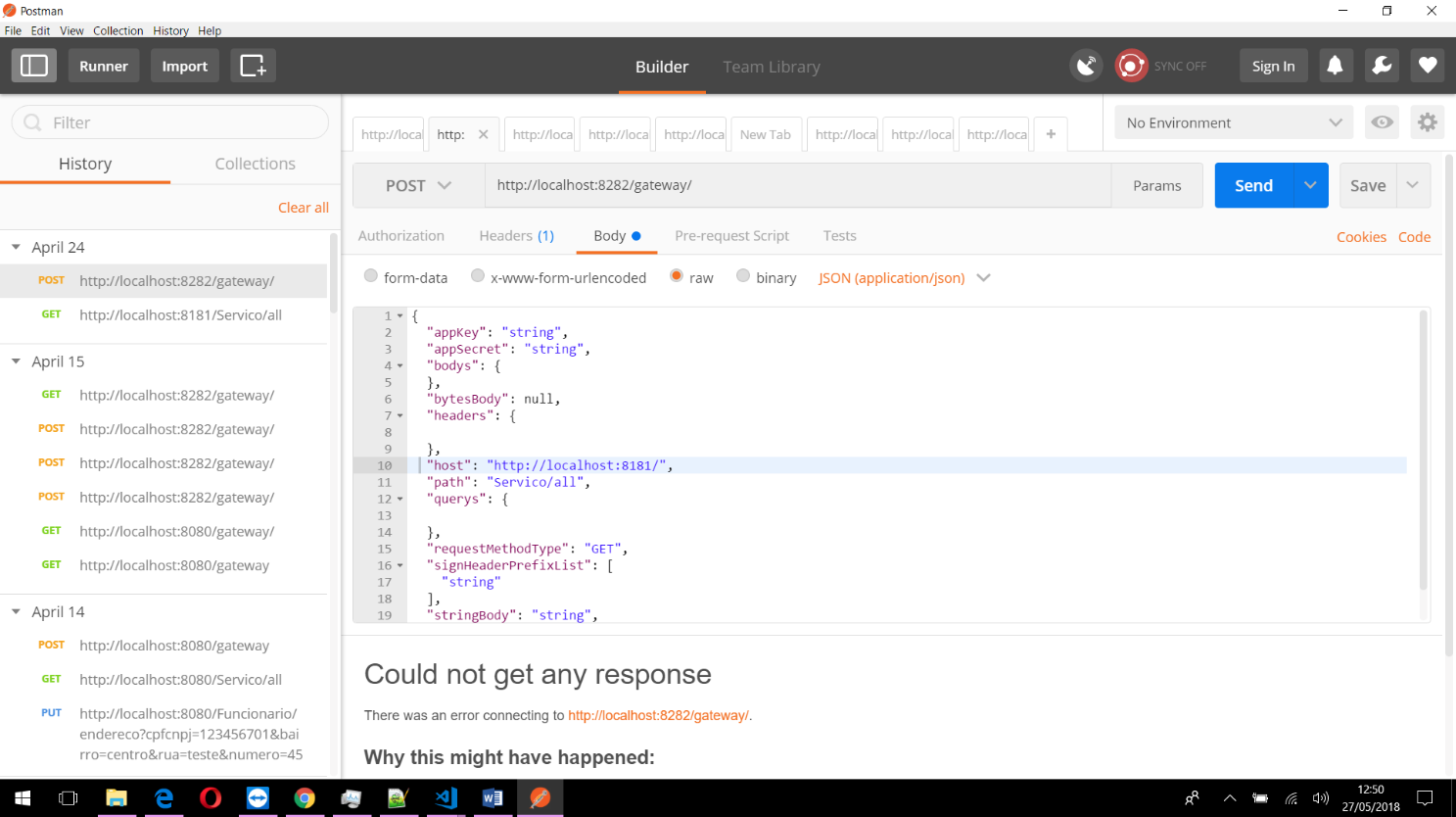
Fonte: MATERA

## 3.3. Postman

Postman é uma das ferramentas com a interface mais amigável ao usuário. Não é preciso codificar, mas isso também ocasiona em algumas limitações. Ainda exige um cadastro manual ou importar um arquivo em formato Json para conseguir testar uma API. O Postman também oferece diversas soluções recentes para teste em carga e monitoramento da API, porém só estão disponíveis em um plano pago. Os arquivos criados no Postman podem ser salvos em Json, além do mais a ferramenta possui um histórico de chamadas que você pode recuperar algo que você não salvou. Pode ser usado em um plugin no navegador chrome ou como uma versão nativa para Mac e Windows [9].

Assim como o JMeter, o Postman possui variáveis globais configuráveis, porém são constantes, forçando a alteração manual do dado para teste.

Figura 6: Tela de cadastro e envio de solicitação no Postman



Fonte: Próprio autor.

## 3.4. Comparativo

Para facilitar a análise de comparação entre as soluções encontradas no mercado, a Tabela 1 resume as principais características de cada solução.

Tabela 1: Comparativo das soluções existentes

|  |  |  |  |
| --- | --- | --- | --- |
|  | Rest-Assured | JMeter | Postman |
| Multiplataforma |  |  |  |
| Cadastro automático |  |  |  |
| Atualização automática |  |  |  |
| Armazenamento Local |  |  |  |
| Armazenamento Remoto |  |  |  |
| Interface amigável |  |  |  |
| Variáveis Globais Atribuiveis |  |  |  |

Fonte: Próprio autor

# 4. Descrição do Projeto Proposto

O sistema proposto nomeado Nix, tem como objetivo facilitar o trabalho de teste para a equipe de desenvolvimento e direcionar a usabilidade para usuários com foco no negócio, aumentando também a qualidade dos testes.

* Importar automaticamente e cadastrar os testes de todos os métodos das APIs a partir de uma documentação;
* Importar todos os métodos alterados, criados, ou já existentes de uma API, sem perder as configurações anterior;
* Gerar algum dado de acordo com o tipo do campo para cada campo e permitir alteração desse dado - uma vez que não é possível adivinhar sua validação dentro da API;
* Salvar o Json enviado e alterado num banco não relacional e permitir o usuário importar ou exportar, além de permitir estatísticas de forma filtrada pelo banco;
* Permitir salvar variáveis globais e locais que podem ser alteradas/atribuídas de acordo com o resultado; e
* Executar os testes e apresentar comparação do esperado e o resultado recebido.

## 4.1. Armazenamento

O repositório de armazenamento será configurado no arquivo de configurações do projeto. Armazenando as informações de requisições, é possível:

* Maior segurança de que os testes serão salvos;
* Alteração em lote em diversos testes de diversas APIs que podem se integrar;
* Geração de dados através de funções aplicadas no banco;
* Pesquisa, filtros, views, e estatísticas sobre os resultados obtidos; e
* Escalabilidade.

Resumindo: armazenando as informações dos testes, é possível realizar diversas funções que o sistema proposto ainda não oferecerá.

O armazenamento escolhido é em banco não relacional, devido a sua flexibilidade [11]. Cada cadastro de uma API e seus testes terá seu próprio modelo. Além disso, o uso de um banco não relacional permite um maior desempenho no sistema, visto que neste sistema é esperado um curto tempo de resposta.

As possibilidades listadas acimas, também serão muito bem realizadas com um banco não relacional. Com o peso do open source nos bancos não relacionais, a maioria desses bancos oferecem várias funcionalidades especiais, como: geração de scripts, índices específicos, replicação automática de dados, entre outros.

## 4.2. Importação e Exportação

A importação já foi explicada como um dos maiores diferenciais nesse sistema, poupando muito tempo de trabalho manual e repetitivo.

Já a exportação é proposta devido aos fatores positivos, também analisado nas ferramentas já existentes, que seguem:

* Possibilidade de compartilhamento entre a equipe mais rapidamente;
* Possibilidade de versionamento dos testes junto ao código fonte; e
* Backup.

Para a importação de cadastro e atualização de testes, é preciso escolher um formato. O formato escolhido é o Json, uma vez que é importável e exportável de maneira mais simples de um banco não relacional. Também vimos o formato de Json sendo utilizado pela ferramenta analisada com mais pontos positivos, o Postman. O formato Json também é ótimo para compreensão dos usuários que não tem aprofundamento técnico, é conhecido por ser um formato mais verboso.

Para fazer esse processo de importação automático, sem a intervenção do usuário, é preciso escolher uma ferramenta que gere esse Json, com todos os métodos e suas particularidades. A ferramenta escolhida para documentação automática da API, é o Swagger, que será melhor abordado no tópico 5.1. O importante levantarmos agora, é que para chamarmos um método precisamos de uma chave única, em qualquer caso. No caso de uma API não é diferente, cada rota deve ser única e a documentação do Swagger já identifica essa rota para nós. Essa rota única será uma chave, tratada pelo sistema, auxiliando na importação para atualização. Se essa rota já existia no cadastro anterior, será analisado seus dados quanto há alguma mudança, permitindo assim a importação para atualização. Se a rota não existia, seu método será criado. Caso a rota deixe de existir, o método será desativado.

## 4.3. Cadastro e atualização manual

A importação de dados não será a única forma de entrada. Assim como nas outras ferramentas, o cadastro e alteração dos testes e seus valores poderá ser feito manualmente, facilitando alguma alteração necessária durante o fluxo.

## 4.4. Variáveis globais e locais

O uso de variáveis globais facilitará muito os testes, nessas variáveis terão informações que poderão ser reutilizadas em todos os métodos, e estas também serão persistidas. Por exemplo:

* Rota de publicação para o teste e sua porta;
* Dados de autenticação;
* Resultados de métodos que podem ser atribuídos em variáveis globais e utilizado em outros métodos; e
* Dados necessários para diversos testes, que não tem geração em métodos anteriores.

## 4.5. Interface do usuário

A interface para uso do usuário será um dos diferenciais nesse sistema. Para atingir os diversos perfis de público alvo, é preciso uma interface bem acessível, de fácil navegação para os usuários, fácil compreensão e alto desempenho.

Pensando que os usuários já estão familiarizados com as ferramentas apresentadas, a interface não deve fugir muito desse padrão, para maior adaptabilidade. O padrão a ser usado para maior inspiração é o Postman e a interface gráfica do Swagger- que será melhor detalhada mais a frente-. Ambas tecnologias conseguem de forma dinâmica gerar os campos necessários para execução dos testes, sem muito detalhamento ou aprofundamento na especificação técnica - como o JMeter - permitindo que usuários mais leigos consumam de forma simples e rápida.

## 4.6. Execução dos testes

A execução dos testes depende de encaminhar as chamadas cadastradas para a API de destino. Este tipo de execução, será realizada por uma estrutura intitulada gateway – em português: porta de acesso-. O gateway vai receber a uma rota, seus dados necessários de entrada, previamente cadastrado no teste, e vai executar o método vinculado. Ao executar o método o gateway encaminhará o resultado para quem o chamou, no nosso caso será o usuário quando solicitar a execução através do sistema Nix.

No HTTP Rest, temos alguns padrões esperado como retorno, o HttpStatusCode ou código de status http, é um padrão universal de códigos para retorno que representam o sucesso ou falha da requisição. Isso é importante, pois o sistema permitirá vários testes com resultados diferentes para o mesmo método, e os dados de entrada para cada cenário também serão persistidos. O resultado recebido será comparado com o resultado esperado e ambos serão apresentados para o usuário.

A ferramenta Swagger, também consegue extrair em seu extenso Json, os possíveis retornos e seus objetos esperados, desde que esses estejam na documentação nativa da linguagem. A partir destes resultados esperados, o sistema irá sugerir para o usuário os resultados esperados de acordo com a documentação. Por padrão, assim que um teste for cadastrado, será preenchido valores aleatórios de acordo com a tipagem dos dados de entrada, apenas para uma situação: a de sucesso. O sistema não será responsável por gerar dados que acerte em todas as situações, uma vez que o dado é enviado para API, esta pode fazer qualquer tratamento com o mesmo, e considerar um dado válido ou não. Por exemplo: a entrada de uma string na API, pode ser opcional, mas dentro da API retornar uma exceção se a variável for nula ou com menos de N caracteres. Portanto, para a regra de negócio aplicada na API caberá ao usuário valida-la.

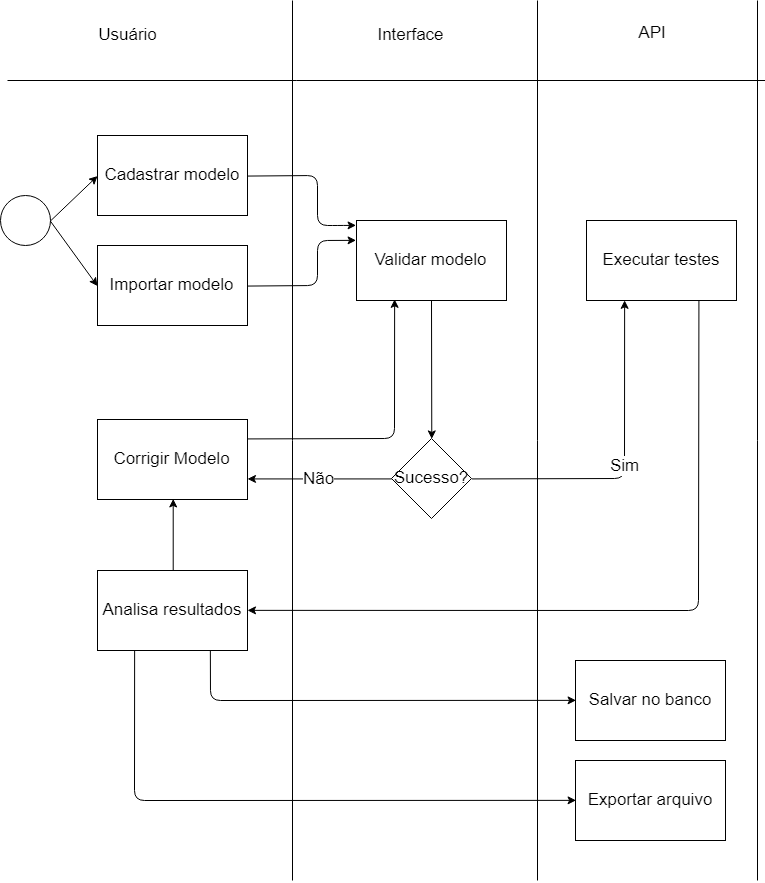
Os resultados dos testes não serão persistidos, pois este poderá variar em cada momento, e não é de interesse para a equipe. Se sua API tiver filtros para gravação de log, por exemplo, esta informação do resultado já deverá estar persistida no mesmo.

## 4.7. Diagramas

Nos diagramas desenvolvidos é possível uma melhor visualização das ações do usuário e suas possíveis sequências no sistema.

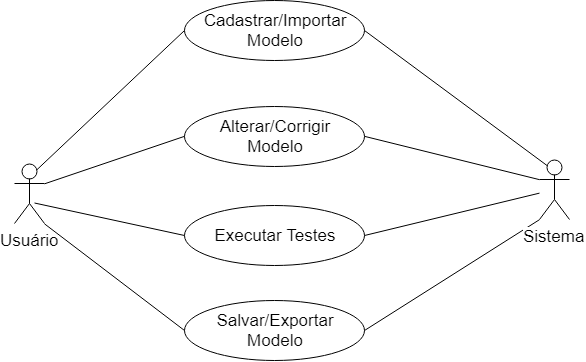
Na Figura 7 e na Figura 8 é possível visualizar as ações do usuário, da interface e da API. Dessa forma, podemos ver que o usuário pode cadastrar ou alterar um modelo já existente com requisições, a interface por sua vez irá validar o modelo e encaminhá-lo para a API. A API ficará responsável pela execução e armazenamento das requisições.

Figura 7: Diagrama de uso



Fonte: próprio autor

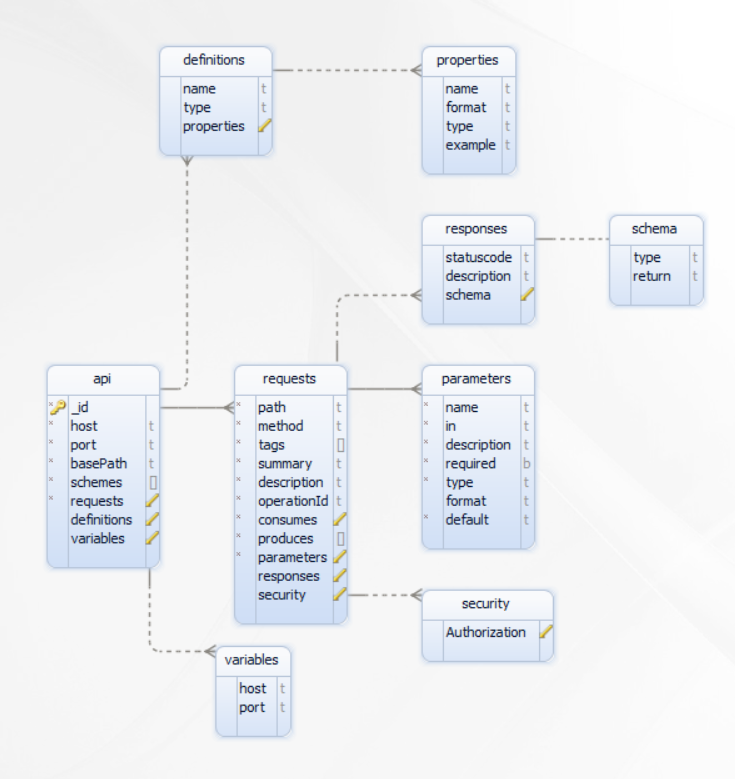
Figura 8: Diagrama de usuário



Fonte: próprio autor

Na Figura 9 é possível entender um pouco melhor a estrutura que o banco de dados se parecerá. Como foi decidido um banco não relacional, não temos tabelas bem definidas, mas os campos apresentados são os dados mínimos gravados, podendo ter variações de campos obrigatórios ou não. Então podemos ver a entidade API como um modelo, que contém uma coleção de requisições e seus parâmetros necessários para o correto funcionamento.

Figura 9: Diagrama de entidades



Fonte: própria autora

# 5. Tecnologias Pesquisadas e Escolhidas

Neste tópico será abordado as ferramentas escolhidas para o desenvolvimento do sistema, seus pontos positivos e negativos, e porque ela foi escolhida.

## 5.1. Swagger

Não é à toa que o Swagger é o framework mais utilizado para desenvolvimento de APIs. Este tem suporte em mais de 25 linguagens e mais de 10 milhões de downloads registrados, segundo seu próprio site. O Swagger, assim que implementado na API, gera documentação de todas as rotinas, entrada e saída, cabeçalhos, rotas, autenticação entre outros [12].

Na Figura 10, podemos ver o Json gerado pelo Swagger. Ele apresenta uma lista de métodos, que se distinguem com uma chave única: rota(path). A rota de cada método de uma API deve ser única, é o endereço de destino que o servidor irá se encaminhar. Cada um desses métodos, identificados por uma rota, pode conter vários parâmetros e respostas. O Swagger também consegue extrair da documentação do código fonte a descrição do método, seus objetos de retorno de acordo com o código correspondente ao HTTPCode, detalhando seus possíveis erros [13].

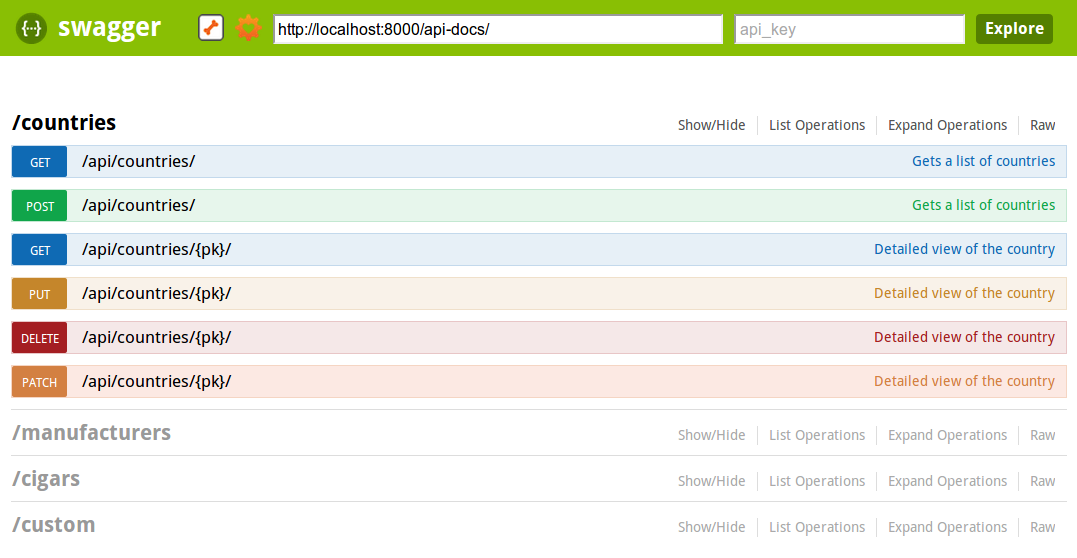
Figura 10: Json gerado pelo swagger



Fonte: Taoffi Nassar

O Swagger também possibilita o teste manual de métodos da API. Na Figura 11 é possível ver um exemplo com os métodos de uma API disponíveis para teste.

Figura 11: Interface do Swagger para teste



Fonte: tomchristie

A interface do Swagger é muito utilizada pelos programadores para testar em tempo real junto com o desenvolvimento da API. Porém não é possível salvar o teste, ou realizar testes de integração ou aceitação. A interface para teste do Swagger tem como principal funcionalidade a documentação, os testes realizados através dessa representam o mesmo principio que o gateway realizará: encaminhar chamadas.

## 5.2. Java

As linguagens de programação consideradas para desenvolver a parte do back-end, ou seja o gateway, são linguagens com mais ferramentas e facilidades de se integrar com o HTTP. Essas linguagens são C# e Java.

A linguagem escolhida para desenvolvimento do gateway e integração com o banco de dados, foi o Java. Não apenas devido ao conhecimento que já possuo sobre o Java, mas também por este dar um excelente suporte a todas outras ferramentas escolhidas. O java é uma das principais linguagens utilizadas para o back-end, inclusive para desenvolvimento de APIs, devido a sua rapidez e programação em baixo nível que permite diversas funcionalidades. Além disso, em comparação ao C#, o java pode ser considerada uma linguagem de mais baixo nível e também possui uma integração mais rápida com ORMs para banco de dados não relacionais.

A IDE selecionada para desenvolvimento foi o Eclipse, devido à familiaridade com a ferramenta.

### 5.2.1. Spring

O Spring é um framework opensource para Java, que se responsabiliza pela injeção de dependência e configurações automáticas, deixando o programador livre para desenvolver apenas o necessário.

O Spring tem várias segmentações que podem ser implementas que facilitam mais ainda o desenvolvimento. Uma dessas implementações é o Spring Boot que se responsabiliza pelo start, atribuição e finalização do servidor, deixando mais uma vez o desenvolvedor Java livre da responsabilidade do server.

## 5.3. MongoDB

Sabendo que foi escolhido um banco NOSQL – não relacional-, será preciso escolher a ferramenta de manuseio e armazenamento. Dentre os tipos de NoSql, o modelo escolhido foi o orientado a documento [14], pelos seguintes motivos:

* Melhor desempenho em consultas;
* Ideal para variedade de atributos com grandes conteúdos;
* Facilidade de uso; e
* Suporte de backend para sites com altos volumes de leituras e gravações.

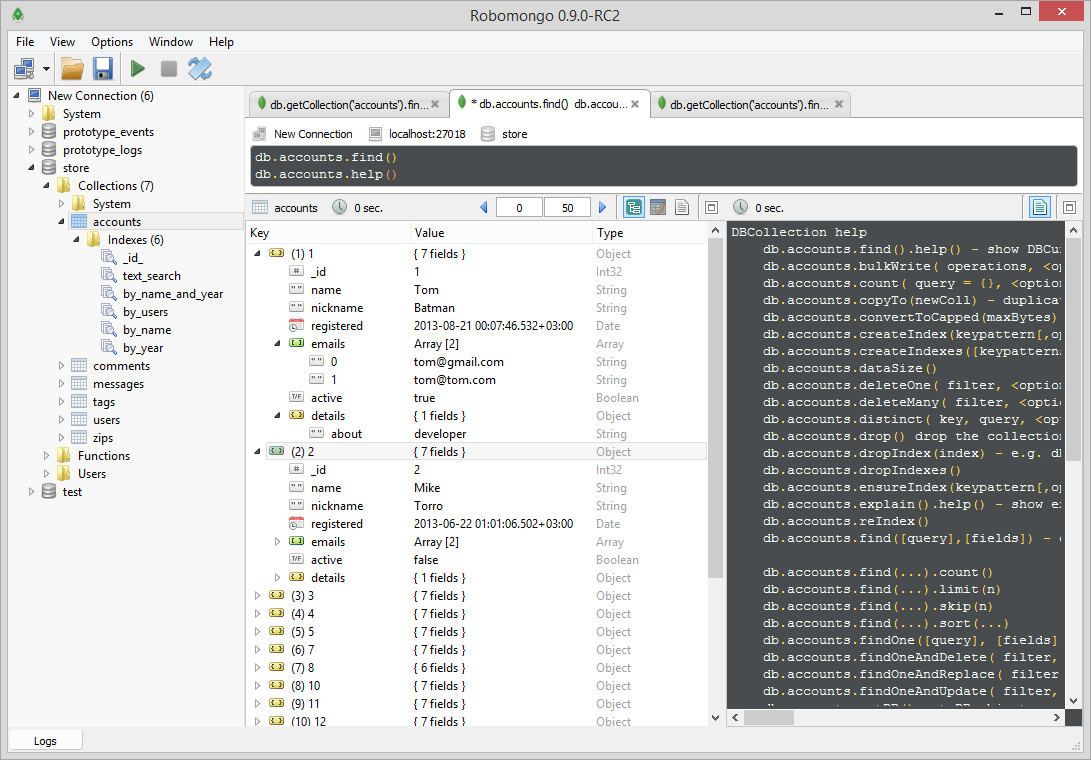
A ferramenta escolhida no final, foi a ferramenta MongoDB que contém uma comunidade e número de funcionalidades gigantesca. Além de ser orientada à documento, já possuí integrações com a linguagem escolhida, e fácil manuseio do Json para armazenamento. Outro ponto para escolha do mongo é quanto a possibilidade de expansão, com o sharding: possibilita partições do banco distribuído em diferentes máquinas.

O principal destaque do mongo se deve ao GridFS ou GridFileSystem, ferramenta que possuí armazenamento e indexação automática de arquivos. Particiona o arquivo como um documento separado. Por padrão, o GridFS usa um tamanho de bloco de 255 kB; isto é, o GridFS divide um arquivo em pedaços de 255 Kb, com exceção do último pedaço. Quando você consulta GridFS para um arquivo, o sistema irá remontar os pedaços conforme necessário [15].

O mongo também oferece execução de partes de código javascripts, permitindo a equipe de desenvolvimento criar rotinas diretamente no banco para alterar ou extrair alguma parte do dado.

## 5.3.1 RoboMongo

Figura 12: Interface RoboMongo



Fonte: BLOG ROMO MONGO

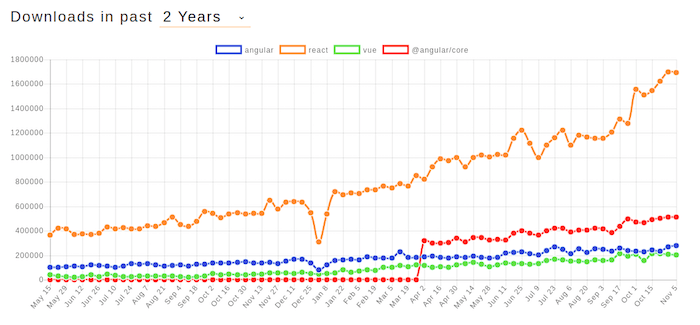
## 5.4. React

Para uma interface amigável à um usuário sem conhecimento técnico, foi escolhido desenvolver o front-end com a linguagem de marcação HTML, a linguagem de desenvolvimento JavaScript e a biblioteca React.

O React vem se popularizando com os desenvolvedores front-end. É uma biblioteca JavaScript declarativa, eficiente e flexível criada pelo Facebook para conseguir gerenciar melhor o seu feed, principalmente as notificações. A grande vantagem é que ao usá-lo, você facilitará a construção, a manipulação e a atualização do estado da sua tela, sem precisar recarrega-la a todo momento.

JavaScript é a linguagem de desenvolvimento mais popular entre desenvolvedores atualmente, segundo o StackOverflow Survey, 2017, movendo uma comunidade de desenvolvedores e plataformas para facilitar seu uso. O modo como o React trabalha para criar interfaces de usuário (ou User Interfaces, as UIs) é por meio da quebra de toda a estrutura da aplicação em componentes. O que o React propõe é: separar todo o código em pequenas partes (em arquivos diferentes), que se comportam como componentes reutilizáveis. Além de tudo isso, o react pode ser reutilizável para múltiplas plataformas migrando para o react native [17]. Na Figura 13 podemos observar o seu crescimento em relação à outras tecnologias emergentes com a mesma finalidade.

Figura 13: comparação do uso do React em relação a outras tecnologias em 2017



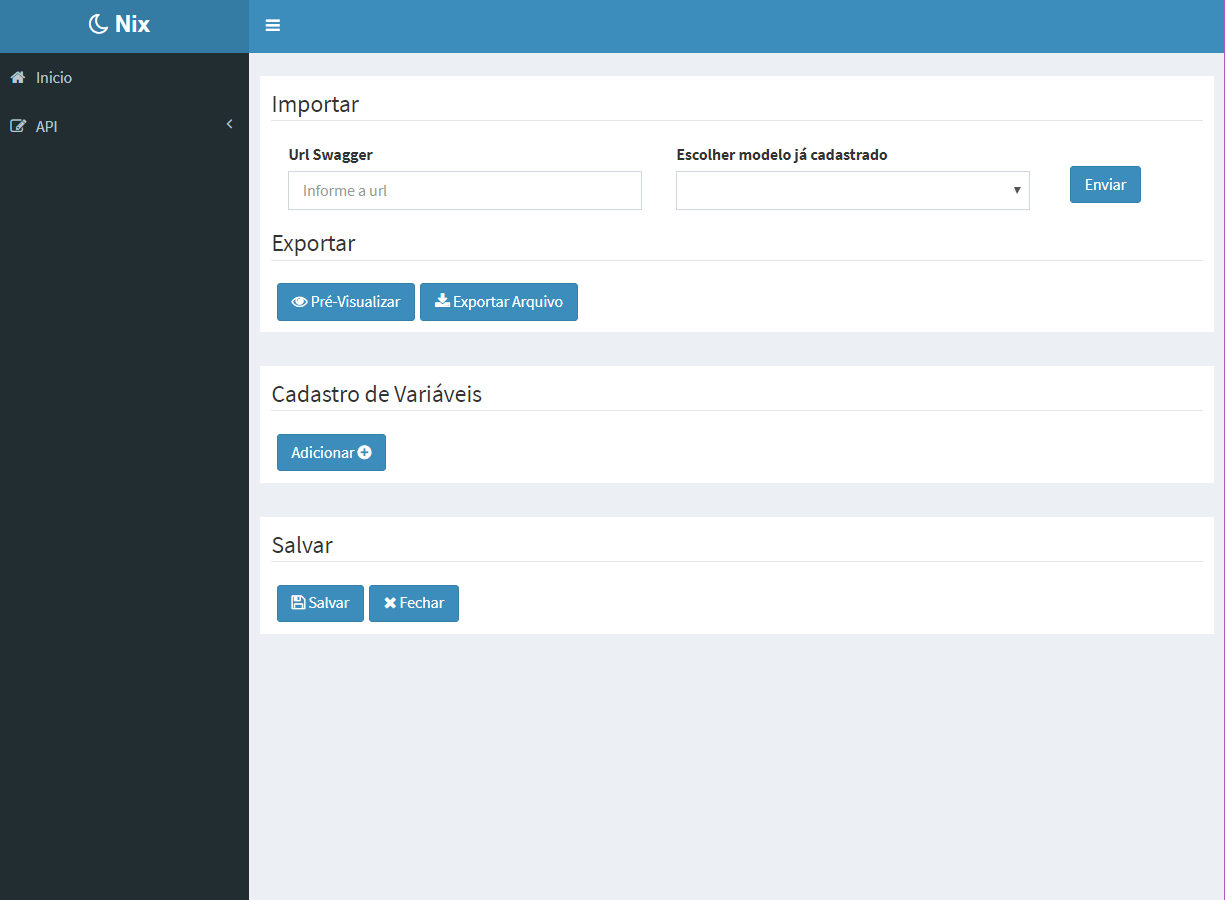
Fonte: Udacity

# 6. Implementação

Neste tópico será abordado a implementação do sistema, suas telas e funcionalidades.

O sistema tem como tela inicial a possibilidade de importar as informações iniciais da sua API, ou escolher um modelo já cadastrado no banco de dados, podemos visualizar a tela na Figura 14.

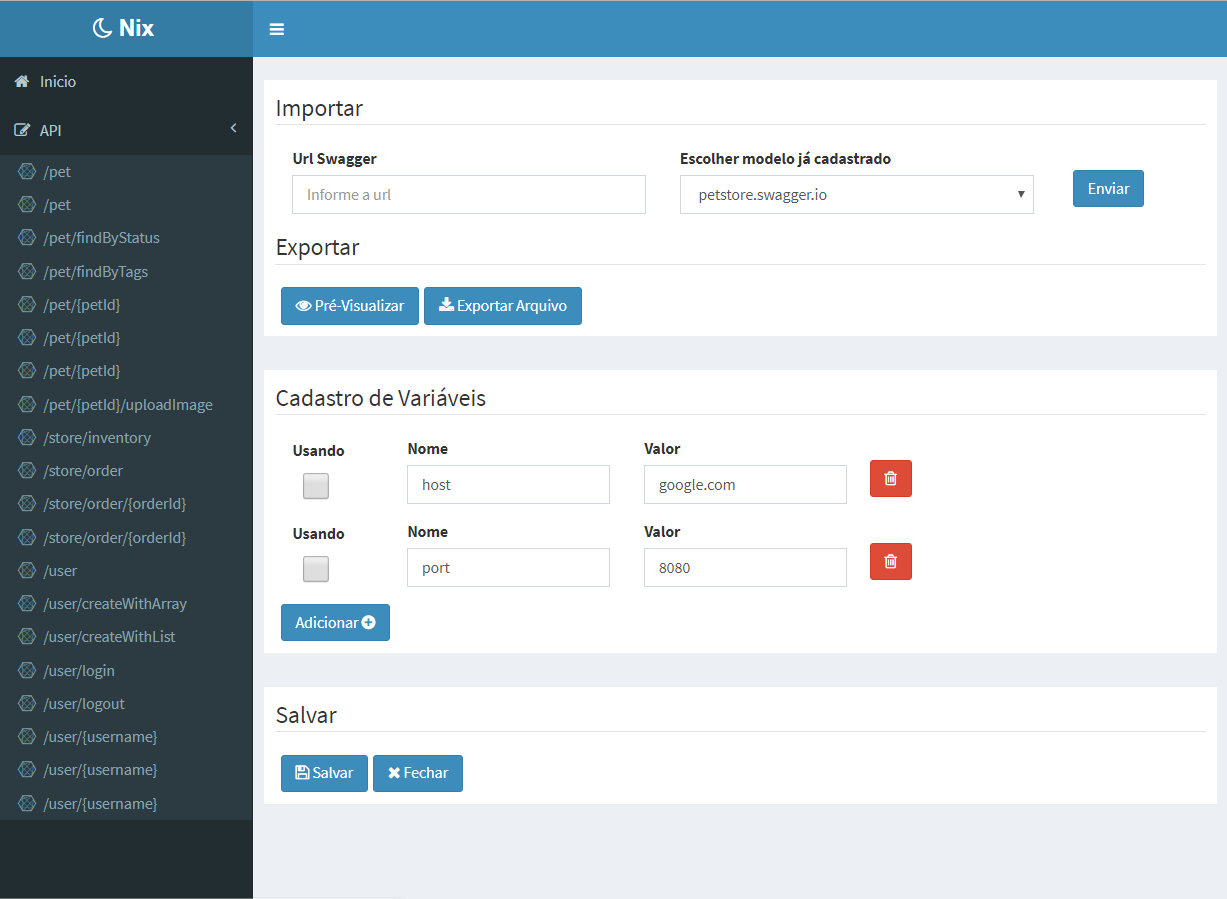
Figura 14: Tela inicial da importação do cadastro



Fonte: própria autora

Na Figura 15 é possível ver um modelo já cadastrado foi importado para a tela inicial, apresentando na coluna a esquerda suas requisições possíveis que redirecionará para a execução ou alteração dos campos para essas requisições.

Figura 15: Tela inicial com API selecionada

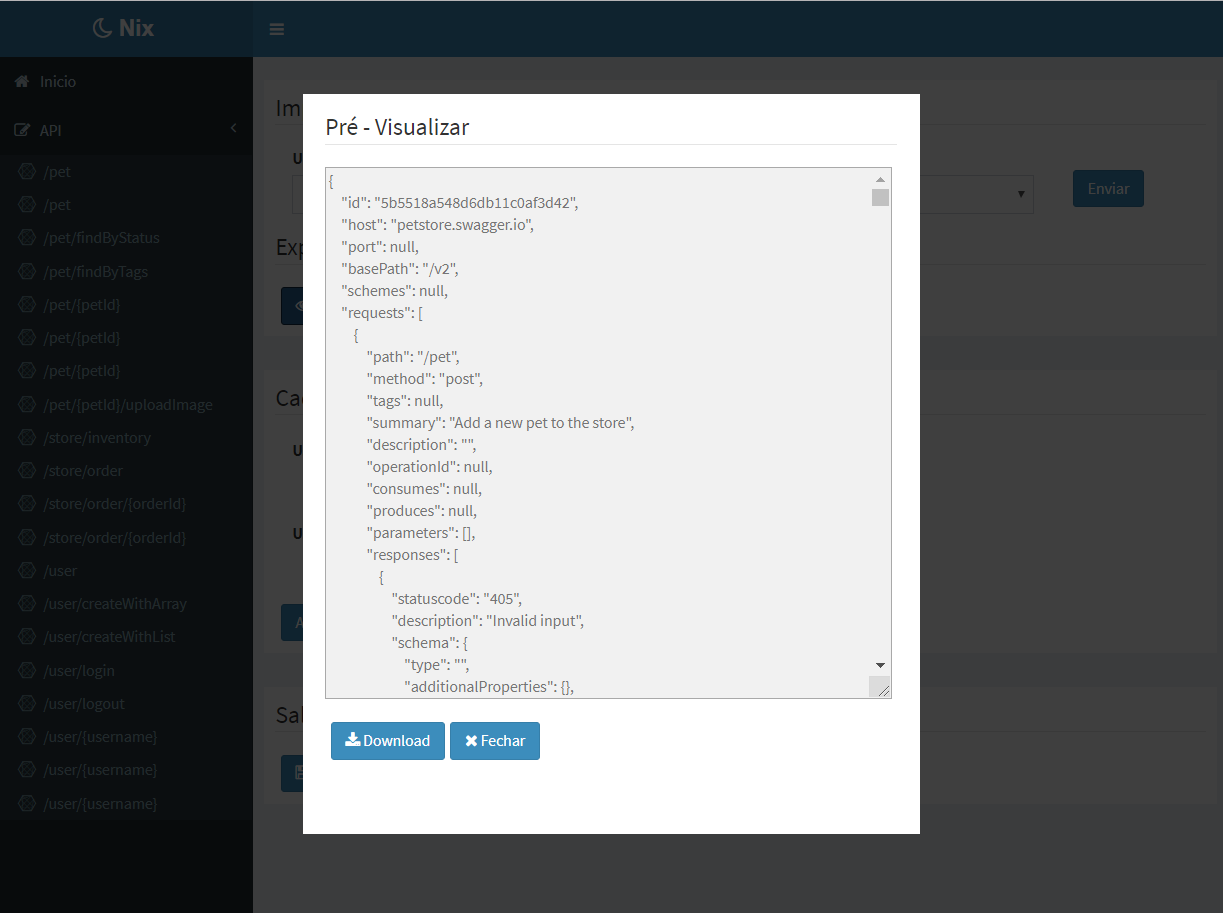


Fonte: própria autora

Na seção de cadastro de variáveis é possível incluir variáveis globais para uso nas requisições, o cadastro dessas variáveis é ilimitado e permite ao usuário melhor aproveitamento do sistema.

Na Figura 16, é possível exportar o arquivo que está na manipulação atual, ou seja todas as requisições cadastradas e as variáveis também. Há uma pré-visualização desse arquivo para o usuário analisar.

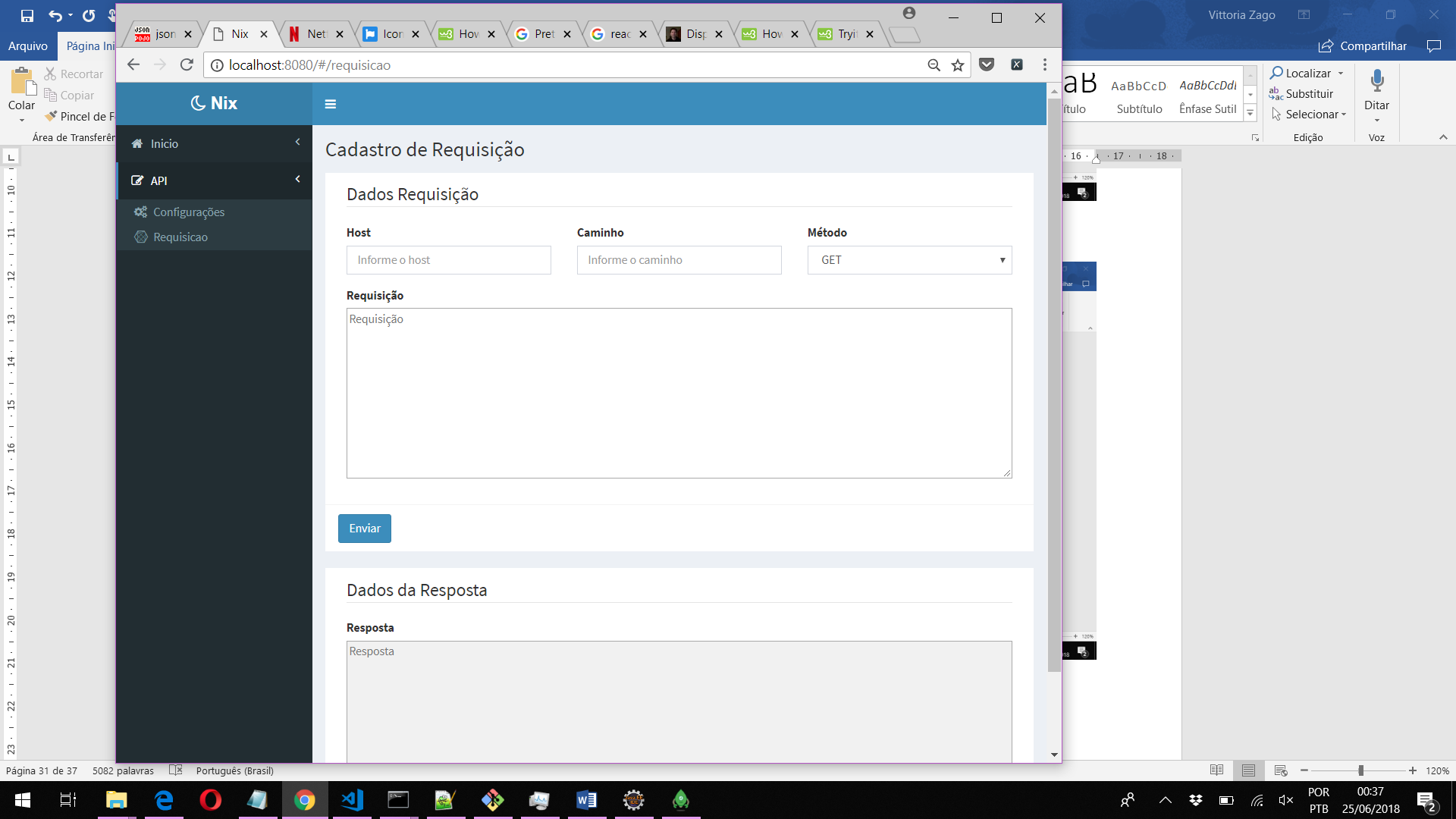
Figura 16: Exportar arquivo



Fonte: própria autora

O cadastro da requisição poderá ser feito manualmente. O usuário terá a opção de preencher um formulário em branco como mostra a Figura 17. Em caso de ter um arquivo importado, os dados da requisição já virão preenchidos, o usuário pode ou não altera-los, ou ainda cadastrar manualmente outras requisições.

Figura 17: Cadastrar requisição



Fonte: própria autora

# 7. Procedimento de Teste e Validação do Projeto

Os métodos do backend, ou seja, do gateway e da API que integra ao banco de dados, terão testes unitários, mas o acesso ao banco será simulado, uma vez que o projeto será orientado a estrutura MVC, e o teste da camada de negócio não deve testar conexões com o banco. Enquanto o frontend terá um teste automatizado, que simulará o cadastro de uma requisição usando ao menos uma variável global e executará um teste de aceitação. Esses testes, onde se conhece a estrutura e possíveis comportamentos é conhecido como Teste da Caixa Branca ou Teste Estrutural [18].

Para validação integrada do projeto, uma API pública deve ser escolhida, esta API terá o Swagger implementado e seus dados importados para o sistema já desenvolvido. Os dados cadastrados serão alterados e testados sua execução, sua exportação também deverá ser testada. Sem ter acesso ao código fonte, esse teste irá testar os requisitos funcionais do sistema, ou seja, nas ações que o sistema deve desempenhar, também conhecido como Teste da Caixa Preta ou Teste Funcional [18].

# 8. Análise dos Riscos

Um dos riscos que cabe ao desenvolvimento desse sistema, é o gateway conseguir interpretar todas entradas e saídas, seja em formato Json, base64, XML, entre outros. O fracasso dessa etapa ocasionará em diversos atrasos para refatoração ao longo do desenvolvimento. Porém se alguns casos não forem previstos, ocasionará no ainda pior que é a falha do sistema em algumas situações.

Outro risco possível é erro de estimativa da complexidade com o desenvolvimento da interface para o usuário, visto a baixa familiaridade com essa frente de desenvolvimento. Apesar do estudo feito sobre as tecnologias novas, ainda é possível um desenvolvimento de menor qualidade nessa área devido ao pouco contato.

# 9. Cronograma do Projeto

Dado as funcionalidades apresentadas e tecnologias escolhidas, assim como suas dificuldades, foi montado um cronograma com a previsão de término de cada etapa.

O cronograma para o desenvolvimento do projeto pode ser visualizado no plano apresentado na Figura 18.

Figura 18: Cronograma do projeto

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Atividades |  | 15/03 | 15/04 | 15/05 | 30/06 | 15/07 | 15/08 | 15/09 | 15/10 | 15/11 |
|  |  |  |  |  |  |  |  |  |  |  |
| Especificação TCC |  |  |  |  |  |  |  |  |  |  |
| Revisão da Especificação |  |  |  |  |  |  |  |  |  |  |
| Gateway |  |  |  |  |  |  |  |  |  |  |
| Diagramas |  |  |  |  |  |  |  |  |  |  |
| Prototipo |  |  |  |  |  |  |  |  |  |  |
| Importação/Exportação |  |  |  |  |  |  |  |  |  |  |
| Integração Banco de Dados |  |  |  |  |  |  |  |  |  |  |
| FrontEnd |  |  |  |  |  |  |  |  |  |  |
| Integração e testes |  |  |  |  |  |  |  |  |  |  |
| Testes Validações |  |  |  |  |  |  |  |  |  |  |
| Preparos finais/Apresentação |  |  |  |  |  |  |  |  |  |  |

Fonte: Própria autora

As datas são uma estimativa considerando a dificuldade que cada etapa pode trazer, é sempre estudado uma margem de erro, podendo ter períodos menores para algumas partes ou o risco de extrapolar alguma data planejada.

# 10. Conclusão

Devido ao crescimento do desenvolvimento em camadas e popularização dos testes como garantia de sucesso na disponibilização do produto para o usuário, a busca pela facilidade de testar e integração contínua vem aquecendo o a procura dessas ferramentas. O sistema Nix tem como propósito ajudar um time de desenvolvimento a melhorar a entrega de produtos, qualidade de testes e consequentemente o tempo focado no core do negócio.

O projeto traz consigo uma grande responsabilidade: ter como público alvo especialistas em tecnologia, conhecidos por serem usuários críticos e grandes expectadores de tecnologias que facilitam. Mas isso também pode ser visto como uma oportunidade de crescimento para o sistema, com uma comunidade tão ativa não faltará suporte e sugestões.

É um desafio muito grande um projeto dessa magnitude competindo com várias ferramentas já consolidadas no mercado, mas este traz propostas inovadoras que acrescentarão muito a todo um ecossistema de desenvolvimento.

# 11. Referências Bibliográficas

[1] CANALTECH. **O que é API** . Disponível em < https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 15 mar. 2018.

[2] TECHCRUNCH. **The Rise of APIs.** 2016. Disponível em

< https://techcrunch.com/2016/05/21/the-rise-of-apis/>. Acesso em: 04 abr. 2018.

[3] DEVMEDIA. **Teste de integração na prática**. 2017. Disponível em

<https://www.devmedia.com.br/teste-de-integracao-na-pratica/31877/>. Acesso em: 15 mar. 2018.

[4] TESTE DE SOFTWARE. **Teste de integração**. 2017. Disponível em

<http://testesdesoftware.com/teste-de-integracao//>. Acesso em: 15 mar. 2018.

[5] MARTIN FOWLER. **Test pyramid.** 2012. Disponível em

< https://martinfowler.com/bliki/TestPyramid.html>. Acesso em: 04 abr. 2018.

[6] DEVMEDIA. **JMine realizando testes de integração**. 2017. Disponível em

<https://www.devmedia.com.br/jmine-realizando-testes-de-integracao/29297/>. Acesso em: 15 mar. 2018.

[7] JULIO DE LIMAS. **Teste em APIs com Rest Assured**. 2015. Disponível em

<https://pt.slideshare.net/juliodelimas/tdc2015-testes-em-APIs-rest-com-restassured

imagem JMeter/>. Acesso em: 15 mar. 2018.

[8] MATERA. **Medindo performance de uma api**. 2017. Disponível em

<matera.com/br/2017/02/21/medindo-performance-de-uma-API-rest/

[9] TECHBEACON. **5 melhores ferramentas testes para API**. 2017. Disponível em

<https://techbeacon.com/5-top-open-source-API-testing-tools-how-choose

imagem rest-assured/>. Acesso em: 15 mar. 2018.

[10] BLOG NAHURST.  **Guia visual para nosql** 2017. Disponível em

<http://blog.nahurst.com/visual-guide-to-nosql-systems/>. Acesso em: 29 mar. 2018.

[11] DICAS DE PROGRAMAÇÃO. **6 motivos para usar bancos de dados nosql**. 2017. Disponível em

<https://dicasdeprogramacao.com.br/6-motivos-para-usar-bancos-de-dados-nosql//>. Acesso em: 29 mar. 2018.

[12] TOM CHRISTIE. **Framework em rest.** 2017. Disponível em

<http://www.tomchristie.com/rest-framework-2-docs/topics/documenting-your-API/>. Acesso em: 15 mar. 2018.

[13] TAOFFI. **Swagger Navegação e Definições**. 2017. Disponível em

<http://taoffi.isosoft.org/post/2016/05/25/swagger-browsing-rest-service-definitions/>. Acesso em: 15 mar. 2018.

[14] SEARCH DATA MANAGEMENT. **Critério para escolher diferentes tipos de bancos nosql**. 2017. Disponível em

<http://searchdatamanagement.techtarget.com/feature/Key-criteria-for-choosing-different-types-of-NoSQL-databases/>. Acesso em: 29 mar. 2018.

[15] IMASTERS. **3 Razões para usar mongodb.** 2017. Disponível em

<https://imasters.com.br/artigo/18334/mongodb/3-razoes-para-usar-mongodb/?trace=1519021197&source=single/>. Acesso em: 29 mar. 2018.

[16] BLOG ROMO MONGO. **RoboMongp para Windows, mac e linux.** 2016. Disponível em

<http://blog.robomongo.org/robomongo-rc2-for-windows-mac-os-x-and-linux/>. Acesso em: 04 abr. 2018.

[17] UDACITY. **React o que é e como funciona.** 2017. Disponível em

<https://br.udacity.com/blog/post/react-o-que-e-como-funciona#/>. Acesso em: 29 mar. 2018.

[18] BASE2. **Conceitos: testes de caixa branca e caixa preta.** 2013. Disponível em

<http://www.base2.com.br/2013/07/24/teste-caixa-branca-caixa-preta//>. Acesso em: 24 jun. 2018.