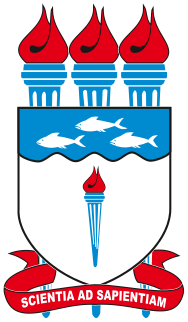

Universidade Federal de Alagoas
Curso de Ciência da Computação

Arquitetura e implementação de um sistema de gerenciamento de notas fiscais eletrônicas

Camilo Costa Campos

Maceió
2016

Camilo Costa Campos

Arquitetura e implementação de um sistema de gerenciamento de notas fiscais eletrônicas

Monografia apresentada para obtenção do Grau de Bacharel em Ciência da Computação pela Universidade Federal de Alagoas.

Orientador: Prof. Dr. Leandro Melo de Sales

Maceió
2016

Camilo Costa Campos

Arquitetura e implementação de um sistema de gerenciamento de notas fiscais eletrônicas

Monografia de Projeto Final de Graduação sob o título "Arquitetura e implementação de um sistema de gerenciamento de notas fiscais eletrônicas", defendida por Camilo Costa Campos e aprovada em 29 de julho de 2016, em Maceió, Estado de Alagoas, pela banca examinadora constituída pelos professores:

Banca Examinadora:

Prof. Dr. Leandro Melo de Sales
Universidade Federal de Alagoas
Orientador

Prof. Msc. Rodrigo José Sarmento Peixoto
Universidade Federal de Alagoas
Examinador

Prof. Dr. Thiago Bruno Melo de Sales
Universidade Federal de Alagoas
Examinador

Para o meu pai que sempre me apoiou em tudo.

Resumo

Neste trabalho, apresenta-se uma API/biblioteca para armazenagem e manipulação de notas fiscais eletrônicas e uma aplicação que utiliza essa API para fornecer um serviço de nota fiscal pessoal. Esse sistema foi desenvolvido com a pilha de soluções MEAN e a arquitetura REST. Neste trabalho, explica-se detalhes do funcionamento do sistema, as tecnologias utilizadas e as decisões tomadas durante a implementação. Em seguida, detalham-se as funcionalidades do serviço de nota fiscal pessoal abordado como estudo de caso.

Palavras-chave: Nota Fiscal Eletrônica, Biblioteca, API, REST, Javascript.

Abstract

In this work is presented an API/library for storage and handling of electronic invoices and an application that uses this API to provide a personal service tax invoice. This system was developed with the stack MEAN solutions and REST architecture. In this work is explained details of system operation, the technology used and the decisions made during implementation. Next details the features of personal tax invoice service approached as a case study.

Keywords: Eletronic tax invoice, Library, API, REST, Javascript.

Lista de Figuras

Lista de abreviaturas e siglas

API - Application Programming Interface

BSON - Binary JSON

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IIFE - Immediately Invoked Function Expression

JSON - JavaScript Object Notation

MEAN - MongoDb Express Angular Nodejs

MVC - Model View Controller

NF - Nota Fiscal

NF-e - Nota Fiscal eletrônica

NFC-e - Nota Fiscal do Consumidor eletrônica

QR Code - Quick Response Code

REST - Representational State Transfer

SPA - Single Page Application

SSL - Secure Socket Layer

URI - Uniform Resource Identifier

XML - eXtensible Markup Language

Sumário

1

Introdução

A nota fiscal (NF) é um documento que registra a transferência de uma propriedade ou a realização de um serviço. A sua existência é fundamental para a sociedade, pois sua posse garante direitos ao consumidor, autorização de transporte pelas empresas de frete e recolhimento de impostos pelo governo.

Em 2005, instituiu-se a nota fiscal eletrônica (NF-e), a qual consiste em um documento digital com o objetivo de substituir a nota fiscal convencional e que contém todas as informações necessárias para tal. A partir do projeto da NF-e desenvolveu-se a nota fiscal do consumidor eletrônica (NFC-e), a qual se encontra em fase de testes e sua adoção será gradual nos próximos anos.

Esse processo de mudança nas notas fiscais criará diversas oportunidades para o desenvolvimento de sistemas que automatizem a manipulação e o armazenamento de suas informações. Com o objetivo de tornar mais fácil o desenvolvimento desses sistemas, neste trabalho, decidiu-se implementar uma *Application Programming Interface* (API) que forneça as funcionalidades necessárias para carregar, processar e exibir informações sobre as NF-e.

Como cenário de uso, desenvolveu-se um sistema de administração de NF com o nome de Nota Fiscal Pessoal, onde um usuário cadastrado pode consultar, submeter, modificar e remover suas NF-e.

O caso de uso e a API foram desenvolvidos utilizando a pilha de soluções MEAN[1] e disponibilizados via modelo *Software as a Service* (SaaS).

1.1 Estado da arte

O estado da arte apresentado nesse trabalho aborda às técnicas e ferramentas utilizadas no desenvolvimento das solução computacionais. Os trabalhos a seguir são relevantes para o assunto e representam o que há de atual na implementação da API, no funcionamento

[1] `http://meanjs.org/`

da pilha de soluções MEAN e no modelo Software as a Service.

Título: *Design Patterns and Extensibility of REST API for Networking Applications*

Autores: *Li Li, Wu Chou, Wei Zhou, Min Luo*

Instituição: IEEE

Ideias e conclusões: Uma API RESTfull está sob constante mudança e atualização, para facilitar essa atualização foi proposto um arcabouço baseado em redes de Petri que fornece um modelo formal facilmente extensível.

Ano: 2016

Título: *Using the MEAN Stack to Implement a RESTful Service for an Internet of Things Application*

Autores: *Andrew John Poulter, Steven J. Johnston, Simon J. Cox*

Instituição: University of Southampton

Ideias e conclusões: Uma visão geral sobre os componentes da pilha MEAN e a demonstração dos beneficios de sua utilização em uma API para aplicações para Internet das coisas.

Ano: 2015

Título: *Frontiers in Information and Software as Services*

Autores: *K. Selçuk Candan, Wen-Syan Li, Thomas Phan, Minqi Zhou*

Instituição: Arizona State University

Ideias e conclusões: Uma visão geral sobre os desafios e oportunidades encontrados ao desenvolver uma aplicação SaaS e sugestões sobre a direção e a forma que esse modelo seguirá.

Ano: 2009

1.2 Problemática

É comum uma pessoa após comprar um aparelho eletrônico precisar recorrer à garantia do fabricante. Mas será que a NF está guardada e preservada? Será fácil localizar a NF referente ao determinado produto em meio a tantas outras? Postulamos que uma solução para esse problema seria, para o usuário final, utilizar um aplicativo de gerenciamento de NF-e. Caso surjam necessidades específicas, um desenvolvedor de aplicativos pode utilizar uma API/biblioteca para facilitar a criação da sua própria solução.

Um usuário que deseja organizar suas NF-e terá algumas dificuldades, pois as informações contidas na NF-e estão em um arquivo com formato XML e este demanda um programa para exibir os dados de forma amigável, e haverá trabalho repetitivo para pesquisar todas as NF-e e retornar os valores gastos de acordo com algum critério.

Supondo o caso de um desenvolvedor que pretenda criar um programa capaz de armazenar e consultar notas fiscais, sem dispor de tempo para estudar as particularidades da nota fiscal e nem de uma infraestrutura para acessar os dados destas remotamente. Uma solução para o seu problema seria utilizar uma API que abstraísse os detalhes técnicos da tecnologia utilizada, bem como regras legais relativas à nota fiscal. A escolha equivocada com relação as tecnologias utilizadas pode acarretar em um sistema lento, inseguro, de difícil manutenção e de grande consumo dos recursos computacionais.

1.3 Objetivos

O objetivo deste trabalho é desenvolver uma API/biblioteca capaz de armazenar e manipular os dados referentes a NF-e, de modo a oferecer um ambiente simples para que desenvolvedores possam criar aplicações.

1.4 Metodologia

A metodologia do trabalho consiste em:

- **Pesquisa bibliográfica e técnica:** Realizou-se um levantamento dos principais artigos e publicações sobre as ferramentas e paradigmas utilizados para o desenvolvimento do software, bem como, das leis e portarias relativas a criação e funcionamento das diversas modalidades de NF;
- **Avaliação das ferramentas:** As ferramentas utilizadas foram avaliadas através de pequenos teste, certificando-se quanto a eficácia e eficiência em comparação com ferramentas similares;
- **Desenvolvimento:** O desenvolvimento e testes da API/biblioteca foram realizados

utilizando uma máquina com sistema operacional Linux Ubuntu 16.04[2], o navegador web Google Chrome[3], o editor de texto gedit[4] e o gerenciador de versões git[5];

- **Validação:** A validação da API/biblioteca foi realizada através do caso de uso apresentado neste trabalho.

1.5 Estrutura da Monografia

No Capítulo 2, apresentam-se as revisões bibliográficas e técnicas. No Capítulo 3, aborda-se a API/biblioteca. No Capítulo 4, apresenta-se os estudos de caso. No Capítulo 5, apresenta-se as conclusões e trabalhos futuros.

[2] http://www.ubuntu.com/
[3] https://www.google.com/chrome/
[4] https://wiki.gnome.org/Apps/Gedit
[5] https://git-scm.com/

2

Revisão bibliográfica e técnica

Neste capítulo, apresenta-se uma revisão bibliográfica e técnica dos principais assuntos abordados durante este trabalho.

2.1 Nota Fiscal Eletrônica

Uma nota fiscal é um documento que registra a transferência de uma propriedade ou a realização de um serviço e precisa ser informada ao órgão competente (secretaria ou ministério da fazenda) para recolhimento de impostos.[1]

Com o objetivo de agilizar, simplificar e tornar mais segura a emissão de notas fiscais, publicou-se no diário oficial da união em 5/10/2005 [2] o ajuste que institui a nota fiscal eletrônica (NF-e), a qual se caracteriza por ser gerada, transmitida, armazenada e assinada por meios digitais em contraste aos meios analógicos baseados em papel.

2.1.1 Origem

Em julho de 2004, realizou-se o I Encontro Nacional de Administradores Tributários (ENAT), este evento tinha como objetivo buscar soluções para facilitar a integração e intercambio de informações entre os diversos órgãos tributário brasileiros. Durante o ENAT instituiu-se a nota fiscal eletrônica (NF-e).

Em agosto de 2005 durante o II encontro nacional de administradores tributários, redigiu-se o protocolo de cooperação Nº 03 / 2005 [3], um documento que define o esforço conjunto para a criação da nota fiscal eletrônica.

CLÁUSULA SEGUNDA – No desenvolvimento da NF-e, serão observados os seguintes pressupostos, entre outros que vierem a ser definidos de comum acordo pelos partícipes:

I - substituição das notas fiscais em papel por documento eletrônico;

[1] http://www.nfe.fazenda.gov.br

[2] http://www1.fazenda.gov.br/confaz/confaz/ajustes/2005/AJ_007_05.htm

[3] https://www.fazenda.sp.gov.br/nfe/legislacao/legislacao_protocolo_ENAT_03_2005.pdf

II - validade jurídica dos documentos digitais;

III - padronização nacional da NF-e;

IV - mínima interferência no ambiente operacional do contribuinte;

V - compartilhamento da NF-e entre as administrações tributárias;

VI - preservação do sigilo fiscal, nos termos do Código Tributário Nacional.

Em 15 de setembro de 2006 emitiu-se a primeira NF-e com validade jurídica, em 2007 após a publicação do Protocolo ICMS nº. 10/2007 [4] a NF-e deixa de ser facultativa a todos os setores e torna-se obrigatória para combustíveis e cigarros, posteriores ajustes incluíram outros setores na obrigatoriedade.

2.1.2 Vantagens e Desvantagens

A mudança de nota fiscal convencional para a nota fiscal eletrônica trouxe várias benefícios tanto para o governo, as empresas e para os consumidores, de modo geral os benefícios estão ligados à diminuição de custo, desburocratização e maior confiabilidade. Além destes podemos especificar outros benefícios, tais como:

- Menor consumo de papel;
- Menor espaço dedicado a armazenamento;
- Menor quantidade de erros;
- Envio via e-mail;
- Impresso via impressora não fiscal.

As desvantagens estão relacionadas a necessidade de investimentos em novos sistemas de controle, em tempo e energia para se estudar as mudanças na legislação e o risco de indisponibilidade do sistema devido a falhas na internet.

2.1.3 Nota Fiscal do Consumidor Eletrônica

A nota fiscal do consumidor eletrônica (NFC-e) instituida através do ajuste SINIEF 22, de 6 de dezembro de 2013 [5], é um projeto derivado da NF-e, sua versão atual (1.2) se encontra em estágio de testes e visa substituir os cupons fiscais, tendo como foco os consumidores do varejo. Ao contrário da NF-e os dados de identificação do consumidor como CPF endereço e nome não são obrigatórios, também é facultativo a descriminação

[4] http://www1.fazenda.gov.br/confaz/confaz/Protocolos/ICMS/2007/pt010_07.htm
[5] http://www1.fazenda.gov.br/confaz/confaz/ajustes/2013/AJ_022_13.htm

dos impostos pagos. Além dessas diferenças, a NFC-e possui um campo para descrever a forma de pagamento (dinheiro, cheque, cartão), o qual não existe na NF-e.

A NFC-e quando impressa contém um *QR Code (Quick Response Code)* com um endereço do sistema de consulta da secretária da fazenda e a chave de acesso da compra, o qual permite visualizar as informações detalhadas, comprovando sua validade.

Por fim, a NFC-e conta ainda com um sistema de contingência que permite que a nota fiscal seja gerada mesmo quando não há conexão com o servidor, bastando para tal que as informações sejam enviadas dentro de 24 horas.

2.2 Pilha de solução MEAN

Uma pilha de solução é um conjunto de componentes necessários para o desenvolvimento de uma solução para um problema, onde geralmente esses componentes são desenvolvidos por equipes independentes.

Mean é o acrônimo para MongoDB, Express, Angular e NodeJS e se propõe a ser uma forma de desenvolver aplicações web modularizadas em que a comunicação entre as diferentes camadas acontece de forma eficiente e fácil de entender. As quatro camadas da pilha e seus objetivos são:

- **MongoDB** para armazenar os dados do programa;
- **Express** para lidar com a lógica de comunicação;
- **Angular** para gerar uma interface;
- **NodeJS** para armazenar a lógica do programa.

Todas as camadas utilizam a linguagem Javascript e o formato JSON.

2.2.1 Javascript

"Javascript é uma linguagem de script multi plataforma orientada a objetos"[6]. Javascript é uma linguagem interpretada, baseada em protótipos e de tipagem fraca. É também multi-paradigma de forma que ela suporta os paradigmas orientado a objeto, imperativo e funcional.

2.2.2 JSON

A comunicação entre as camadas do software entre outras tarefas é feita através do formato JSON, formato esse definido pela como "formato leve de intercambio de dados

[6] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

independente de linguagem baseado em texto"[7], o formato JSON pode ser facilmente entendido por humanos e é o padrão utilizado pelos outros componentes da pilha de soluções MEAN.

2.2.3 NodeJS

"NodeJS é uma plataforma construida sobre o motor javascript V8 do Chrome. Nodejs usa um modelo de leitura e escrita dirigido a eventos e não bloqueante, que o torna leve e eficiente"[8].

NodeJS é um ambiente javascript para servidores, majoritariamente implementado em C e C++ com foco em performance e baixo consumo de memória[S. Tilkov, S. Vinoski 2010].

NodeJS tem como principais características ser:

Single-threaded

Single-threaded implica que há apenas uma thread encarregada de todas as operações e que dispõe de todos os recursos computacionais, dessa forma não há subutilização dos recurso devido a uma thread ociosa, ao contrário do que ocorre com *multi-thread*.

Event-driven

O paradigma *event-driven* cria um laço de eventos que espera por notificações e responde de acordo, diferente dos paradigmas estruturais em que o fluxo do programa é mais rígido.

As características mencionadas acima implicam em NodeJS ser assíncrono e não bloqueante, ou seja é necessário desenvolver os programas tendo em mente que a interação com o usuário, processos de leitura e escrita ou os erros de rede podem surgir a qualquer momento e que precisam ser tratados.

2.2.4 MongoDB

MongoDB é um banco de dados NoSQL *(No Only SQL)* com esquemas dinâmicos e orientado a documentos, por armazenar os dados internamente em BSON *(binary JSON)*, a leitura e escrita de dados em formato JSON é otimizada. Os documentos são organizados de forma hierárquica ao invés da forma relacional utilizada pelos bancos de dado SQL.

Validação do banco de Dados

Por validação de banco de dados entendemos o conjunto de regras que determinam o tipo de dado e intervalo dos valores que um campo pode ter, no MongoDB essas restrições devem ser definidas, pelo programador, na aplicação cliente.

[7] `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`
[8] `https://nodejs.org`

Modelagem do banco de dados

MongoDB utiliza uma abordagem de esquema flexível para armazenamento dos dados, de modo que o responsável pela verificação de conformidade dos dados precisa ser a camada de aplicação, diferente dos bancos de dados relacionais cuja verificação é responsabilidade da própria camada de banco de dados.

Comparação com bancos de dados relacionais

MongoDB persiste os dados na forma de um documento, ao contrário dos bancos de dados relacionais como postgresql[9] que armazena os dados em tabelas. Em um contexto em que os dados foram planejados de forma hierarquica, as solicitações ao banco de dados serão mais simples, pois os dados estão armazenados em sequência no disco e não é necessária a execução de joins, a mesma requisição será realizada de forma mais complexa em um banco de dados relacional, visto que há um custo computacional ao realizar a operação de união entre diversas tabelas, bem como, a leitura destes dados levará mais tempo por não estarem armazenados em sequência no disco rígido.

2.2.5 Angular

"AngularJS ou Angular é um arcabouço estrutural para aplicações web dinâmicas"[10]. Angular foi planejado com foco em desenvolver *single-page applications (SPA)* onde a estrutura da página se mantém e apenas o conteúdo é modificado.

Vantagens do Angular

Client-side: Implica que parte do processamento que tradicionalmente fica no servidor é transferido para o lado cliente, dessa forma podemos ter um sistema mais escalável.

Two-ways data binding: Quando um dado é modificado, seja pelo usuário ou pelo programa, o outro é notificado e tem seu valor alterado, assim temos confiança de que as informações que estamos vendo são as mesmas que o sistema manipula e vice versa.

Dependency injection: Simplifica o carregamento de componentes, carregando-os somente quando estes forem necessários, facilitando a sua substituição por outro sem comprometer o código já escrito.

Immediately Invoked Function Expression (IIFE): As variáveis de uma função existem somente durante sua execução, dessa forma evitasse poluir o escopo com variáveis globais e são removodas da memória após sua utilização.

[9] `https://www.postgresql.org/`
[10] `https://angularjs.org/`

Conceitos

Module

Um sistema web pode ser composto de diversas funcionalidades independentes, sendo estas feitas com angular ou não, para cada uma dessas funcionalidades damos o nome de *module*, estes são referenciados dentro do HTML pela *tag ng-app*.

Controller

O *controller* é responsável pela lógica da aplicação, suas funções e dados são expostos para o *scope*, dessa forma o *controller* não precisa trocar informações diretamente com a *view*. As funções de um *controller* podem ser modularizadas em um *service* e a criação de objetos em uma *factory*, os quais podem ser injetados dentro de um controller, dessa forma temos reuso de código quando *controllers* diferentes precisem de uma mesma funcionalidade ou objeto.

View

Devido ao foco em SPA, a *view* costuma consistir em um *template* HTML com alguns campos de valor variáveis os quais são chamados *expressions* e são referenciados por objeto.atributo. A *view* pode ser é composta por *directives*, estes são os únicos componentes que podem acessar o *Document Object Model (DOM)* e modificar seus atributos.

Model

O *model* como conhecemos na arquitetura *Model View Controller* (MVC) é exercido pelo objeto *scope*, graças ao *data binding* as alterações realizadas pelo usuário na view são transmitidas para o *controller* via *events*.

Diagramas

Data Binding

Na Figura 2.1 temos o Diagrama de *data binding*, o qual demonstra que as variáveis *cost* e *qty*, declaradas no *Scope*, estão sendo referenciadas em campos *input* dentro da *View*, uma alteração do valor nesses campos será informada ao *Scope* que atualizará suas variáveis, em seguida o *Scope* propaga os novos valores de volta a *View* atualizando a expressão {{*qty* * *cost*}}.

MVC

Na Figura 2.2 é apresentado o Diagrama MVC, onde está representada a forma com que a *View* disponibiliza os valores de *qty* e *cost* para o *Controller* declarado no *Scope*, e como a *View* executa o método *invoice.total()* e *invoice.pay()* definidos em *Controller*.

2.2.6 Express

"Express é um framework web rápido, flexível e minimalista para Nodejs."[11]

[11] `http://expressjs.com/`

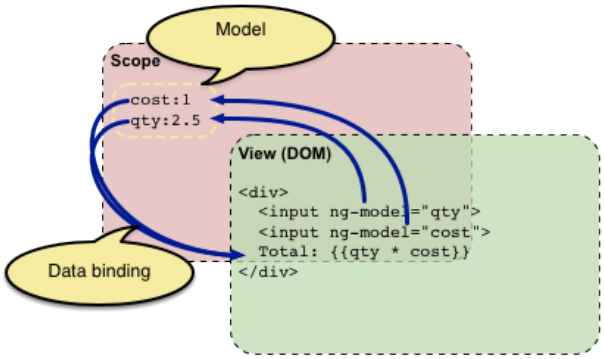

Figura 2.1: Diagrama de data binding
Fonte: https://docs.angularjs.org/guide/concepts

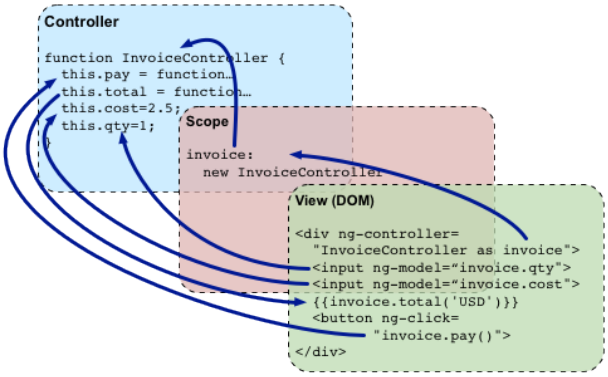

Figura 2.2: Diagrama MVC
Fonte: https://docs.angularjs.org/guide/concepts

O proposito do Express é disponibilizar uma série de funcionalidades de forma mais simples e objetiva do que teríamos utilizando nodejs diretamente. Entre as funcionalidades que utilizamos no desenvolvimento do API/biblioteca podemos destacar:

- A criação e configuração do *web service*;
- Definição de rotas;
- O tratamento de erros relacionados a URL;
- Envio e inicialização dos arquivos necessários;
- Gerenciamento da API;
- Intermediação da comunicação entre as outras camadas da pilha.

2.3 Bilioteca

Uma bilioteca é um conjunto de classes que incorpora um design abstrato para soluções em uma família de problemas relacionados, e auxilia o reuso em uma granularidade maior do que classes [R. E. Johnson, B. Foote 1988].

2.4 API

Uma *Application Programming Interface (API)* é um conjunto de funcionalidades fornecido por um componente de terceiros (ex. bibliotecas e arcabouços) o qual é disponibilizado para desenvolvedores de software [R. E. Johnson, B. Foote 1988].

2.4.1 REST

Representational State Transfer (REST) é uma arquitetura para a construção de *web services*, criada por Roy Thomas Fielding em sua tese de doutorado [Fielding 2000], na qual ele enfatiza seis principais características:

Cliente Servidor

A separação entre cliente e servidor, desse modo cada um pode evoluir independentemente e tornar mais fácil se adaptar a novas situações.

Sem estado

A interação entre cliente e servidor deve ser realizada enviando todas as informações necessárias de modo que o servidor não precise armazenar dados de sessão do usuário.

Cache

Cacheabilidade é a capacidade de armazenar a resposta de uma solicitação, com o objetivo de melhorar o custo computacional e a eficiência de rede, para tal é necessário rotular os dados como possível de cache ou não.

Interface Uniforme

A uniformidade da interface existe para desacoplar o cliente do servidor e encorajar a evolução independente do sistema. A interface deve receber requisições composta pelo recurso individual representado no *Uniform Resource Identifier (URI)* e pelo método HTTP, em seguida deve responder com uma resposta HTTP.

Sistema em camadas

A arquitetura do sistema deve possuir camadas hierarquizadas de modo que cada camada só interaja com as camadas adjacentes, essa organização facilita o balanceamento da carga e o reaproveitamento de sistemas legados.

Código sob demanda

Essa é uma característica opcional, a qual pode permitir que funcionalidades do cliente sejam estendidas através de *download* de código, tanto na forma de *applets* Java ou

scripts Javascript, assim, tem-se um cliente mais simples ao diminuir a quantidade de funcionalidades pré-implementadas.

2.5 CRUD

CRUD é o acrônimo de *Create, Read, Update e Delete*, que são as quatro operações básicas de persistência em banco de dados. Cada uma das operações e métodos HTTP estão associados da seguinte forma:

Método	Operação
GET	READ
POST	CREATE
PUT	UPDATE
DELETE	DELETE

Tabela 2.1: Metodo X Operação

3

API/Arcabouço para sistemas de notas fiscais

O objetivo deste trabalho é definir e implementar uma solução que facilite e aumente a produtividade de desenvolvedores que trabalham com NF-e. Concluiu-se que a melhor forma de se atingir esse objetivo é através de uma API/arcabouço que abstrai detalhes das tecnologias envolvidas.

Os arcabouços, bibliotecas e paradigmas utilizadas na implementação da API/arcabouço foram escolhidas por priorizarem reuso de código, facilidade de uso e escalabilidade, além de possuírem uma enorme perspectiva de crescimento, tanto na adoção de usuário e funcionalidades, como na adaptabilidade frente à tendencias como: *Big data*, *Internet of things* e pervasividade.

A grande vantagem dessa solução é ser flexível o bastante para se adaptar aos diversos cenários, públicos alvo e dispositívos de acesso. Dessa forma, pode-se responder às mudanças na sociedade e aproveitar as oportunidades que surgem.

3.1 Modelo de negócio

A decisão sobre qual modelo de negócios deve ser adotado costuma ser uma das mais dificeis durante o planejamento de um serviço, visto que é preciso equilibrar quantidade de clientes, custo, retorno, risco e oportunidade para otimizar o ganho finânceiro. A API e o NFP possuem três públicos alvos: Desenvolvedores, usuários finais e empresas, cada um com objetivos, recursos e necessidades diferentes. Para descobrir qual o melhor modelo de negócios vale a pena testar diferentes alternativas e avaliar os resultados obtidos. A seguir estão as principais formas de remuneração para os diversos públicos.

Desenvolvedores

- **Receita direta**: Cobrar diretamente o cliente pelo uso da API. Ex. planos mensais, semestrais ou anuais.

- **Freemium**: As funcionalidades básicas grátis, as avançadas pagas; Ex. inserção e consultas de NF-e oferecidas gratuitamente, porém os filtros (valor, cnpj ou data) pagos.
- **Limitada**: Uma quantidade pequena de consultas grátis. Ex. 100 consultas grátis por dia, a partir dessa quantidade será cobrado cada 1000.

Usuários finais

- **Receita direta**: Cobrar uma assinatura mensal do usuário pelo uso do NFP.
- **Freemium**: As funcionalidades básicas grátis, as avançadas pagas.

Empresas

- **Venda dos dados**: O valor pode depender da especificidade e da quantidade. O setor de televendas pode ter um grande interesse nesse tipo de dado, visto que, há informações sobre os clientes e seus hábitos de consumo.

É possível combinar vários desses modelos, a venda de dados para empresas parceiras pode ser informado no termo de uso do software para usuários finais, e oferecer um serviço mais caro para os desenvolvedores que desejam manter a privacidade das NF-e submetidas. Outra forma de se obter dinheiro é através da venda dos dados de preços dos produtos oferecidos, uma loja de aparelhos eletrônicos pode querer saber o preço praticado por um concorrente.

3.2 Implementação

Para desenvolver a API, o arcabouço proposto e o caso de uso foi utilizaram-se como base os modelos descrito no livro *"Getting MEAN with Mongo, Express, Angular, and Node"*, [Holmes 2015].

3.3 Arcabouço

A classe nfe é o ponto mais importante do arcabouço, os campos (ou propriedades) da nfe podem ser obrigatórios ou facultativos. Por exemplo o campo CPF do comprador, em NF-e é obrigatório na NFC-e facultativo. Essa abordagem permite que apenas uma classe abranja ambas NF-e e NFC-e. Todos os métodos que manipulam os dados da NF-e estão definidos de forma privada nesta classe, que contribui para facilitar o uso do arcabouço.

MongoDB persiste os dados em documentos ao invés de tabelas, de modo que a um documento pode conter um subdocumento, similar a um objeto JSON que contem outro objeto JSON. O documento nfe possui o campo infNfe, que agrupa os seguintes subdocumentos.

total: Armazena dados relativos aos impostos pagos

ide: Identifica o tipo de operação

emit: Armazena os dados do emissor da nota fiscal

det: Armazena os dados do produto ou serviço adquirido

dest: Armazena os dados do destinatário do produto

transp: Armazena dados relativos ao transporte da mercadoria

A Figura 3.1 apresenta o diagrama com os principais campos da classe nfe, todos os campos dos objetos são do tipo *String*, pois não há necessidade de se fazer cálculos com os valores numéricos presentes na NF. Devido ao fato de que o banco de dados armazena os dados de forma não relacional, não faz sentido destrinchar a classe nfe em uma composição de sub classes.

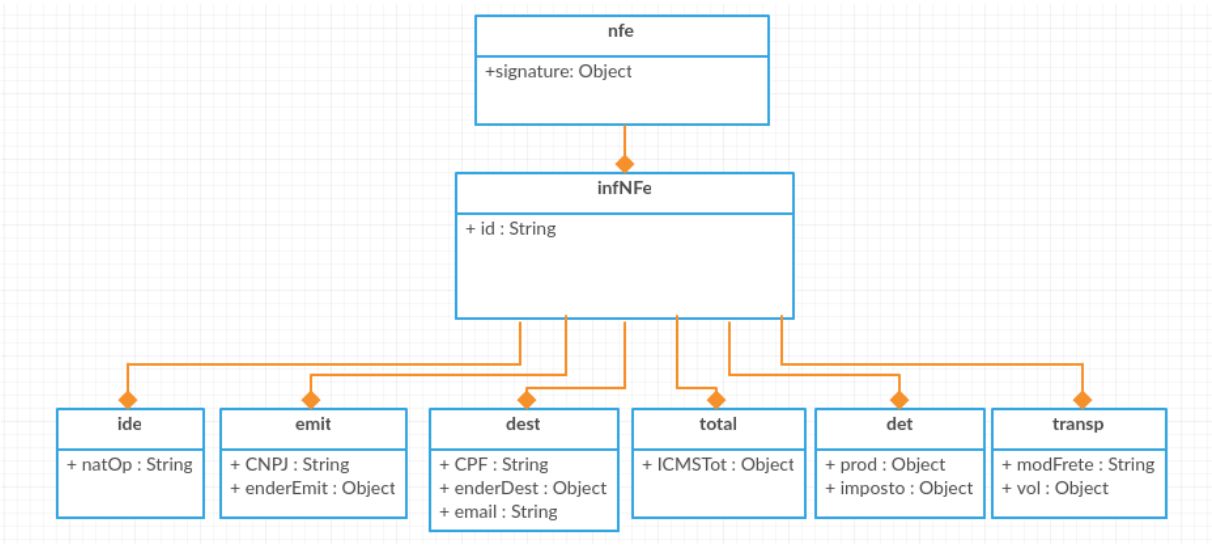

Figura 3.1: Diagrama de Classe da nfe.

3.4 API

A API foi desenvolvida baseada na arquitetura *Representational State Transfer (REST)*, de modo a ser capaz de realizar as operações básicas *Create, Read, Update, Delete (CRUD)*. A arquitetura REST é o ponto mais importante da API, pois esta é uma solução baseada em componentes acopláveis, de modo que possam ser substituídos ou aperfeiçoados conforme necessário.

3.4.1 CRUD

A API responde as requisições CRUD com um objeto JSON com o conteúdo requisitado ou o tipo de erro obtido. Os métodos aceitos pela API estão apresentados na Tabela 3.1.

URI	Método HTTP	Descrição
api/nfes	GET	Retorna a lista das NF-e
api/nfes	POST	Cria uma nova NF-e
api/nfes/:objectId	GET	Retorna a NF-e especificada
api/nfes/:objectId	PUT	Atualiza a NF-e especificada
api/nfes/:objectId	DELETE	Remove a NF-e especificada
api/profile	GET	Retorna o usuário logado
api/register	POST	Cria um novo usuário
api/login	POST	Cria um novo token de autenticação

Tabela 3.1: Método HTTP x CRUD

3.4.2 Diagrama de fluxo

O diagrama de fluxo a seguir apresenta as etapas realizadas para a submissão de uma NF-e. O código XML é inserido no campo adequado da aplicação cliente que o converte para JSON, a API esquematiza, de modo a ser persistir os valores desejados, o banco de dados retorna uma resposta para a API informando sobre o sucesso, e essa resposta é encaminhada para a aplicação cliente.

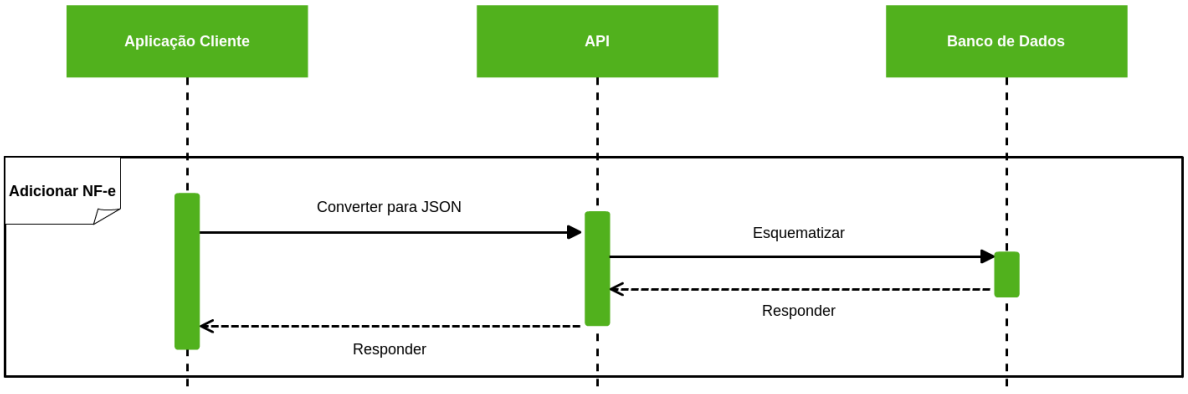

Figura 3.2: Diagrama de fluxo.

3.4.3 Respostas

Após processar o método da API, o servidor retorna um HTTP *response message* composto por um código de *status* e pelo corpo da mensagem que contem o objeto solicitado,

sendo necessário fazer um tratamento na aplicação cliente para cada um dos possíveis código retornados. O significado dos códigos são:

Código 2XX: operação realizada com sucesso, a aplicação segue seu fluxo normal.

Código 4XX: operação não realizada, a aplicação deve exibir uma mensagem de erro, informando ao usuário o motivo pelo qual o método não foi executado.

Código 5XX: erro interno do servidor, a aplicação informa ao usuário que algo inesperado ocorreu, e o lado cliente deve enviar um log com os dados da requisição para serem analisados pelo desenvolvedor.

3.5 Autenticação

Para garantir que os dados submetidos pelos usuários sejam vistos somente por estes, utilizou-se o *middleware* para nodeJS *Passport*. *Passport* modulariza diferentes estratégias de autenticação baseadas em dados locais ou em API de terceiros como *Facebook*, *Twitter* ou *OpenID*. Para o arcabouço, escolheu-se manter os dados de acesso no servidor.

3.5.1 Registro

- **Salt** é uma string de caracteres gerada aleatoriamente para cada novo usuário, e armazenada no banco de dados.
- **Senha** deve ser uma sequencia de caracteres escolhida pelo usuário, não é armazenada no banco de dados.
- **Hash** é gerada pela encriptação da concatenação do salt com a senha do usuário, é armazenado no banco de dados.

Durante ao registro do usuário, o sistema concatena a senha fornecida com o *salt*, aplica a criptografia e gera o *hash*, depois salva o *salt* e o *hash* no banco de dados.

3.5.2 Acesso

Durante o login, uma vez que a senha do usuário é confirmada como correta, o servidor gera e retorna um JSON Web Token (JWT), o qual é salvo no armazenamento local do navegador. O JWT possui uma data de expiração e deve ser submetido via *web service* sempre que uma operação exigir autenticação. A função de logout apenas apaga o conteúdo do JWT do armazenamento local.

O JWT é composto por três strings separadas por ponto, esses três segmentos são:

- **Header** é um objeto JSON contendo o tipo e o algoritmo de hash utilizado.
- **Payload** é um objeto JSON contendo o dado do token.

- **Signature** é o Header e o Payload encriptado por uma chave que só o servidor conhece [Auth0 Inc 2016].

3.5.3 Segurança

O JWT é ponto central da segurança, ele é que garante que o acesso à serviços restritos à usuários autenticados. Para garantir que a visualização e manipulação dos dados será realizada somente pelos usuários autorizados é preciso garantir que o JWT não possa ser lido ou gerado por um terceiro.

Criptografia

Os três contúdos do JWT são codificados utilizando o método Base64url, o qual é descrito em `https://tools.ietf.org/html/rfc4648` e gera um resultado formado por caracteres alfanumericos e os símbolos '_', '-' e '=' dessa forma o resultado pode ser enviado em ambientes HTTP sem afetar o endereçamento.

Cross-site scripting

Um usuário mal intencionado de posse do JWT, pode se passar pelo usuário autenticado mesmo sem saber sua senha. Uma forma de se obter o JWT é fazer com que um usuário legítimo execute um código que leia o conteúdo do armazenamento local e envie para outro servidor. Para evitar que esse tipo de ataque ocorra é necessário sinalizar os dados persistidos no sistema, para que nunca sejam executados como código javascript. O arcabouço AngularJS realiza essa proteção por padrão[1], todo texto exibido dentro de chaves duplas não é interpretado como HTML nem como uma declaração de função.

3.6 Instalação

Para usa a API, disponibilizou-se acesso a uma máquina virtual com o sistema operacional *Linux Ubuntu versão 14.04.03*, seu endereço público é `http://ec2-52-67-13-38.sa-east-1.compute.amazonaws.com` e seu ip `http://52.67.13.38`. O primeiro passo na configuração da máquina foi a instalação dos programas via comando apt-get:

- git versão 1.9.1
- nodejs 4.4.4
- npm 2.15.1
- mongodb

[1] https://docs.angularjs.org/guide/expression

Após a instalação do mongoDB é necessário criar a pasta */data/db* com permissão de escrita.

3.6.1 Dependências

A seguir estão as dependências do projeto, contidas no arquivo package.json.

```
"dependencies": {
    "body-parser": "~1.13.2",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "express": "~4.13.1",
    "jade": "~1.11.0",
    "mongoose": "^4.4.7",
    "morgan": "~1.6.1",
    "request": "^2.69.0",
    "serve-favicon": "~2.3.0",
    "xml2json": "^0.9.0"
}
```

4

Estudo de caso

Nesse capítulo, apresenta-se um estudo de caso de um sistema que utiliza a API/biblioteca abordada no capítulo anterior.

4.1 Nota Fiscal Pessoal

O sistema Nota Fiscal Pessoal (NFP) é um serviço de gerenciamento de NF-e, acessível através do endereço `http://52.67.13.38/`, que oferece ao usuário as funcionalidades de cadastro, edição, remoção e pesquisa de NF-e. O lado cliente da aplicação foi desenvolvido utilizando o arcabouço AngularJS e as bibliotecas bootstrap para responsibilidade e xml2json para a realização de *parses* entre os formatos XML e JSON. Durante o primeiro acesso, o arcabouço Express envia os arquivos necessários para exibir a página inicial, conforme o usuário acessa outros endereços, os arquivos e *scripts* vão sendo enviados.

4.1.1 Páginas

O cabeçalho do site pode ser exibido de duas formas, se o usuário não está autenticado é exibido como na Figura 4.1, se estiver autenticado como na Figura 4.2. Nas figuras, cada número está associado a um dos possíveis *links*, segue abaixo a função destes.

- **Início** - Leva à página inicial;
- **Entrar** - Leva à página de entrada;
- **Nome do usuário** - Leva à página de perfil do usuário;
- **Sair** - Termina a sessão do usuário e leva à página inicial.

Inicial

Nessa página, o usuário tem a opção de entrar ou se registrar.

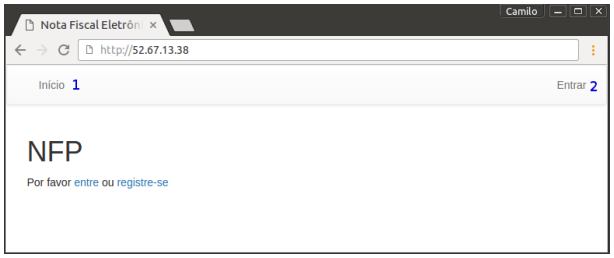

Figura 4.1: Cabeçalho de navegação sem autenticação do usuário.

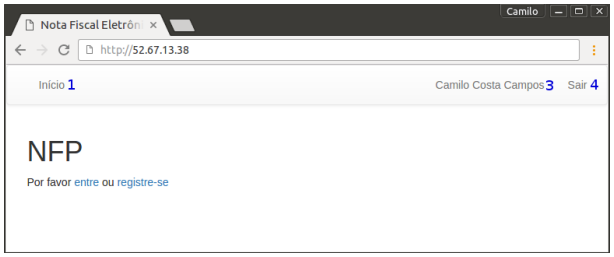

Figura 4.2: Cabeçalho de navegação com autenticação do usuário.

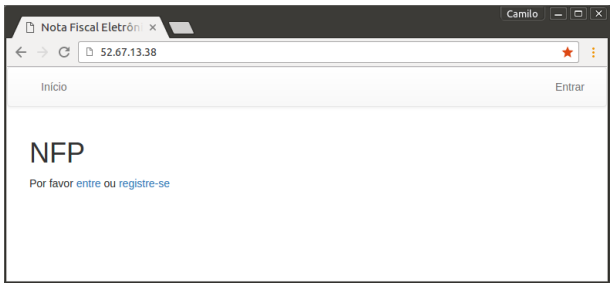

Figura 4.3: Página incial.

Cadastrar

Nessa página, o usuário pode se cadastrar, bastando inserir seu nome, email e senha.

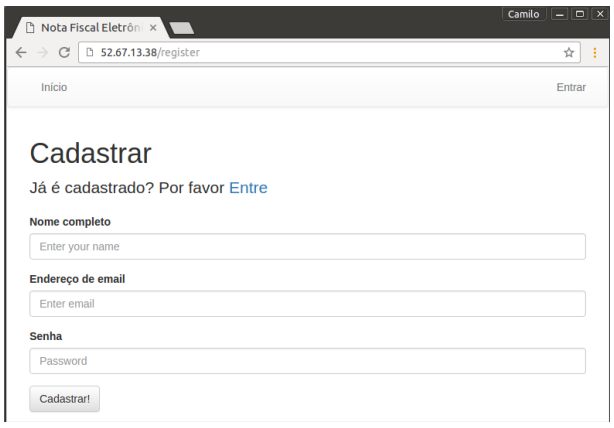

Figura 4.4: Página de cadastro de novo usuário.

Entrar

Nessa página, o usuário pode inserir seu endereço de email e senha cadastrados para entrar no sistema.

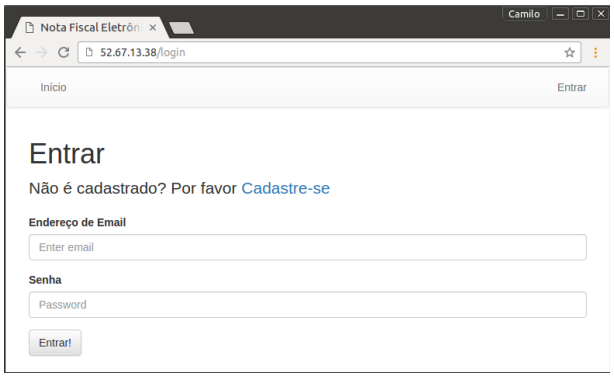

Figura 4.5: Página de entrada.

Perfil

Essa é a principal página do sistema, onde o usuário pode cadastrar, visualizar e manipular suas NF.

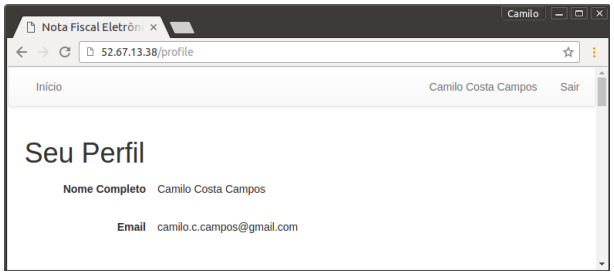

Figura 4.6: Topo da página do perfil do usuário.

4.1.2 Funcionalidades da página perfil

A primeira funcionalidade é um campo de texto que exibe todas as NF-e que o usuário cadastrou, seu conteúdo é obtido pelo método GET na URI /api/nfes.

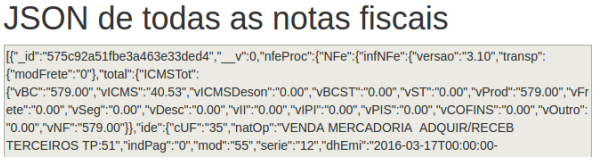

Figura 4.7: Campo de texto com todas as NF-e.

A segunda funcionalidade é composta por um *drop down menu*, dois botões e um campo de texto. Essa funcionalidade permite remover ou editar a nota fiscal especificada no *drop down menu*, através dos métodos DELETE e PUT da API.

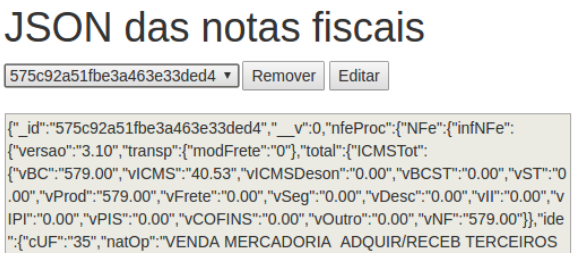

Figura 4.8: Funcionalidades remover e editar.

A terceira funcionalidade é relativa a submição de novas NF-e. O usuário pode escolher se deseja submeter a NF-e em formato json ou xml, bastando preencher o campo adequado e apertando o botão salvar.

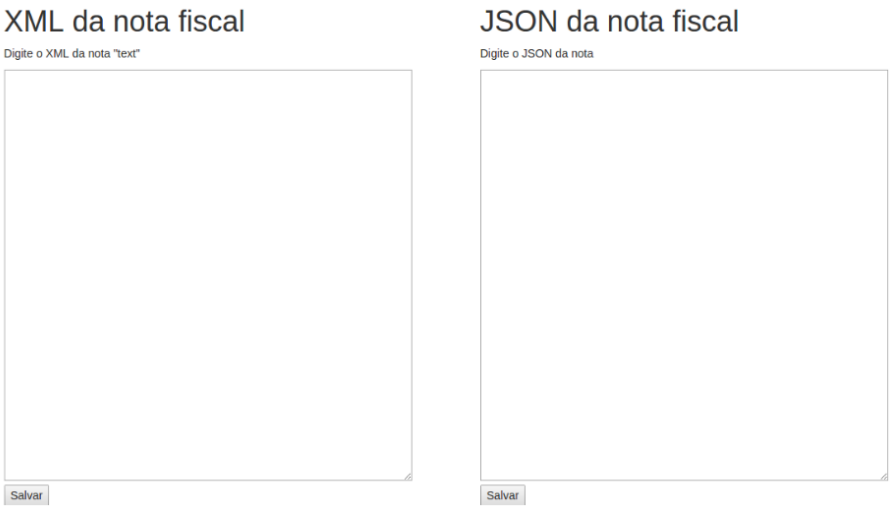

Figura 4.9: Campos de submissão de NF-e.

5

Conclusões e trabalhos futuros

Este capítulo, divide-se em duas seções que apresentam as conclusões do trabalho realizado e as propostas de trabalhos futuros.

5.1 Conclusões

Nesse trabalho foi apresentado, o projeto e implementação de uma API/biblioteca para todo o ciclo de gerenciamento de NF-e e uma aplicação cliente que utiliza essa API/biblioteca. Organizou-se um resumo sobre as tecnologias e arquiteturas utilizadas, as decisões tomadas e os passos necessários para a configuração e instalação da infraestrutura.

Os objetivos propostos neste documento foram atingidos, conseguiu-se desenvolver a API/biblioteca e uma aplicação cliente, feitos a partir da pilha de soluções *MEAN* e os hospedar em um servidor, tornando-o acessível para qualquer dispositivo, munido de um navegador, com acesso à Internet.

Esse trabalho representou uma grande oportunidade de aprendizado técnico, acadêmico e organizacional. Foi possível desenvolver do início ao fim um sistema útil, funcional e adaptável, baseado em ferramentas e paradigmas que são utilizadas mundialmente em diversos projetos bem sucedidos de todos os tamanhos.

O maior desafio durante o desenvolvimento foi devido a grande quantidade de arcabouços, bibliotecas e paradigmas, ainda que estes sejam fáceis de utilizar foi preciso dedicar um considerável tempo para entender cada um deles. Lidar com as leis que regem as notas fiscais também se mostrou um desafio, visto que o processo de digitalização das NF implica em grandes mudanças na sociedade. A digitalização e automação dos procedimentos, relativos às NF, impacta a sociedade ao liberar os trabalhadores das tarefas repetitivas. Pode-se utilizar a experiência adquirida sobre os problemas mais comuns e as excessões à regra, para desenvolver processos mas rápidos, eficientes e menos sucetíveis a erros. Com isso toda a sociedade é favorecida.

A API/biblioteca poderá ser utilizado por desenvolvedores em diversos projetos relacionados a NF-e, sua arquitetura simples e clara será um atrativo. Ao usuário final é

oferecido um serviço para solucionar os problemas de armazenamento e organização das suas NF-e. Esse sistema torna mais fácil usufruir da garantia em uma compra, auxilia a evitar a inclusão de informações incorretas ou incompletas em declaração de imposto de renda, e pode ainda permitir uma visão ampla dos gastos e assim aplicar um controle finânceiro.

5.2 Limitações

O presente trabalho detalha o que o sistema já faz, e as propostas do que o sistema fará em futuras implementações, porém é necessário definir as limitações que não serão incluidas no sistema.

O que não convem ao serviço fazer:

O sistema não será uma rede social, informações sobre a vida pessoal ou fotografias não serão armazenadas.

O que o serviço não é capaz de fazer:

O serviço não é capaz de obter dados de notas fiscais que não tenham sido inseridas diretamente pelo consumidor ou pelo fornecedor do bem ou serviço.

O que o serviço é capaz de fazer porém inviável:

É inviável implementar um mecanismo que vasculhe todos os dados inseridos em busca de uma fraude fiscal.

5.3 Trabalhos Futuros

As sugestões de trabalhos futuros organizam-se em dois grupos, os relacionados à API, à biblioteca e à aplicação NFP.

As funcionalidades da API seriam:

- Determinar a quantidade e paginação dos resultados;
- Limitar a quantidade de requisições que um cliente pode fazer ao servidor;
- Criptografar com *Secure Socket Layer (SSL)* a comunicação entre o cliente e a API;
- Filtrar, ordenar e pesquisar os resultados.

A mudança proposta na biblioteca é:

Modificar a classe nfe de modo que sua extensão ocorra de forma direta, conforme se vê em bibliotecas ou arcabouços maduros.

O estudo de caso apresentado possui poucas funcionalidades, planeja-se expandir essas funcionalidades ao adicionar:

- Leitura de *QR-Code*, o qual, futuramente estará presente na maioria dos cupons fiscais.

- Notificação por email de eventos ligados às NF-e, o fim do periodo de garantia de uma compra; A proximidade da data de vencimento de uma parcela a ser paga;

- Impressão de NF, é possível desenvolver um modelo gráfico de NF, o qual seria preenchido com os dados da NF-e selecionada, para ser impresso pelo usuário.

Integração com um software existente:

Os aplicativos de controle finânceiro pessoal são potenciais consumidores da API, essa integração pode evitaria ao usuário a necessidade de inserir diversos dados de uma compra no aplicativo, bastando adicionar apenas a NF-e. Entre os softwares de controle finânceiro disponíveis para essa integração destaca-se o Quanto Gastei[1].

É possível também utilizar adaptar a infraestrutura para outros contextos diferentes de NF, como passagens aéreas, comprovantes de pagamento, aluguel de equipamentos, etc.

[1] `https://www.quantogastei.com/`

Referências Bibliográficas

Angularjs.org. 2016. Acessado em 11/07/2016. Disponível em: <https://docs.angularjs.org/guide/introduction>.

Auth0 Inc. 2016. Acessado em 11/07/2016. Disponível em: <https://jwt.io/introduction/>.

ECMA International. 2016. Acessado em 11/07/2016. Disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000.

HOLMES, S. Getting mean with mongo, express, angular, and node. MANNING SHELTER ISLAND, Greenwich, CT, USA, 2015.

Mozila. 2016. Acessado em 11/07/2016. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

Node.js Foundation. 2016. Acessado em 11/07/2016. Disponível em: <https://nodejs.org>.

R. E. Johnson, B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, 1988.

S. Tilkov, S. Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, 2010.

StrongLoop. 2016. Acessado em 11/07/2016. Disponível em: <http://expressjs.com/>.

Xml2json authors. 2016. Acessado em 11/07/2016. Disponível em: <https://github.com/buglabs/node-xml2json>.