UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

KAROLINE KEIKO IKENAMI

PLATAFORMA PARA ANÁLISE DE GASTOS FINANCEIROS UTILIZANDO DATA VISUALIZATION

BAURU, SP

2018

KAROLINE KEIKO IKENAMI

PLATAFORMA PARA ANÁLISE DE GASTOS FINANCEIROS UTILIZANDO DATA VISUALIZATION

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para a obtenção do título de bacharel.

Orientador: Profa. Dra. Simone das G. Domingues Prado

BAURU, SP
2018

Ikenami, Karoline Keiko.
Plataforma para análise de gastos financeiros utilizando *Data Visualization* / Karoline Keiko Ikenami, 2018
54 f. : il.

Orientador: Simone das Graças Domingues Prado

Monografia (Graduação)–Universidade Estadual Paulista. Faculdade de Ciências, Bauru, 2018

1. Aplicação *Web*. 2. *Data Visualization*. 3. *Express*. I. Universidade Estadual Paulista. Faculdade de Ciências. II. Título.

KAROLINE KEIKO IKENAMI

PLATAFORMA PARA ANÁLISE DE GASTOS FINANCEIROS UTILIZANDO DATA VISUALIZATION

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para a obtenção do título de bacharel.

Aprovado em 14/11/2018

BANCA EXAMINADORA

Profa. Dra. Simone das G. Domingues Prado
Orientadora
Faculdade de Ciências
Universidade Estadual Paulista – Bauru

Profa. Dra. Márcia A. Zanoli Meira e Silva
Faculdade de Ciências
Universidade Estadual Paulista – Bauru

Prof. Dr. Renê Pegoraro
Faculdade de Ciências
Universidade Estadual Paulista – Bauru

AGRADECIMENTOS

Agradeço a minha família, especialmente aos meus pais que apresentaram ter imensa compreenção e paciência durante todos esses anos de graduação. Obrigada por acreditarem em mim e tornar possível este capítulo de minha vida.

Aos meus amigos que tive o prazer de conhecer durante minha graduação. Guardo com carinho os risos e lágrimas que compartilhamos nesta fase tão sensível e de profundas descobertas. O suporte de vocês para a realização deste trabalho foi indispensável.

Aos meus colegas de classe por me fazerem acreditar que apesar das tempestades e tormentas enfrentadas, estávamos todos no mesmo barco com rumo ao mesmo destino. Perdemos alguns homens ao longo da jornada, mas em nossas pequenas vitórias... no fundo todos estão conosco.

Agradeço ao professor Morgado por ter tornado o LTIA possível. Neste laboratório didático foi onde descobri o valor de se trabalhar em grupo e de passar o conhecimento adiante. Graças ao LTIA conheci pessoas incríveis e compartilhei muitas memórias, guloseimas e momentos de tensão com meus amigos e colegas.

Muito obrigada a todos os meus professores da Unesp por acreditarem em mim além do que eu mesma acreditava. Com meus professores aprendi diversas coisas sobre a vida e a minha carreira. Aprendi a questionar se eu estava fazendo a coisa certa, e se estava fazendo certo a coisa.

“Devemos exigir de cada um o que cada um pode fazer, – disse o rei. – A autoridade é baseada principalmente na razão.”
(O Pequeno Príncipe)

RESUMO

Os avanços tecnológicos das últimas décadas possibilitaram coletar um volume imenso de dados sobre os mais diversos temas. Com isso, o desejo de compreender esses dados e poder realizar uma análise fiél de forma simples e intuitiva também aumentou. A solução encontrada foi a de adotar técnicas de *Data Visualization*, na qual é possível transmitir uma mensagem por meio da visualização dos dados através de gráficos. Este trabalho tem como objetivo o desenvolvimento de uma aplicação *web* com o intuito de auxiliar as pessoas a realizarem a análise de seus gastos pessoais através da adoção de princípios de *Data Visualization*. Para executar este trabalho foi necessário realizar uma revisão bibliográfica, planejamento da arquitetura da aplicação, estudo de ferramentas a serem utilizadas e, por fim, sua implementação. Seu desenvolvimento está dividido em três partes: o *Front-End*, a *API* e o banco de dados. Ao final da implementação, optou-se por servir a aplicação *web* através da plataforma *Heroku*. Algumas das principais ferramentas e plataformas utilizadas foram: *Node.js*, *Express*, *Vue*, *Postman*, *Auth0* e *mLab*.

Palavras-Chave: *Data Visualization*, Aplicação *Web*, *Express*, *Vue*.

ABSTRACT

The technological advances of the past decades allowed us to collect a huge amount of data of all sorts. With it, the desire to understand the data and to be able to analyze it in a reliable and intuitive way has also increased. The solution that was found was to adopt Data Visualization techniques, in which it's possible transmit a message throught the visualisation of data using graphics and charts. The goal of this paper is to develop a web application in order to assist people to analyze their personal expenses through the adoption of Data Visualization principles. To perform this work, it was necessary to do a bibliographic review, plan the application architecture, study the tools that would be used, and, finally, the implementation. Its development is divided into three parts: the Front-End, the API and the database. At the end of the implementation, it was choosen to serve the web application through the Heroku platform. Some of the main tools and frameworks used were: Node.js, Express, Vue, Postman, Auth0 and mLab.

Keywords: Data Visualization, Web Application, Express, Vue.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A tecnologia permite reunir uma quantidade cada vez maior de dados e há um crescente desejo de extrair informações de todos eles. Ser capaz de visualizar dados e estabelecer uma narrativa com eles é a chave para transformá-los em informações que podem ser utilizadas para melhorar a tomada de decisões (KNAFLIC, 2015, p. 2).

Dados vão além de números, e para visualizá-los é preciso entender o que eles representam. O dado, assim como uma fotografia, captura um pequeno momento no tempo (YAU, 2013, l. 110). É uma simplificação do mundo, enquanto que a visualização é a abstração do dado. Contudo, não significa que a visualização ofusca a interpretação do dado. Ela, na verdade, pode ajudar a destacar dados pontuais e explorá-los de um ângulo diferente (YAU, 2013, l. 138). A conexão entre o dado e o que ele representa é a chave para uma visualização coerente, e se mostra fundamental para a análise cuidadosa e para uma compreensão aprofundada dos seus dados (YAU, 2013, l. 177).

Na hora de representar a informação de forma visual, é muito mais importante responder primeiro as seguintes questões: “Quem irá ver isso?”, “O que eles querem?”, “Qual mensagem ou ideia quero transmitir?”, “O que eu poderia mostrar?”, “O que eu deveria mostrar?”; e só então questionar: “Como eu vou mostrar isso?” (BERINATO, 2016, l. 156).

Isso se deve ao fato de que a história que se deseja passar pode variar dependendo da audiência a ser atingida. A ênfase, a chamada de ação, e até mesmo os dados mostrados (ou a decisão de mostrar os dados) seriam diferentes para cada público. Se fosse criada uma única mensagem destinada a atender a todas as necessidades desses públicos distintos, ela provavelmente não atenderia por completo às necessidades de qualquer um deles (KNAFLIC, 2015, p. 27).

Estes aspectos abordados fazem parte de *Data Visualization*. *Data Visualization* é a técnica utilizada para comunicar sua mensagem através do uso de representações visuais como eixos, linhas ou barras, fundamentados nos dados. Ela permite a visualização de grandes volumes de informações que de outra forma seriam impossíveis de analisar por meio de uma simples tabela (PARK, [20-?], l. 36).

Aplicativos e sites de gerenciamento de gastos financeiros muitas vezes falham em transmitir ao usuário de forma clara como seu dinheiro está sendo gasto. Visando este problema, o objetivo deste trabalho é de desenvolver uma plataforma *web* que auxilie seus usuários a entenderem seus gastos de forma simples e intuitiva através da implementação de técnicas de *Data Visualization*.

1.1 PROBLEMA

Plataformas de gerenciamento de gastos financeiros muitas vezes apresentam seus dados de forma não eficientes. Isso pode ocorrer pela falta de contexto na hora de representar um dado, escolha de gráficos inadequados, excesso de informações apresentadas ou informações redundantes, utilização de cores que podem causar confusão na hora de interpretar o gráfico, poluição visual gerada pelo excesso de cores ou informações, gráficos que não possuem uma mensagem definida a ser passada – são usados apenas como uma forma de visualizar os dados –, ou por gerar ambiguidade na hora de interpretar os dados apresentados.

Também é comum encontrar plataformas nas quais é necessário o usuário preencher ou editar os gastos individualmente para adicionar informações relevantes como, por exemplo, em qual categoria aquele gasto se enquadra. Um trabalho árduo e desgastante onde se investe muito tempo, este que poderia ser utilizado para focar em outras atividades mais produtivas.

Essas plataformas normalmente não possuem como público alvo um grupo específico, e tentam agradar diversos públicos ao mesmo tempo. Criam inúmeras funcionalidades para atender os mais diversos tipos de situações e, por consequência, tornam sua plataforma complexa e difícil de ser utilizada por novos usuários. Estes, que buscam uma solução simples e fácil para acompanhar seus gastos.

1.2 JUSTIFICATIVA

Nos últimos anos o aumento do volume de dados coletados trouxe consigo a necessidade de interpretar essas informações de uma forma clara. Normalmente,

para visualizar essas informações costumamos fazer uma representação por meio de tabelas ou por meio de gráficos. Enquanto que na representação em formato de tabelas é possível ler facilmente todos os detalhes de um determinado dado, os gráficos, por outro lado, possibilitam apresentar um volume grande de dados de rápida compreensão.

Para este projeto, utilizou-se a representação visual das informações através de gráficos. Mais especificamente, foi desenvolvida uma plataforma que representa as informações a fim de facilitar a compreensão de como o dinheiro do usuário foi gasto ao longo do tempo. No desenvolvimento da aplicação *web* foi posto em prova os conhecimentos adquiridos ao longo da graduação tanto em sua programação, quanto a arquitetura, banco de dados e comunicação do sistema.

Foram aplicadas técnicas de *Data Visualization*, promovendo seus pontos principais. Dentre eles a caracterização do público-alvo, a importância de se definir uma mensagem ou propósito a ser transmitido a partir do dado, e compartilhar boas práticas sobre a escolha da melhor representação visual na hora de se transmitir determinada mensagem.

O projeto pode ser facilmente escalado tanto em relação ao público a ser atingido, quanto em relação a análise e representação dos dados a serem consumidos. Sobre o público, é possível expandir para outras formas de coletar os gastos financeiros - seja importando extratos bancários, ou fazendo integrações com os bancos para a coleta automática de dados. Sobre a análise e representação dos dados, também seria possível implementar técnicas de Inteligência Artificial para reconhecimento de padrões e fazer suposições embasadas sobre os gastos realizados.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

Desenvolver uma aplicação *web* para auxiliar pessoas a entenderem melhor seus gastos, utilizando os princípios de *Data Visualization* para a representação visual dos dados.

1.3.2 OBJETIVOS ESPECÍFICOS

- Aplicar o conceito de *Data Visualization*;
- Identificar possíveis informações relevantes sobre os gastos dos usuários;
- Definir mensagens a serem transmitidas através de gráficos;
- Utilizar ferramentas de desenvolvimento de aplicações *web*;
- Definir arquitetura da aplicação *web*;
- Implementar a aplicação *web* utilizando tecnologias atuais;
- Testar e verificar integridade da plataforma;
- Conferir integridade dos dados apresentados;

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados alguns conceitos utilizados ao longo do desenvolvimento deste trabalho. São eles: *Data Visualization*, Computação em Nuvem e Banco de Dados.

2.1 DATA VISUALIZATION

Data Visualization é um termo amplo para descrever qualquer método utilizado para auxiliar pessoas a entender o significado de um dado através de uma representação visual. Através do uso de gráficos, mapas, entre outras representações, *Data Visualization* proporciona uma forma mais simples e intuitiva de identificar padrões, anomalidades, similaridades, e de fazer interpretações dos dados (Tableau, 2018), quando comparado a uma representação por meio de uma tabela.

2.2.1 TIPOS DE VISUALIZAÇÕES

Segundo Knaflic (2015), o universo de representações dos dados pode ser bem vasto, mas com apenas um pequeno grupo deles já é possível cobrir grande parte das situações. São elas:

2.2.1.1 Texto simples

Utilizado quando se tem apenas um ou dois números que se deseja representar. Desta forma, dá-se destaque aos números e adiciona-se poucas palavras para dar suporte ao contexto daquele dado a fim de passar sua mensagem claramente.

2.2.1.2 Tabelas

As tabelas são utilizadas para representar um conjunto de dados a uma ampla audiência na qual seus membros irão procurar por uma linha específica de seu interesse. Também é mais indicada para comunicar diferentes unidades de medidas. Não é indicado utilizar tabelas em apresentações ao vivo pois sua audiência perderá sua atenção ao tentar ler todas as linhas da tabela.

Evite que bordas marcantes e sombreados disputem a atenção do leitor com relação aos dados presentes na tabela. Ao invés disso, recomenda-se utilizar bordas de cores claras ou simplesmente espaçamentos para separar os elementos de uma tabela.

2.2.1.3 Gráficos

Enquanto as tabelas interagem com o sistema verbal das pessoas, gráficos interagem com o sistema visual, tornando mais rápido o processamento de informações. Existem diversos tipos de gráficos, a seguir estão os principais tipos que solucionam a maior parte dos casos de visualização de dados.

2.2.1.4 Gráficos de pontos

Os gráficos de pontos podem ser úteis para mostrar a relação entre duas situações porque permitem posicionar o dado no eixo X e no eixo Y, simultaneamente, para que se verifique se existe uma relação entre as duas métricas e que tipo de relação seria. Este tipo de gráfico costuma ser frequentemente utilizado em ramos científicos. É possível adicionar ainda mais métricas para a comparação de relacionamento, como o tamanho dos pontos, legendas e cores. Porém, é preciso tomar cuidado para que o gráfico não contenha informações demais a ponto de ser confuso na hora de transmitir uma mensagem.

2.2.1.5 Gráficos de linhas

São mais comuns para representar informações contínuas como as métricas de tempo (dias, meses, anos, entre outros). Como os pontos estão fisicamente interligados, isso implica um relacionamento de continuidade entre esses dados.

2.2.1.6 Gráficos de barras

São fáceis de comparar qual categoria é a maior entre todas as outras pois os nossos olhos tendem a focar no ponto onde as barras terminam. É importante que os gráficos de barras sempre tenham a base zero, caso contrário pode-se criar uma falsa comparação visual. De forma geral, recomenda-se que as barras do gráfico tenham uma largura maior do que o espaço em branco que as separa, porém não tão largas a ponto de fazer o leitor comparar áreas ao invés de comprimentos.

Os gráficos de barras horizontais são extremamente fáceis de ler. Eles são especialmente úteis quando os nomes das categorias são longos. Devido a forma como a maior parte das pessoas leem, este gráfico é vantajoso pois o leitor processa a informação da esquerda para a direita, e de cima para baixo. Caso os dados não possuam uma ordem lógica definida (ex: dias da semana, grupos por idades - 0-10 anos, 11-20 anos, etc), é aconselhado ordenar os dados do maior valor para o menor, ou vice-versa, dependendo do contexto apresentado.

2.2.1.7 Gráficos de área

Como os olhos humanos não fazem um bom trabalho ao atribuir valores quantitativos para uma área bidimensional, recomenda-se evitar este tipo de gráfico, a não ser quando deseja-se visualizar números de diferentes magnitudes.

2.2.1.8 Outros tipos de gráficos

Na hora de selecionar um gráfico, primeiro é preciso escolher um que permita transmitir de forma mais clara possível a mensagem a ser passada para a audiência.

Ao utilizar gráficos com visuais não tão familiares é preciso tomar um cuidado extra para torná-los acessíveis e fáceis de compreender ao público destinado.

2.2.1.9 Evite se possível

Gráficos de pizza, apesar de serem visualmente atraentes, eles tornam difícil de comparar partes que possuem um valor similar e distinguir qual parte possui um valor maior ou menor do que a outra (Knaflic, 2015). Um tipo de gráfico similar é o gráfico de rosca, no qual ao invés de se comparar as áreas de cada parte como no gráfico de pizza, compara-se os arcos. Ambos os casos fazem com que os dados representados acabem não sendo tão precisos para o observador, por isso recomenda-se evitá-los na maior parte dos casos.

Outra regra, segundo Knaflic (2015), é jamais utilizar um gráfico 3D. A única exceção é se os dados são representados em 3 dimensões. Gráficos 3D criam várias distrações desnecessárias (como painéis laterais e de base), além de dificultar a visualização exata de um dado devido a distorção causada pelo ângulo ao qual o gráfico é submetido.

2.2 COMPUTAÇÃO EM NUVEM

Computação em Nuvem é um modelo que permite acesso a diversos elementos computacionais configuráveis (como redes, servidores, aplicações, serviços, e armazenamento) que podem ser rapidamente provisionados e lançados com o mínimo esforço gerencial ou interação com o provedor de serviços (Mell; Grace, 2011).

Existem múltiplos serviços de Computação em Nuvem, também conhecidos como *XaaS*; um acrônimo para "*X as a Service*" onde o *X* é substituído pelo serviço oferecido. Alguns exemplos são *Software as a Service* (*SaaS*), *Platform as a Service* (*PaaS*), *Function as a Service* (*FaaS*), *Infrastructure as a Service* (*IaaS*), *DataBase as a Service* (*DBaaS*) e *Authentication as a Service* (*AaaS*), explicados a seguir.

2.2.1 SOFTWARE AS A SERVICE

Software as a Service, ou *SaaS*, é um modelo no qual ao invés de comprar uma licença e instalar o *software* individualmente em cada máquina, realiza-se uma assinatura para ter acesso à aplicação oferecida por uma empresa através da internet. Isso faz com que o cliente tenha mais flexibilidade e não precise se preocupar em comprar versões mais recentes do *software*, ou com a manutenção do mesmo (Dubey; Wagle, 2007). Alguns exemplos mais comuns de *SaaS* são: *Netflix*, *Spotify*, serviços de *e-mail* como *Gmail* ou *Outlook*, *Dropbox* e *Slack*.

2.2.2 PLATFORM AS A SERVICE

Neste modelo, empresas oferecem um ambiente online para que seus clientes possam rodar suas aplicações sem se preocupar com a infraestrutura, *softwares*, ou com o *hardware* necessário para provisionar sua aplicação (Rouse, c2018). Isso faz com que o processo de implementar, executar e escalar uma aplicação seja bem mais simples. Porém, alguns *PaaS* possuem restrições para suporte de linguagens, *frameworks* e implementações (Choi; Varma, 2016). Alguns exemplos de *PaaS* são: *AWS Elastic Beanstalk*, *Google App Engine*, *Windows Azure*, *OpenShift* e *Heroku*.

2.2.3 FUNCTION AS A SERVICE

Function as a Service é um modelo que permite os desenvolvedores executarem trechos de código independentes em resposta a eventos, sem se preocuparem em construir ou manter uma infraestrutura complexa para isso (Han, 2017). *FaaS* pode ser visto como uma parte do que constitui a Arquitetura *Serverless*, na qual se abstrai o gerenciamento de servidores e de infraestruturas de baixo nível na hora de tomar decisões de desenvolvimento (Johnston, 2018). Alguns exemplos de *FaaS* são: *AWS Lambda*, *Google Cloud Functions*, *IBM OpenWhisk* e *Microsoft Azure Functions*.

2.2.4 INFRASTRUCTURE AS A SERVICE

Neste modelo o provedor *Cloud* oferece os componentes de infraestrutura como *data centers*, servidores, *hardwares* de armazenamento, *hardware* de redes, assim como a camada de virtualização ou *hypervisor*. Também é oferecido vários serviços além dos componentes de infraestrutura, como os custos detalhados, monitoramento, registros de acesso, segurança, balanceamento de carga e clusterização, assim como *backup* de dados, réplicas e recuperação de estados (Rouse, c2018). Alguns exemplos de *IaaS* são: *Amazon Web Services* (*AWS*) e *Google Cloud Platfrm* (*GCP*).

2.2.5 DATABASE AS A SERVICE

Oferece um ambiente onde é possível gerenciar e realizar operações no Banco de Dados de forma simplificada, e com a flexibilidade e agilidade de escolher entre diversos tipos de Bancos de Dados já pré-configurados. De forma geral, *DBaaS* facilita a resolução de problemas, correção de erros e a transferência de dados de um sistema para outro. Com ele é possível escalar o Banco de Dados conforme a carga aumenta, sendo compatível com modelos de negócios nos quais o usuário precisa de uma maior segurança e disponibilidade (The Atlas Team, 2017). Alguns exemplos de *DBaaS* são: *Amazon Relational Database Service* (*RDS*), *Amazon DynamoDB*, *MongoDB Atlas* e *mLab*.

2.2.6 AUTHENTICATION AS A SERVICE

Authentication as a Service é um modelo no qual se oferece serviços de autenticação e gerenciamento de usuários para aplicações. Nestes serviços é possível encontrar páginas de *login* configuráveis, funções de *logout*, banco de dados de usuários, integrações com redes sociais, entre outras funcionalidades (Okawa, 2018). Alguns exemplos de *AaaS* são: *Auth0* e *Amazon Cognito.*

2.3 BANCO DE DADOS

Nesta seção serão apresentados dois tipos de Banco de Dados: o Banco de Dados Relacional e o Banco de Dados Não Relacional.

2.3.1 BANCO DE DADOS RELACIONAL

Bancos relacionais são constituídos por tabelas com relacionamentos pré-definidos entre si. Essas tabelas são compostas de linhas e colunas, nas quais as colunas representam os atributos presentes nos itens daquela tabela, e cada linha representa um item inserido na tabela. Cada item de uma tabela pode ter um atributo de identificação principal como a chave única. Para relacionar um item de uma tabela com o de outras tabelas utiliza-se chaves estrangeiras (AWS Amazon, c2018).

Para este tipo de Banco de Dados, *SQL* é a linguagem padrão utilizada para se realizar consultas e interagir com os dados presentes no banco. Por permitir que os usuários categorizem seus dados, isso torna mais fácil, no futuro, realizar consultas e filtrar dados específicos, além de tornar mais simples o ato de adicionar outras categorias de dados (Rouse, c2018).

Alguns exemplos de banco de dados relacionais são: *MySQL*, *Amazon RDS*, *Microsoft SQL Server*, entre outros.

2.3.2 BANCO DE DADOS NÃO RELACIONAL

Enquanto o Banco de Dados Relacional não foi feito para acompanhar os desafios de agilidade e de escala que as aplicações modernas enfrentam, os Banco de Dados Não Relacionais - também conhecidos como *NoSQL* - possuem maior escalabilidade e uma performance superior (MongoDB, c2018). Neste caso os dados podem ser guardados de diversas formas, entre elas: *key-value*, documentos, ou em formato de grafos (Rouse, c2018).

Banco de Dados Relacional requer que a modelagem seja feita antes de se adicionar os dados. Por isso pode não ser uma boa solução para desenvolvimentos ágeis nos quais o escopo e a estrutura do projeto estão em constante

transformação. Já o banco de dados não relacional permite a adição de dados sem ter um modelo pré-definido, o que torna mais simples fazer alterações significantes na aplicação em tempo real (Rouse, c2018).

Alguns exemplos de banco de dados não relacionais são: *MongoDB*, *DynamoDB*, *Cassandra*, *Redis*, *Elasticsearch*, entre outros.

2.4 UTILIZAÇÃO

Os conceitos abordados neste capítulo são utilizados ao longo do desenvolvimento da aplicação *web* deste trabalho. Alguns exemplos são o uso das plataformas *Auth0*, *Heroku* e *mLab*, que são, respectivamente, um *AaaS*, *PaaS* e *DBaaS*. Neste trabalho também se utiliza o Banco de Dados Não Relacional, pela sua versatilidade e simplicidade. Já os conceitos de *Data Visualization* são utilizados nas escolhas e composições dos gráficos presentes na aplicação *web*.

Nos próximos capítulos serão abordados com mais detalhes os materiais e as ferramentas utilizadas para o desenvolvimento, assim como o processo de desenvolvimento da aplicação.

3 MATERIAIS E MÉTODOS

Neste capítulo serão apresentados alguns dos materiais utilizados no desenvolvimento do projeto e a metodologia abordada.

3.1 MATERIAL

Nesta seção estão algumas das principais ferramentas e bibliotecas utilizadas para desenvolver a aplicação *web*.

3.1.1 VUE.JS

Vue é um *progressive framework* utilizado para desenvolver interfaces gráficas para os usuários. Isso significa que ele pode ser facilmente adotado em uma pequena parte de um projeto já existente, como também tem recursos suficientes para que se possa desenvolver um projeto do início utilizando *Vue* e outros componentes complementares. Ele também pode ser utilizado para desenvolver *Single Page Applications* (SPA).

Esta ferramenta fornece um modelo lógico para organizar os recursos de uma aplicação *web*. *Vue* permite dividir uma página *web* existente em vários componentes reutilizáveis, sendo cada um deles constituídos de sua própria estrutura *HTML*, *CSS* e *JavaScript*, necessários para serem renderizados individualmente (Vue.js, c2018). Algumas ferramentas populares e similares ao *Vue* são *Angular* - previamente conhecido como *AngularJS* - e *React*.

Utiliza-se *Vue* no projeto devido a sua versatilidade e pela facilidade de trabalhar com esta ferramenta. *Vue* possui uma documentação extensa e detalhada, tornando mais simples de entender os conceitos necessários para a resolução de determinados desafios encontrados ao longo do desenvolvimento. Detalhes sobre como esta ferramenta foi utilizada no projeto será abordado na seção de Desenvolvimento.

3.1.2 BOOTSTRAP

Bootstrap é um framework *open source* feito para auxiliar o desenvolvimento *front-end* de aplicações *web*. Ele fornece uma biblioteca com diversos componentes pré-construídos, grades responsivas, ferramentas para trabalhar com *SASS* e componentes que utilizam *jQuery* (Bootstrap, c2018).

Em sua fase inicial, em 2010, *Bootstrap* era originalmente conhecido como *Twitter Blueprint*. Ele foi desenvolvido por programadores da empresa *Twitter* com o intuito de servir como um guia para o desenvolvimento de projetos e ferramentas internas. Após seu lançamento para o público em 2011, desenvolvedores de todos os níveis passaram a adotar a ferramenta para seus projetos, tornando-o hoje um dos *frameworks* para *front-end* e *open source* mais populares do mundo (Bootstrap, c2018).

Com o *Vue* sendo responsável pela estrutura e componentização da parte visual deste projeto, *Bootstrap* foi escolhido para a estilização dos componentes presentes nas páginas. Por ter uma vasta biblioteca com recursos prontos e exemplos de como utilizá-los, torna-se mais fácil de adotar a tecnologia ao projeto, dispensando a necessidade de criar os estilos individualmente para cada componente utilizado. Além da divisão da tela em grades responsivas que tornam muito mais simples a composição do *layout* das páginas, independente de sua resolução e tamanho.

3.1.3 FUSIONCHARTS

FusionCharts é uma empresa indiana que fornece uma biblioteca *JavaScript* para construção de gráficos direcionada a desenvolvedores. *FusionCharts* foi lançado em 2002 como sendo uma solução privada e foi crescendo gradualmente ao passar dos anos. Em 2007 lançou-se a versão gratuita, e em 2009 tornou-se uma solução *open source*. Na metade de 2018, *FusionCharts* foi eleita pela Forbes como uma das *top* 7 ferramentas de *Data Visualization* em todo o mundo (FusionCharts, c2018).

Uma das vantagens de se utilizar *FusionCharts* é de possuir soluções e integrações prontas para diversas linguagens, como: *JavaScript*, *AngularJS*, *Angular*, *Vue*, *React*, *jQuery*, *PHP*, *Ruby on Rails*, *C#*, *Visual Basic* e *Python*. Em

seu *website* é possível visualizar diversos exemplos de gráficos, assim como o código necessário para compor o mesmo. Possuem uma documentação detalhada sobre todos os atributos que podem estar presentes em um determinado estilo de gráfico, assim como quais os valores que aquele atributo pode receber.

Comparando esta ferramenta com similares como *Chart.js*, *HighCharts*, entre outros, ela apresenta integração direta com *Vue*, assim como uma estética agradável dos gráficos apresentados e uma vasta biblioteca com exemplos de códigos em tempo real nos quais é possível alterar o código dos gráficos *online* a fim de obter o resultado desejado, sem complicações. Além de que a versão gratuita não possui restrições de funções ou de tempo, com o único detalhe de que esta versão inclui um link para o site da empresa em todos os seus gráficos. Como essa ferramenta está sendo utilizada para um projeto acadêmico, não é obrigatório comprar a licença desta biblioteca.

3.1.4 NODE.JS E EXPRESS

Introduzido em 2009 por Ryan Dahl, *Node.js* é um interpretador de código *JavaScript open source* utilizado no lado dos servidores. Diferente de outras plataformas similares como *Apache HTTP Server*, ou *Ruby on Rails*, entre outros, que escalam utilizando *threads*, *Node.js* roda um único *thread* de eventos em *loop* (Heller, 2017).

Sua alta performance se dá pelo fato de que ele recebe todas as requisições e delega grande parte do trabalho para outros *workers* do sistema que rodam no *background*. Quando estes workers terminam seu trabalho, eles emitem eventos *callback* para o Node.js. Essa solução o torna bem mais rápido do que sistemas *multi-threaded* para programas que não consistem extensamente de uma computação numérica, aritmética ou lógica; estas que são mais pesadas e acabam dependendo de um maior nível de processamento. Node.js não é muito indicado para operações que consomem a CPU, pois podem sobrecarregar a sua única e principal thread (Ofoegbu, 2018).

Em 2010 Isaac Schluter lançou *NPM*, um gerenciador de pacotes Node (Heller, 2017), sendo hoje sua instalação considerada parte do processo de instalação do *Node.js*.

Express é um dos pacotes *NPM* utilizados neste projeto, e um dos mais populares para o desenvolvimento de aplicações *web* com Node.js. Algumas de suas principais características são: foco em alta performance, alta cobertura de testes, assistentes HTTP (em questões de redirecionamento, *caching*, entre outros) e robusto processo de roteamento (Express, c2018).

3.1.5 AUTH0

Fundada em 2013, *Auth0* providencia uma plataforma universal para o gerenciamento de identidades para aplicações *web*, *mobile*, *IoT* (*Internet* das Coisas), e aplicações internas (Auth0, c2018). *Auth0* já arrecadou mais de 100 milhões de dólares em investimentos, sendo sua rodada mais recente a de Série D em maio de 2018 na qual arrecadou 55 milhões de dólares através de 6 investidores (Crunchbase, c2018). Para este projeto escolheu-se Auth0 para ser o gerenciador de usuários e intermediário de autenticação.

3.1.6 HEROKU

Heroku é um *PaaS* (*Plataform as a Service*) baseado em containers, no qual desenvolvedores podem servir, gerenciar e escalar suas aplicações. Fundada em 2007 por Orion Henry, James Lindenbaum, e Adam Wiggins, foi adquirida pela *Salesforce* em 2011. Desde então Heroku faz parte da *Salesforce Platform* (Heroku, c2018).

Alguns dos *frameworks* e linguagens aceitas nesta plataforma incluem: *Node.js*, *Ruby*, *PHP*, *Java*, *Python*, *Go*, *Scala* e *Clojure* (Heroku, c2018). *Heroku* foi escolhido para servir tanto o *Front-End* quanto o *Back-End* deste projeto, cada parte sendo uma aplicação distinta.

3.1.7 MLAB

O *mLab* é um dos principais *DBaaS* (*DataBase as a Service*) para o banco de dados *MongoDB* (mLab, c2018). Lançada em 2011, foi uma das primeiras empresas a oferecer um produto *DBaaS*, fazendo com que seus clientes não precisassem se

preocupar com o provisionamento, escala, *backup*, monitoramento e o gerenciamento do seu banco de dados (Shulman, 2018).

No início de outubro de 2018, mLab anunciou o início de sua aquisição pela *MongoDB, Inc*. Espera-se que esta aquisição seja concluída em 31 de janeiro de 2019 (MongoDB, 2018). Com isso, a empresa irá auxiliar seus clientes a migrarem seu banco de dados para a plataforma *MongoDB Atlas*, que oferece uma solução similar ao *mLab* e é um produto oferecido pela *MongoDB, Inc.*

O *mLab* é utilizado para hostear o banco de dados deste projeto pois, com poucos cliques após a criação de conta, já é possível criar um banco e obter as informações necessárias para realizar operações como: salvar, remover e editar dados.

3.1.8 PAPAPARSE

PapaParse é uma biblioteca feita em *JavaScript* para converter arquivos *CSV*. Iniciado em 2013 por Matt Holt, PapaParse é um projeto *open source* e atualmente está na versão 4.6.1, atualizada no início de outubro de 2018 (PapaParse, c2018). Esta biblioteca possui suporte para *Node.js* e pode ser instalada utilizando *NPM*, ou simplesmente baixando um arquivo com sua versão simplificada.

Com essa ferramenta é possível converter *strings*, arquivos locais e arquivos remotos no formato *CSV* para *JSON*, ou também, converter objetos *JSON* para o formato *CSV*. PapaParse é utilizado neste projeto para a importação das faturas do cartão de crédito em formato *CSV*, de forma que seu conteúdo pudesse ser salvo no banco de dados.

3.1.9 GITKRAKEN

Iniciada em 2014 por dois desenvolvedores que estavam insatisfeitos com a qualidade das interfaces gráficas para o controle de versão *Git*, logo ganhou suporte da empresa *Axosoft*. Em maio de 2016 sua primeira versão foi lançada ao público, em julho do mesmo ano lançou a versão *Pro*, e em julho do ano seguinte a versão para empresas. Atualmente, a plataforma possui mais de 1 milhão de usuários, número atingido em agosto de 2018 (GitKraken, c2018).

GitKraken oferece ferramentas para controle de versão utilizando *Git* em uma interface gráfica intuitiva, eficiente e de alta performance. Com soluções para sistemas *Mac*, *Windows* e *Linux*, o *GitKraken* também possui integrações com *GitHub*, *GitLab* e *BitBucket*, tornando mais simples a experiência para trabalhar com alguns dos principais repositórios remotos. *GitKraken* é utilizado no projeto pois, diferente de similares como *GitHub Desktop*, ou *SourceTree* ele oferece uma forma simples, completa e consistente para executar as tarefas necessárias para o projeto.

3.1.10 POSTMAN

Postman é uma plataforma com o intuito de auxiliar o desenvolvimento de *APIs*. Entre suas principais funcionalidades incluem: *Mock Servers*, Monitoramento, *Postman API*, integrações e *Newman*. *Mock Servers* é uma solução para que o desenvolvimento da interface gráfica não dependa do desenvolvimento do *Back-End* da aplicação, pois permite configurar servidores para que simulem as devidas respostas do *Back-End* enquanto o mesmo está sendo construído. A funcionalidade de monitoramento permite configurar o *Postman* de modo que execute um conjunto de chamadas a uma determinada *API* a fim de analisar suas respostas, entre outros dados. A *API* do *Postman* permite que dados das contas dos usuários sejam acessadas sem depender da aplicação. As integrações servem para complementar a aplicação e facilitar a transição de dados com outras plataformas disponíveis no mercado. Já o *Newman* permite utilizar o *Postman* na linha de comando (Postman, c2018)

Em Outubro de 2012 a aplicação *Postman* foi criada como um projeto secundário que rapidamente cresceu, tornando-se hoje uma das aplicações mais populares com mais de 5 milhões de desenvolvedores utilizando sua ferramenta. Em 2014 a empresa foi inaugurada e o projeto tomou foco principal, dando início ao desenvolvimento de novas features para a plataforma. Em 2016 lançou-se a versão *Pro*, e em 2018 foi lançada a versão para as empresas (Postman, c2018).

Esta plataforma é utilizada durante o desenvolvimento da *API* deste projeto. Também se utiliza o *Postman* para compreender e testar a API de outras ferramentas utilizadas, como o *Auth0*, por exemplo.

3.2 METODOLOGIA

A primeira fase deste projeto consistiu no estudo, compreensão e levantamento bibliográfico sobre *Data Visualization* através da realização de cursos *online*, pesquisa e leitura de: livros, artigos científicos e teses relevantes para o tema. Nesta fase também foi realizada uma pesquisa sobre as tecnologias a serem utilizadas no desenvolvimento da aplicação *web*. Estas tecnologias serão definidas abaixo, durante a descrição de como será a estrutura da aplicação.

Foi necessário caracterizar o público alvo da aplicação para assim definir qual tipo de abordagem seria utilizada na transmissão da informação desejada através dos gráficos. Busca-se uma experiência direcionada e personalizada para o usuário.

A plataforma está dividida em três partes: a plataforma *web*, a *API* e o banco de dados. Para a plataforma *web* é utilizado: o *framework Vue.js* (Vue.js, c2018) para o desenvolvimento da interface; *Bootstrap* (Bootstrap, c2018) para o layout e estilização de componentes; *FusionCharts* para a visualização de gráficos; a plataforma *Auth0* (Auth0, c2018) para realizar a autenticação do usuário; e *Node.js* (Node.js, c2018) com *Express* (Express, c2018) para servir a aplicação. A *API REST* (REST API Tutorial, c2018) foi desenvolvida utilizando *Node.js* com *Express* que serve de intermédio entre a plataforma de interface do usuário e o banco de dados, realizando o tratamento de requisições. Os dados enviados entre os sistemas são objetos no formato *JSON* (JSON, c2018). Tanto a plataforma *web* quanto a *API* foram hospedados no *Heroku* (Heroku, c2018). Para o banco de dados utilizou-se o *MongoDB* (MongoDB, 2018), hospedado no *mLab* (mLab, c2018). Para o consumo do arquivo *CSV* (arquivo da fatura do cartão de crédito) utilizou-se a biblioteca *Papa Parse* (Papa Parse, c2018), que permite transformar o conteúdo do arquivo *CSV* em um objeto *JSON*.

A segunda fase foi a de planejamento. Nela foi feita: a escolha final de ferramentas para o desenvolvimento da aplicação; definição da arquitetura do sistema – detalhar quais seriam as requisições feitas pela plataforma de interface à *API*, seus métodos, *URL* de acesso e quais os atributos dos objetos a serem passados em cada caso, além da estrutura do objeto a ser enviado como resposta; estruturação do banco de dados, definir quais as coleções e os modelos de documentos a serem salvos a partir da análise e caracterização dos dados

apresentados na fatura de cartão de crédito do *Nubank* (Nubank, c2018); definição das telas da plataforma, tema, escolha das cores, escolha dos gráficos e escolha das nomenclaturas utilizadas.

Na terceira fase ocorreu o desenvolvimento da aplicação *web* no qual todo o planejamento realizado é colocado em prática para dar vida à plataforma como um todo. O estudo das tecnologias na primeira fase foi crucial para o desenvolvimento, ainda tendo sido necessário realizar consultas das documentações dos respectivos frameworks utilizados para conseguir fazer uso de seus recursos e solucionar problemas ou imprevistos encontrados no caminho.

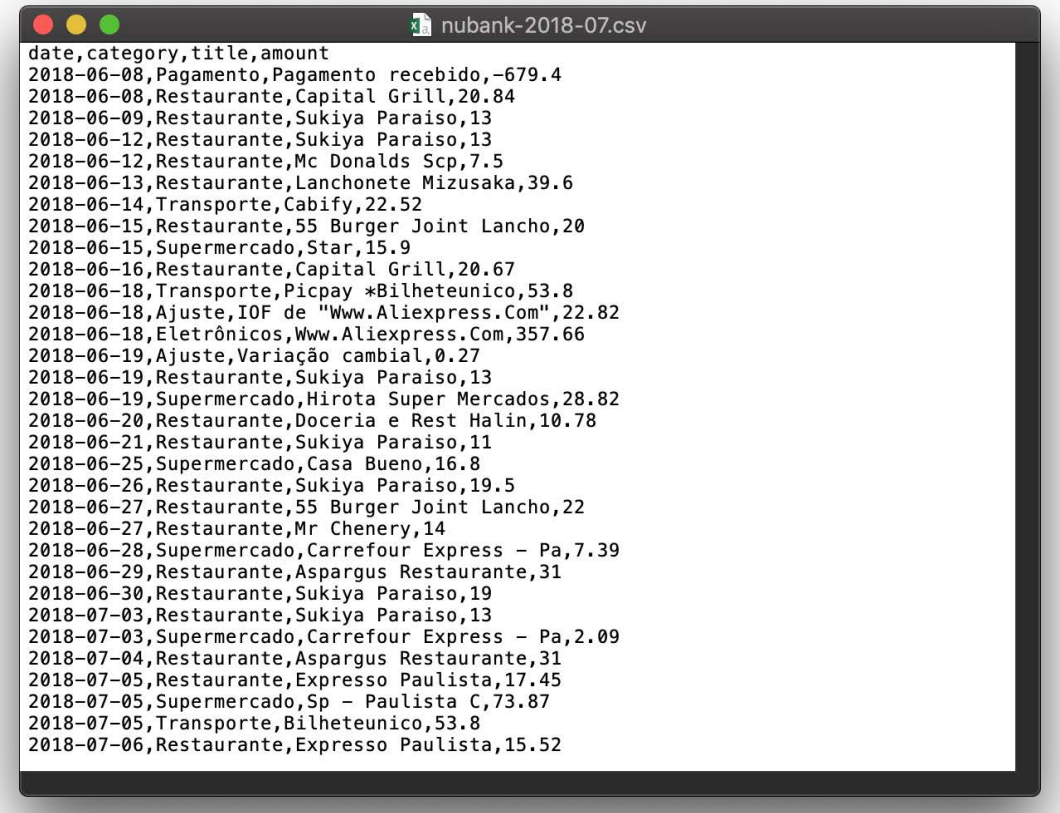
4 DESENVOLVIMENTO

Este trabalho foi dividido em três partes, sendo elas: a aplicação *web*, a *API* e o banco de dados. Tanto a aplicação *web* quanto a *API* possuem seus respectivos repositórios no *GitHub* e foram servidos em projetos individuais no *Heroku*. Já para o banco de dados foi utilizado o *mLab*, um *DBaaS* para banco de dados *MongoDB*.

4.1 DEFINIÇÃO DOS DADOS

A fatura do cartão de crédito *NuBank* é utilizada como fonte de dados sobre os gastos dos clientes. Esta fatura pode ser obtida em formato *CSV* através do site do banco. A Figura 1 apresenta um exemplo de fatura obtida.

Figura 1 – Exemplo da fatura do NuBank.



Fonte: Elaborada pelo autor.

A página do aplicativo do *NuBank* com gráficos e detalhes sobre as compras realizadas nos últimos 30 dias (Figura 2) foi utilizada como base para a plataforma desenvolvida. Nesta figura está presente o total gasto por dia em um gráfico de linha junto com a média de valor gasto; as três categorias que mais se gastou, junto com sua porcentagem sobre o gasto total e seu valor total – neste caso gastou-se mais com Vestuário, Lazer e Transporte, respectivamente; o local com compras mais frequentes e a compra mais cara realizada. O objetivo é de aprimorar as informações fornecidas pelo aplicativo e facilitar a análise de gastos sem restringir aos últimos 30 dias apenas.

Figura 2 – Página sobre os gastos dos últimos 30 dias do NuBank.

Fonte: Elaborado pelo autor.

Os dados obtidos sobre cada compra realizada com o cartão são: data (dia-mês-ano), categoria, título e valor. O *NuBank* possui um conjunto de categorias pré-definidas e reconhece automaticamente a que tipo de categoria tal gasto se enquadra, mas também é possível alterar a categoria caso ela não seja a mais adequada. Essas categorias são: casa, educação, eletrônicos, lazer, restaurante, saúde, serviços, supermercado, transporte, vestuário, viagem e outros. Cada uma delas representada também por um ícone, como pode-se observar na Figura 2, na qual o total de gastos com a categoria "vestuários" foi de R$ 490,00, o total da categoria "lazer" foi de R$ 209,78 e o da categoria "transporte" foi de R$ 178,64.

Porém nem todos os dados obtidos são relevantes para o projeto, então aplicou-se um filtro sobre os dados obtidos. Assim, itens como pagamentos e ajustes, atrasos de pagamento, ou similares foram removidos e apenas dados das compras realizadas foram mantidos. Também foram adicionados outros atributos a cada item, como a identificação do usuário ao qual aquele gasto pertence, o dia da semana no qual o gasto foi realizado, o mês, e o ano. Essas modificações permitem realizar pesquisas mais direcionadas no banco de dados. Na Figura 3 está um exemplo de gasto salvo no *mLab*. Informações como "_id" e "__v" foram atribuídas automaticamente pelo *mLab* ao salvar o dado no banco.

Figura 3 – Exemplo de um gasto salvo no mLab.

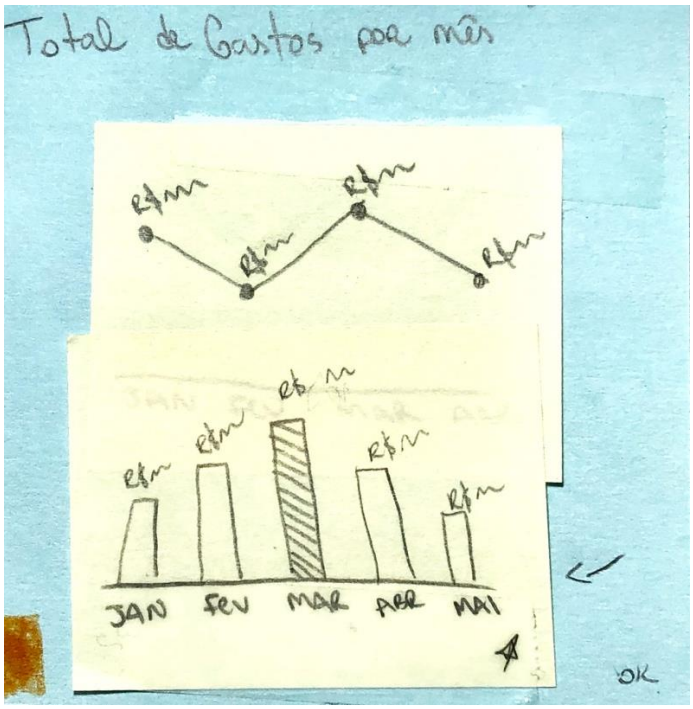

```
{
    "_id": {
        "$oid": "5bae5c3b186bee0389d45818"
    },
    "date": "2018-07-03T00:00:00-03:00",
    "weekday": "Terça-feira",
    "month": "julho",
    "year": "2018",
    "category": "Restaurante",
    "title": "Sukiya Paraiso",
    "amount": 13,
    "user_id": "5b782f91af1e9a7b604df4d3",
    "__v": 0
}
```


Fonte: Elaborado pelo autor.

4.2 DEFINIÇÃO DAS TELAS

Com as informações sobre os gastos já definidas, o próximo passo foi o de escolher as possíveis representações e o *layout* das páginas que iriam contê-las. Para isso optou-se por utilizar post-its com o dado a ser mostrado (Figura 4) – no caso, o “Total de gastos por mês” – e outro post-it para sua possível representação. Este método tornou mais simples de identificar quais seriam as principais representações, e quais poderiam ser descartadas neste momento. Assim como qual tipo de gráfico se enquadra melhor com o dado abordado em cada cartão.

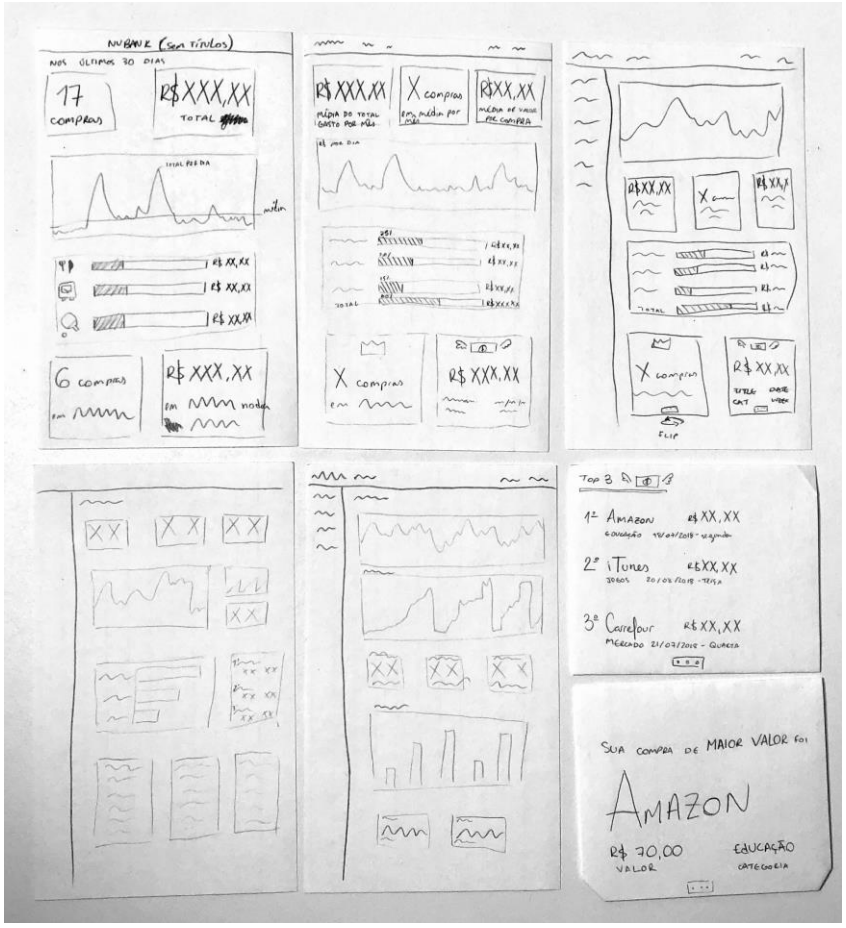
Figura 4 – Exemplo de cartão de dados.



Fonte: Elaborado pelo autor.

Em seguida, foram esboçadas algumas composições de tela para a página principal. O ponto de partida foi a página de gráficos presente no aplicativo *mobile* do *NuBank* (Figura 2). Com os cartões de dados definidos, restou solucionar qual a melhor posição para cada tipo de dado mostrado, a ordem que irão aparecer e o quanto de espaço ocupariam na tela. O objetivo foi escolher um *layout* no qual fosse possível apresentar o máximo de informações na mesma tela, tomando cuidado para não pecar com excesso de informações (Figura 5).

Figura 5 – Protótipos de layout para a página principal.



Fonte: Elaborado pelo autor.

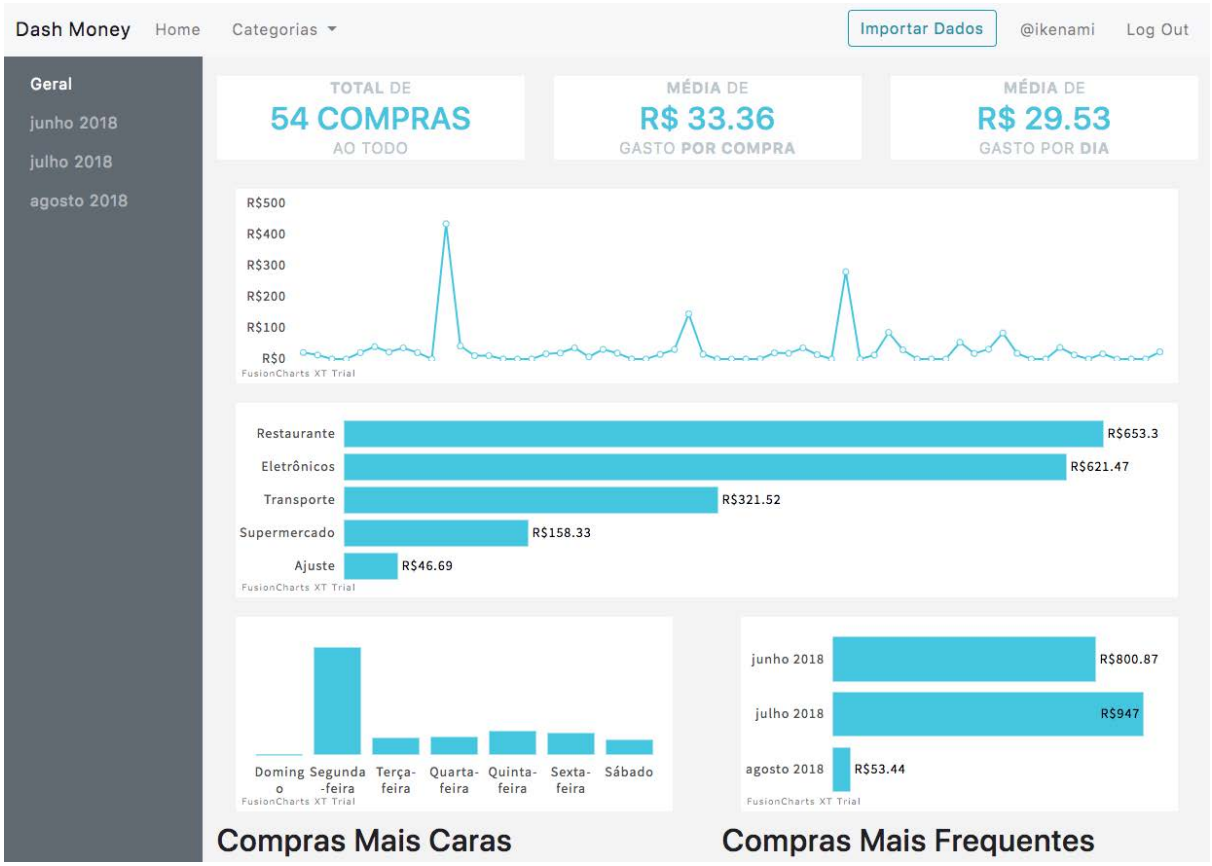
Como as outras telas da aplicação web não possuem componentes tão importantes como a apresentada na Figura 5, optou-se por não realizar seus protótipos. Essas páginas consistem em: página inicial (usuário não autorizado), página de login, página de registro de usuário, página principal (contém os gráficos e informações sobre os gastos), página de importar os dados para a plataforma, e a página com informações do usuário. Para as páginas de login e registro de usuário foram utilizadas a opção padrão da ferramenta Auth0 que oferece AaaS (Authentication as a Service).

4.2.1 PÁGINA PRINCIPAL (DASHBOARD)

Ao longo do desenvolvimento do projeto, a página principal foi a única que sofreu diversas alterações. Sua estrutura é utilizada tanto para as informações gerais de todos os dados contidos no banco sobre determinado usuário, quanto para

os dados filtrados por categoria e/ou filtrados por mês. Para o filtro por categoria, primeiramente, optou-se por um botão *dropdown* no menu localizado ao topo da página. Este botão representaria apenas as categorias compostas de pelo menos um dado. Enquanto que o filtro por tempo ficaria localizado à esquerda da tela, constando a opção "geral" (para visualizar todos os dados obtidos) e os meses com gastos, conforme a Figura 6. Nesta versão o conteúdo da página ainda está bem incompleto, faltando os títulos e algumas informações dos gráficos.

Figura 6 – Primeira versão da página principal.



Fonte: Elaborado pelo autor

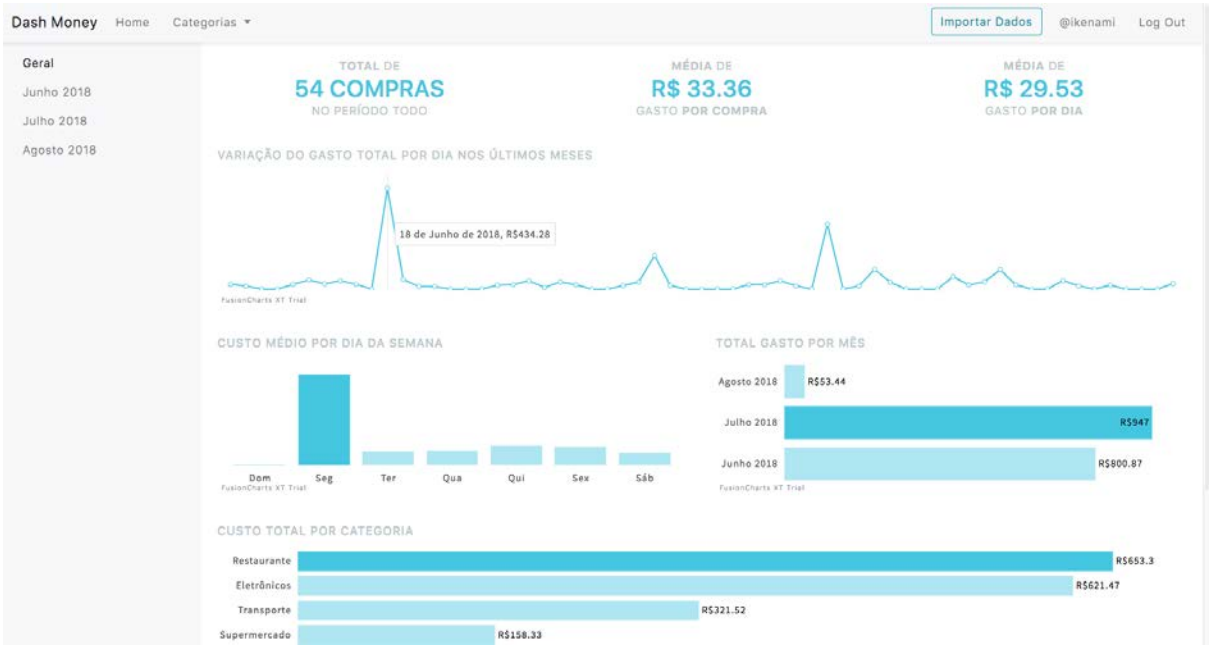
A próxima alteração foi a composição de cores para os gráficos. Pode se observar que no caso do gráfico com o gasto total por categoria, os dados foram ordenados da categoria de maior gasto para a categoria de menor gasto. Já no gráfico de gasto total por dia da semana ou por mês isso não ocorreu. O motivo para isso se deve ao fato de que ordenar os dados em ordem crescente ou decrescente facilita a visualização dos itens de maior ou menor valor dentro daquele grupo. Porém não se pode fazer tal ordenação para itens que possuem uma ordem cronológica já definida em sua base.

Por exemplo, tornaria confuso para o leitor se no caso dos gráficos com o gasto total por mês fosse ordenado da seguinte forma:

- julho 2018: R$ 947,00
- junho 2018: R$ 800,87
- agosto 2018: R$ 53,44

Um leitor desatento pode acabar não percebendo que os meses de julho e junho estão com ordem invertida, o que pode resultar numa conclusão errônea sobre se seus gastos estão aumentando ou diminuindo com o passar dos meses. Os gráficos devem ser uma forma de facilitar a rápida compreensão de um conjunto de dados. Visando isso, optou-se por destacar com uma cor mais forte os itens com o maior valor de cada grupo, conforme a Figura 7. Também adotou-se títulos para cada cartão de dados a fim de tornar explícito qual seu contexto. Um detalhe a ser observado é que no cartão "Total gasto por mês" alternou-se a ordenação para que os meses mais recentes ficassem no topo do gráfico, e no cartão “Variação do gasto total por dia nos últimos meses” é possível ter acesso a data e o valor total gasto naquele dia ao mover o cursor pelo gráfico. Já no gráfico "custo médio por dia da semana", abreviou-se os dias da semana para dar mais leveza ao gráfico, sem perder nenhuma informação por isso.

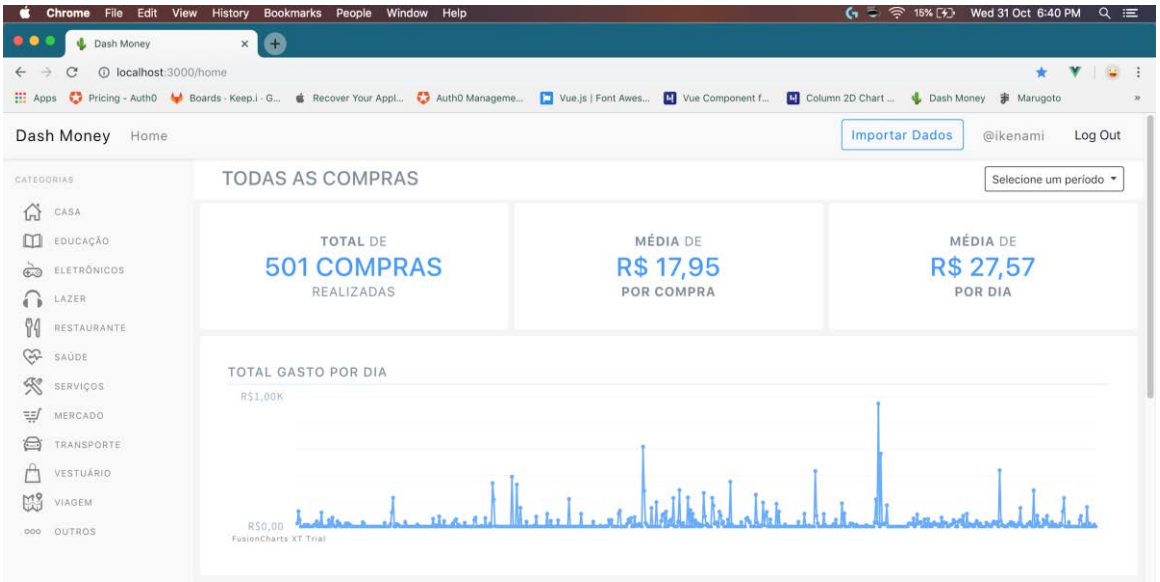
Figura 7 – Segunda versão da página principal.



Fonte: Elaborada pelo autor.

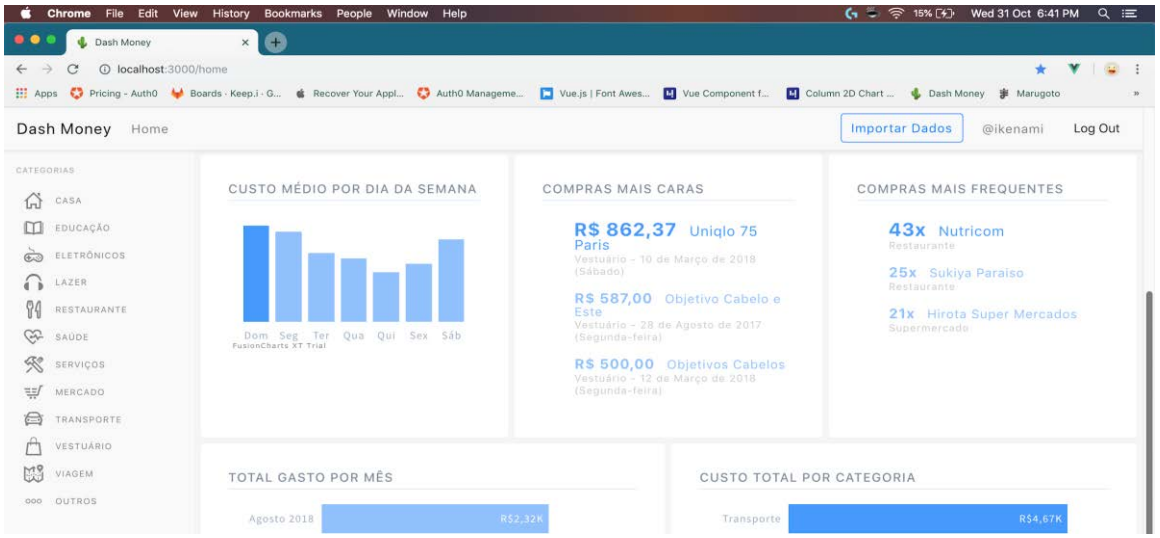
Na versão final (Figuras 8, 9 e 10), optou-se por remover o botão de categorias do menu, listando as categorias na lateral esquerda da tela junto com seus ícones. Enquanto que o filtro com o período de tempo passou a ser um botão *dropdown* junto ao título da página, ordenando pelos meses mais recentes ao topo da lista.

Figura 8 – Versão final da página principal, parte 1.



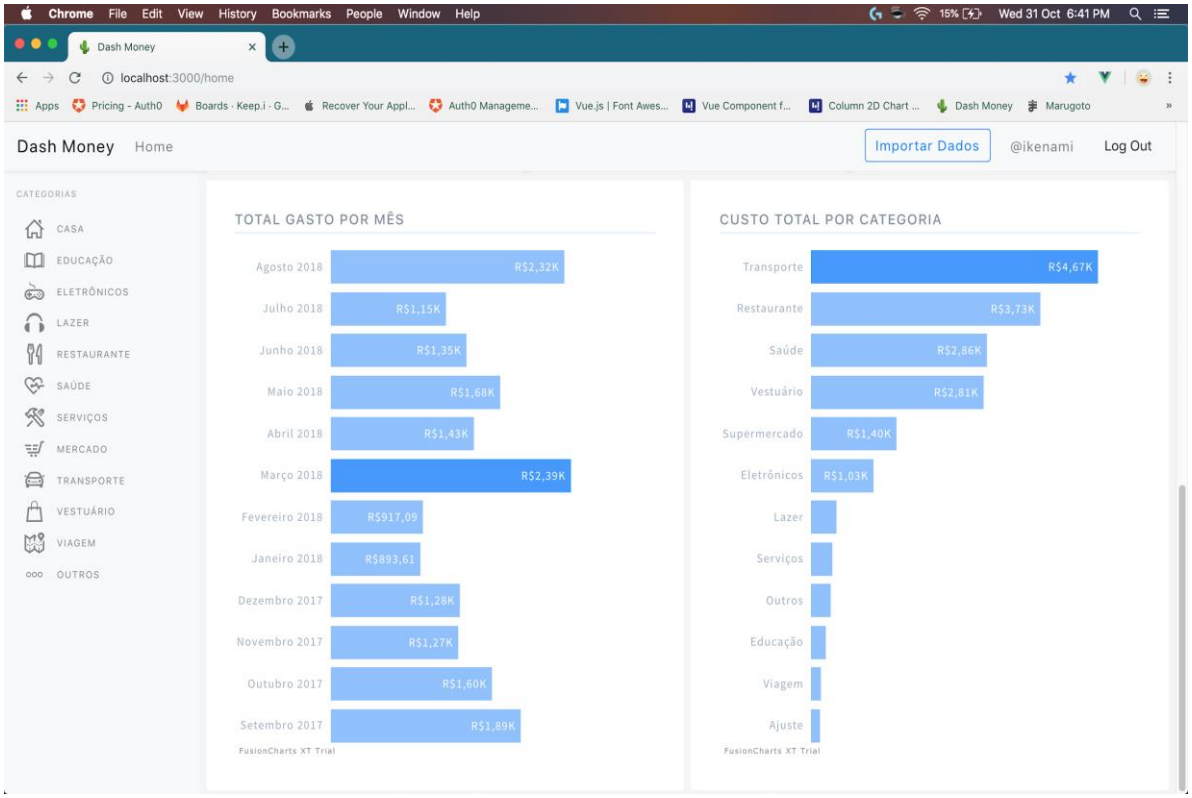
Fonte: Elaborado pelo autor.

Figura 9 – Versão final da página principal, parte 2.



Fonte: Elaborado pelo autor.

Figura 10 – Versão final da página principal, parte 3.



Fonte: Elaborado pelo autor.

Nas Figuras 8, 9 e 10, estão presentes todos os cartões de dados disponíveis na plataforma. São eles, respectivamente:

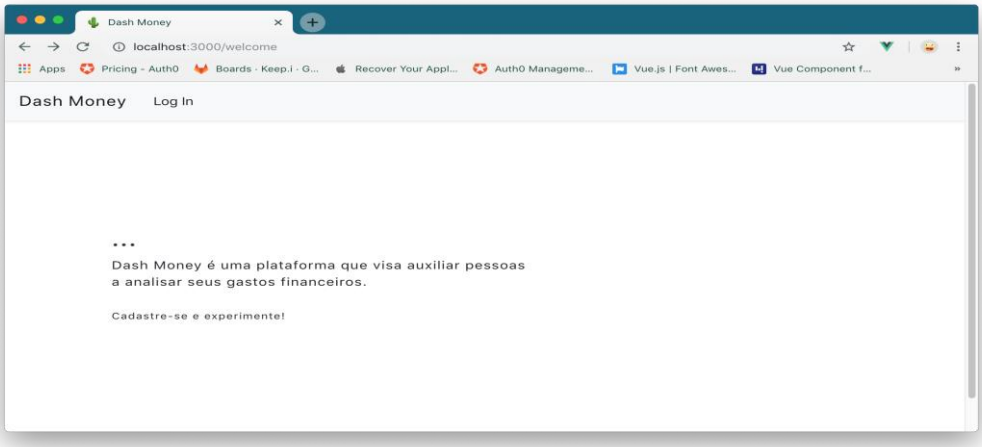
- Total de compras realizadas
- Média de valor por compra
- Média de valor gasto por dia
- Total gasto por dia
- Custo médio por dia da semana
- Compras mais caras
- Compras mais frequentes
- Total gasto por mês
- Custo total por categoria

4.2.2 ACESSO

Foram utilizadas as páginas padrões do *Auth0* para as páginas de *login*, registro de novo usuário e de resetar a senha. Na página inicial da aplicação *web*,

quando o usuário ainda não está autenticado, é preciso clicar no botão de *login* para acessar na plataforma.

Figura 11 – Página de acesso à plataforma.

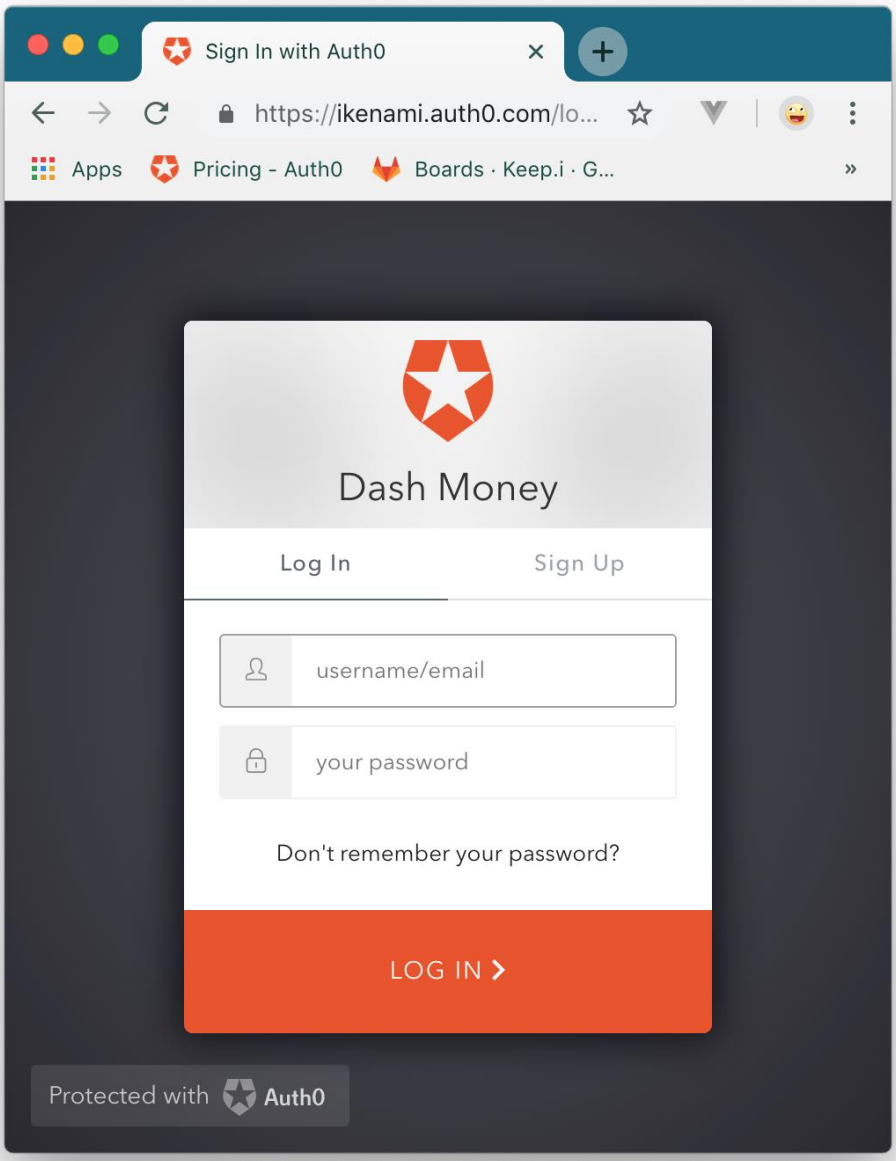


Fonte: Elaborado pelo autor.

4.2.2.1 Login

Ao clicar no botão de *login*, o usuário é redirecionado para a tela de *login* do *Auth0* (Figura 12). Nela o usuário pode se conectar através tanto do *username* quanto do *e-mail* utilizando na hora de se registrar.

Figura 12 – Página de login.

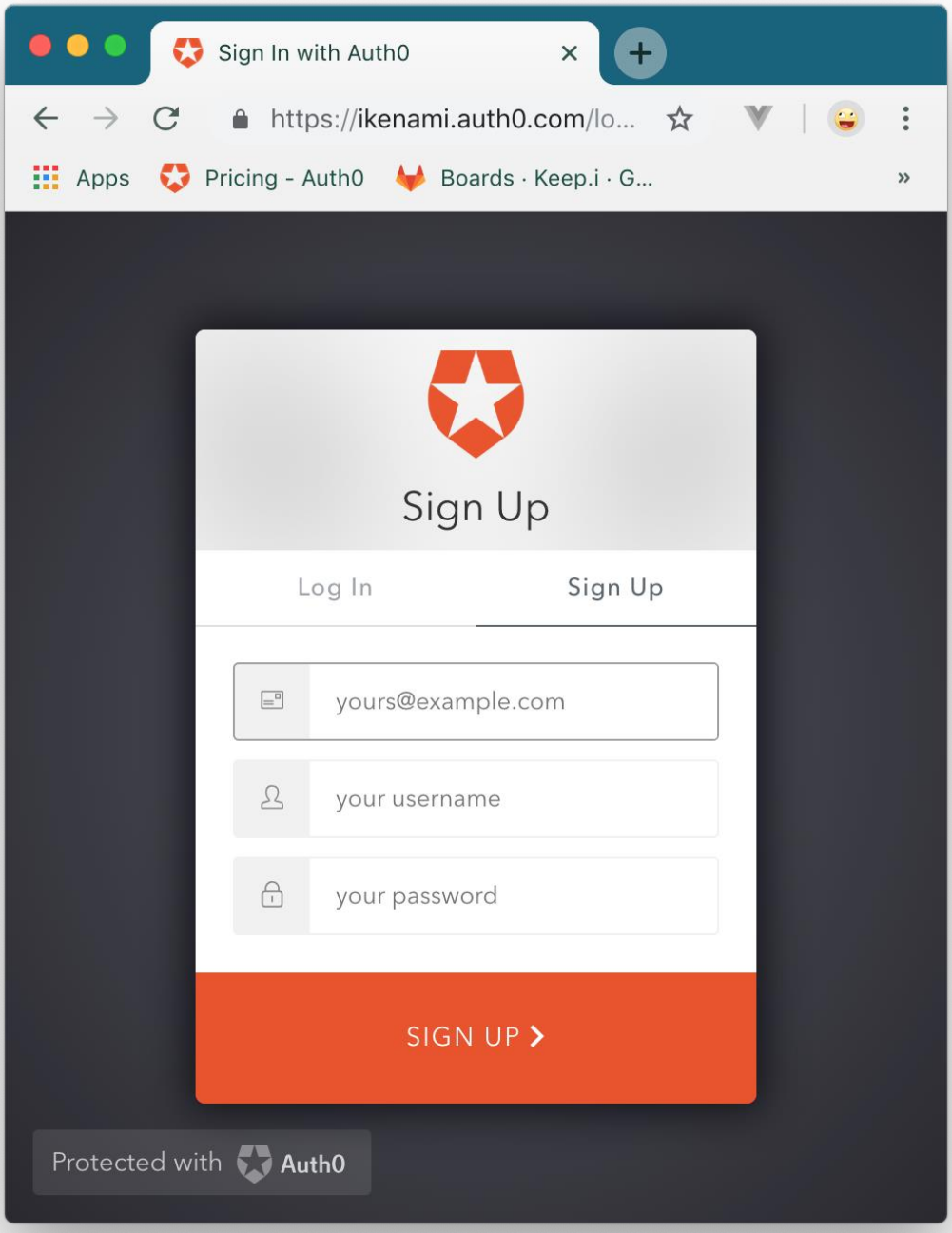


Fonte: Elaborado pelo autor.

4.2.2.2 Cadastro

Para o usuário se registrar na plataforma ele precisa fornecer seu endereço de *e-mail*, escolher um *username* e definir sua senha de acesso (Figura 13).

Figura 13 – Página de cadastro.

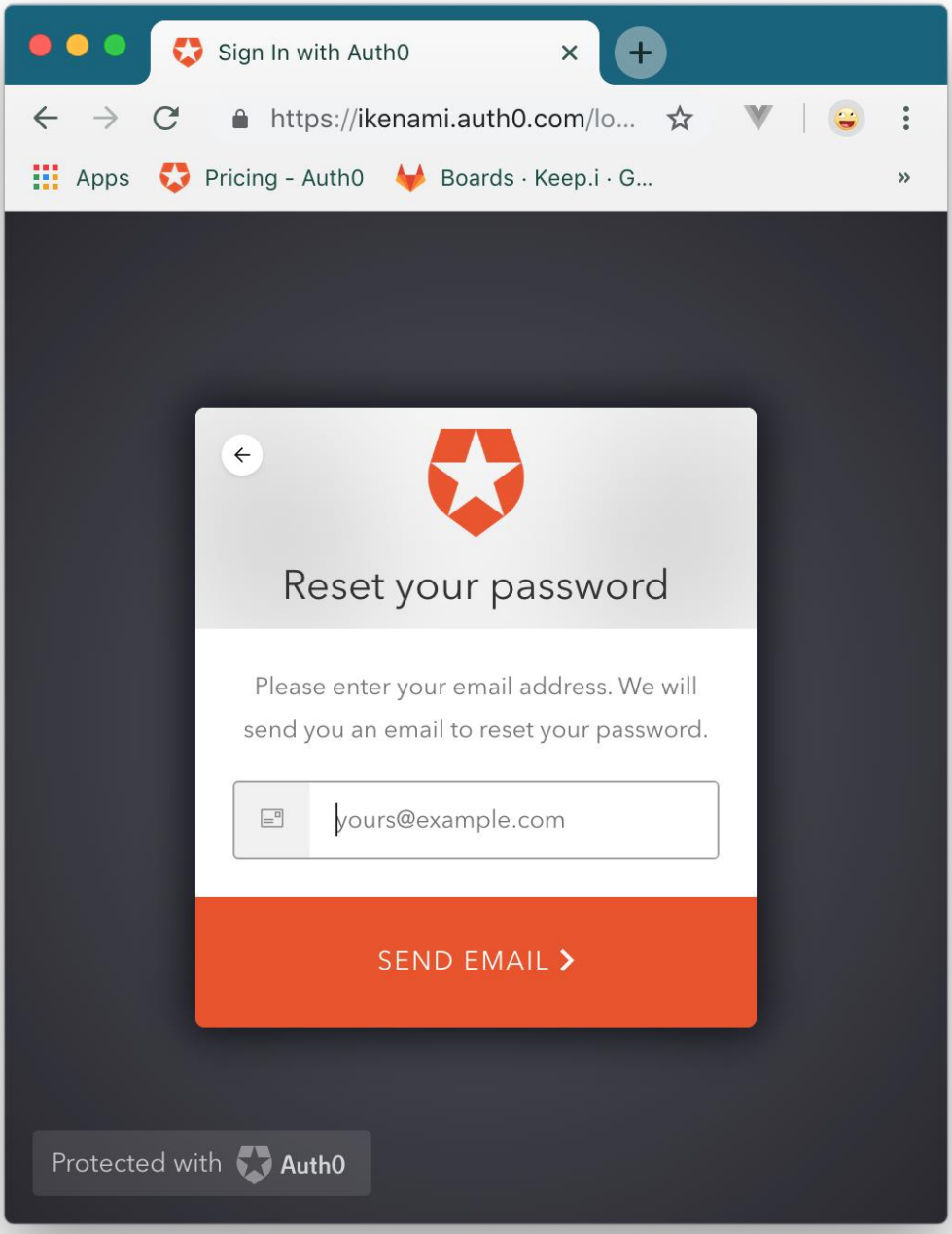


Fonte: Elaborado pelo autor.

4.2.2.3 Redefinir a senha

Para redefinir a senha basta preencher o formulário com o *e-mail* utilizado na hora do registro (Figura 14). O usuário logo receberá um *e-mail* com o *link* de redirecionamento para a redefinição de senha.

Figura 14 – Página de redefinição de senha.

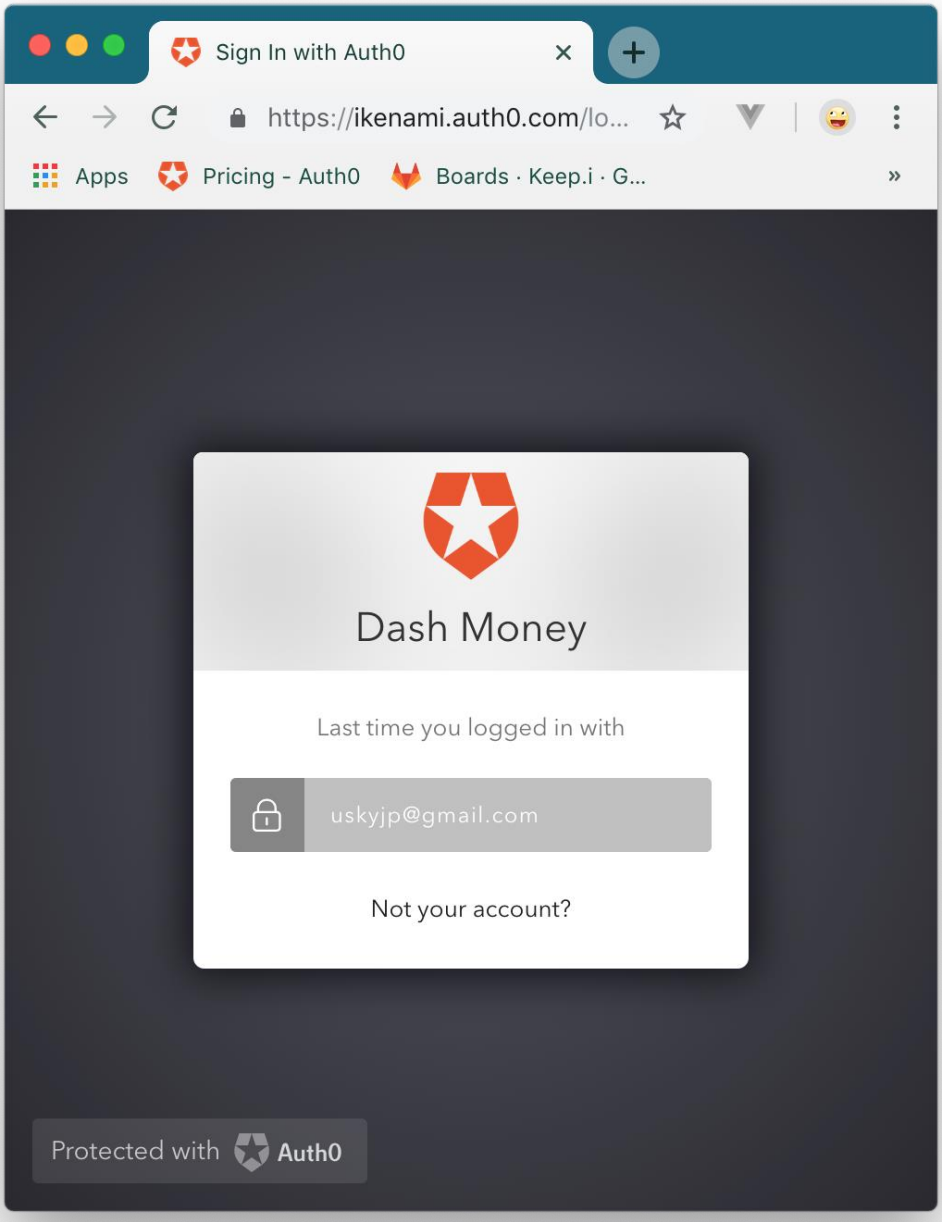


Fonte: Elaborado pelo autor.

4.2.2.4 Acesso rápido

O *Auth0* também pode recordar o último usuário autenticado, caso esta opção seja acionada em sua configuração. Isso permite que o usuário possa acessar a plataforma rapidamente, sem precisar preencher suas informações pessoais (Figura 15).

Figura 15 – Página para acesso rápido.

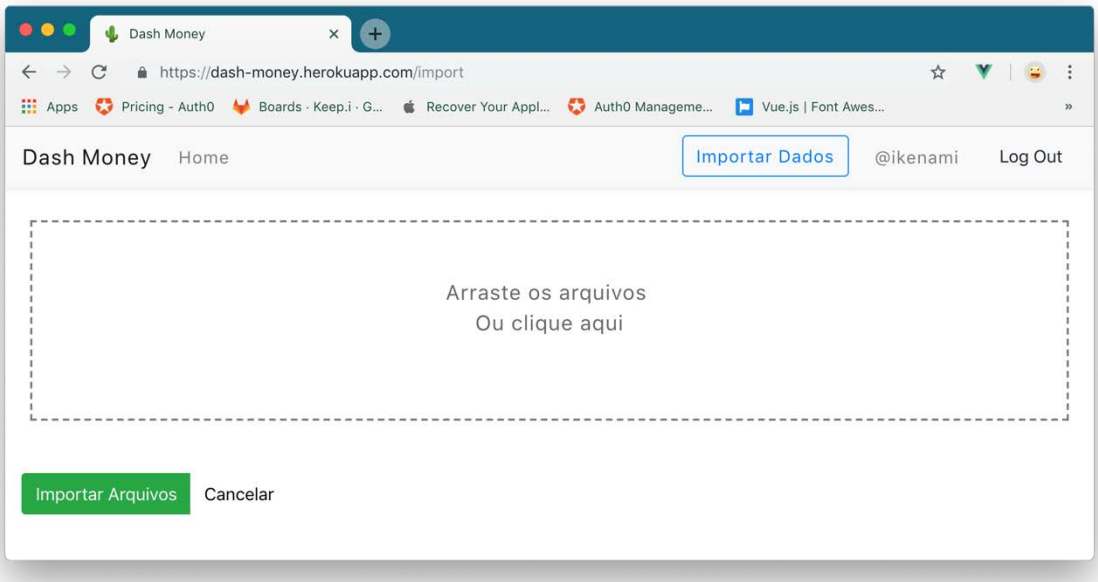


Fonte: Elaborado pelo autor.

4.2.3 IMPORTAR DADOS

Esta página é responsável pela importação dos arquivos *CSV* que contém os dados da fatura. É possível adicionar arquivos ao arrastá-los para a caixa desenhada na página, ou clicando na caixa tracejada (Figura 16). Ao clicar na caixa, uma janela irá aparecer com as pastas do computador para que o arquivo seja selecionado. Após selecionar os arquivos é possível cancelar a ação, ou iniciar a importação de dados. É neste momento que a ferramenta *PapaParse* entra em ação, transformando os dados presentes nos arquivos em objetos *JSON* para que sejam enviados para a *API*, tratados e salvos no banco de dados.

Figura 16 – Página de importação de dados.

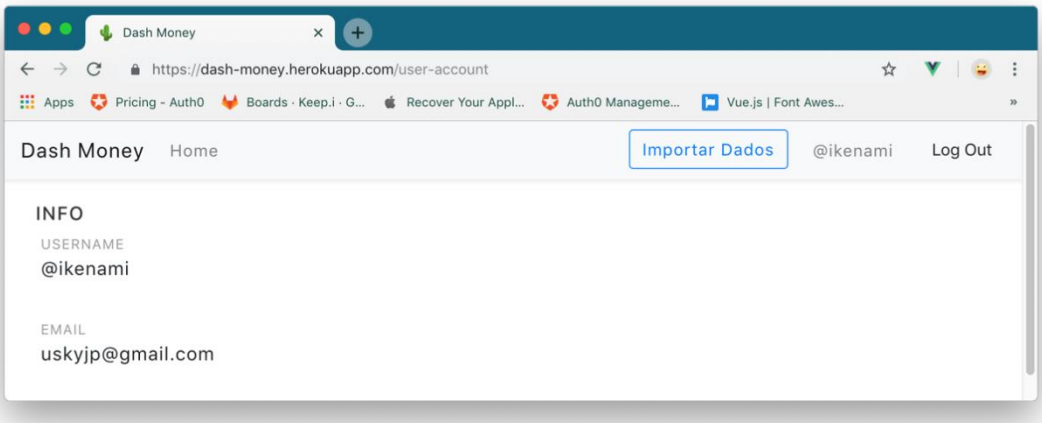


Fonte: Elaborado pelo autor.

4.2.4 PERFIL DO USUÁRIO

A página do perfil do usuário contém informações sobre a conta do mesmo, como o *username* e o *e-mail* utilizado na hora de seu registro. Atualmente não é possível alterar essas informações através da plataforma desenvolvida.

Figura 17 – Página do perfil do usuário.

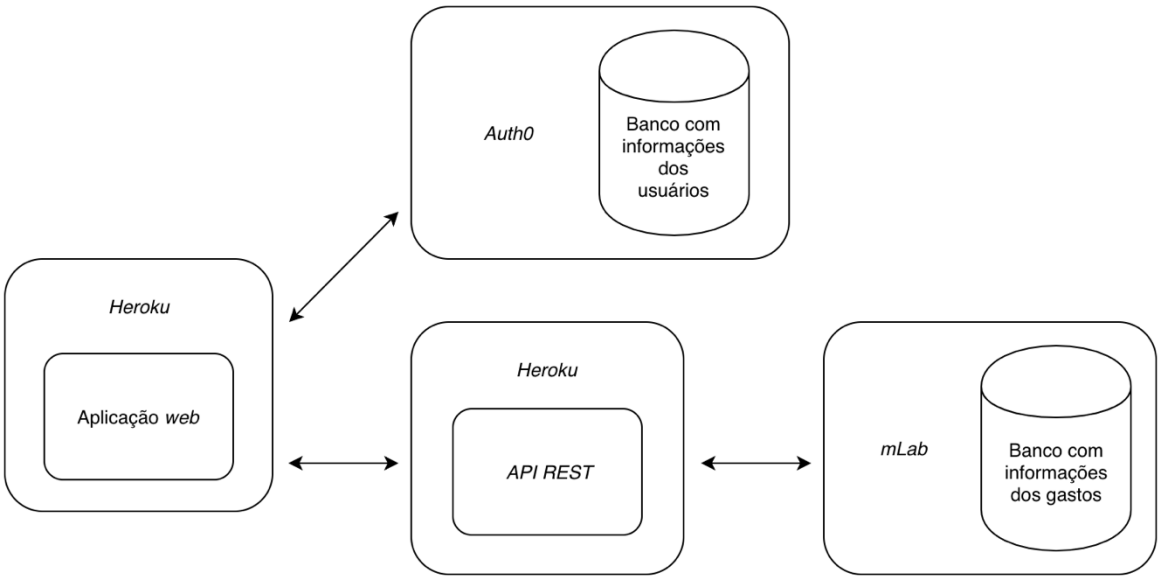


Fonte: Elaborado pelo autor.

4.3 ESTRUTURA DO PROJETO

Como mencionado anteriormente em outros capítulos, projeto está dividido em três partes. São eles o *Front-End* (a aplicação *web*), a *API* e o Banco de Dados (Figura 18). Tanto o *Front-End* quanto a *API* estão hosteados no *Heroku* em projetos separados. Já o Banco de Dados com as informações sobre os gastos dos usuários está hosteado no *mLab*, enquanto que para guardar as informações sobre os usuários utiliza-se o Banco de Dados da plataforma *Auth0*.

Figura 18 – Estrutura do sistema.



Fonte: Elaborado pelo autor.

A plataforma *Auth0* é responsável por gerenciar as autenticações dos usuários na plataforma, assim como providenciar páginas de *login*, registro e redefinição de senha para a aplicação *web*. A aplicação *web* conversa diretamente com a plataforma *Auth0* para realizar estas funções.

Já a *API* funciona como um intermediário entre a aplicação *web* e o Banco de Dados sobre os gastos dos usuários. A aplicação *web* faz requisições individuais à *API* a cada cartão de dados para obter os dados necessários na hora de gerar os gráficos e informações. A *API*, por sua vez, recebe as requisições feitas pela aplicação *web* e, de acordo com a requisição, obtém os dados do Banco de Dados presente no *mLab*, tratando as informações obtidas para que fiquem no formato ideal para se utilizar nos cartões de dados e retorna as informações à aplicação *web* como um objeto *JSON*.

Quando o usuário importa um arquivo *CSV*, a aplicação *web* também é responsável por traduzir este arquivo em um objeto *JSON* e enviar à *API*. A seguir, a *API* valida este conjunto de dados e os salva no Banco de Dados presente no *mLab*.

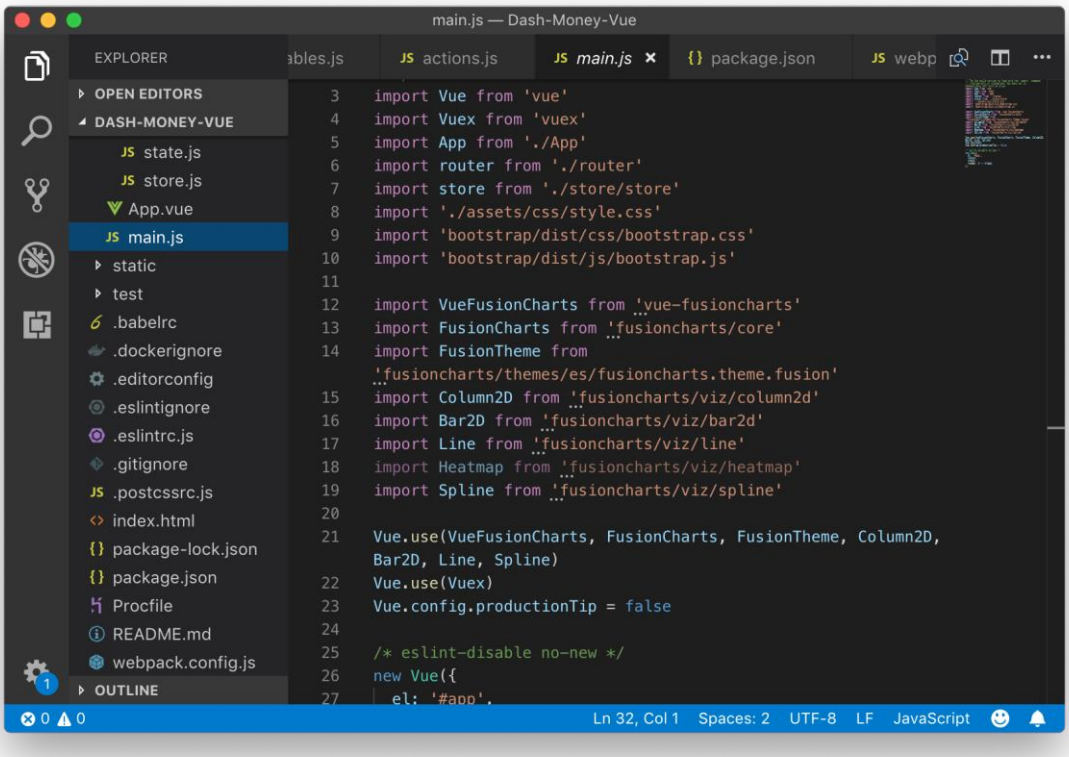
Nesta seção serão abordados detalhes sobre o processo de desenvolvimento do *Front-End* e da *API*.

4.3.1 FRONT-END

A primeira parte do projeto a ser desenvolvida, após a definição dos dados mencionada anteriormente, foi o *Front-End*. Para isso criou-se um projeto *Vue*, chamado de "Dash-Money-Vue", a partir de um *template* oferecido pelo *framework* para desenvolvimento de aplicações *web*.

A partir deste *template* criou-se os componentes das páginas, definiu-se suas rotas de acesso utilizando *Vue Router*, e as funções de requisição de dados para a *API* e o tratamento de dados pelo *Vue* utilizando *Vuex*. Para a estilização dos componentes foi importado a biblioteca *Bootstrap*, e para a construção dos gráficos, importou-se a biblioteca do *FusionCharts* para *Vue* (Figura 19).

Figura 19 – Importação das ferramentas para o projeto Vue.



Fonte: Elaborado pelo autor.

Após obter uma estrutura definida para o *Front-End*, criou-se um projeto no *website* do *Auth0* para lidar com o gerenciamento de usuários e autorizações. O *Auth0* possui materiais prontos para serem utilizados para diversas linguagens e *frameworks*, entre eles o material para projetos com *Vue*.

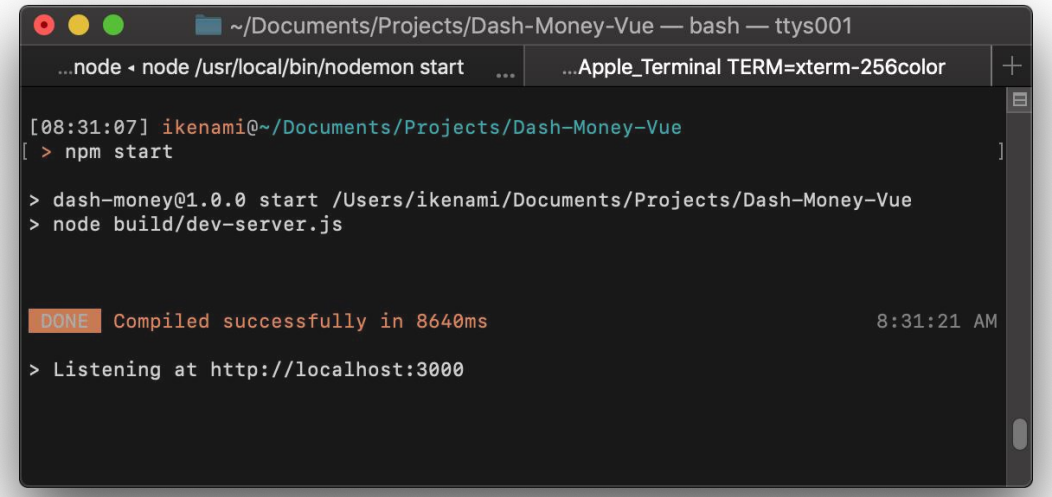
Para configurar o *Auth0* no projeto *Vue*, foi realizado *download* de um projeto de exemplo oferecido pelo *Auth0* e alterou-se as informações como a *Url* de *callback*. Essa *Url* será chamada após a autenticação do usuário pelo *Auth0*. Todas as informações dos usuários são salvas no banco de dados do *Auth0*, e podem ser acessadas através da plataforma.

4.3.1.1 Build

É possível rodar o projeto *Vue* localmente em sua máquina através do comando: *npm start*. Para isso é preciso estar no diretório do projeto onde consta o arquivo 'package.json'. Caso tudo ocorra como o previsto, irá aparecer a mensagem

"Listening at http://localhost:3000" (Figura 20) e a página *web* do projeto será aberta no navegador.

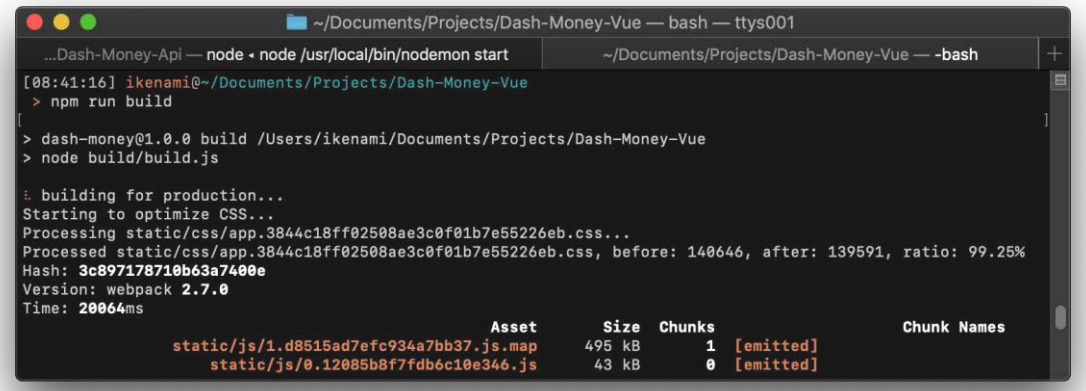
Figura 20 – Iniciando o projeto localmente pelo terminal.



Fonte: Elaborado pelo autor.

Para ser utilizado em produção, o projeto *Vue* precisa ser compilado utilizando o comando *npm run build* (Figura 21). Ao final do *build* será gerado um diretório "dist" com todo o conteúdo da aplicação desenvolvida. Neste diretório consta o arquivo "index.html" e uma pasta "static" que contém o *favicon* (ícone presente na aba do navegador), os arquivos *CSS* e os arquivos *JavaScript* compilados É importante lembrar que apenas abrir o arquivo "index.html" no navegador não irá funcionar, pois essa versão foi desenvolvida para ser utilizada em um servidor.

Figura 21 – Realizando a build do projeto.



Fonte: Elaborado pelo autor.

Como na fase final o projeto é servido no *Heroku*, foi preciso desenvolver o código que irá servir a aplicação *web*. Para isso desenvolveu-se um outro projeto bem simples utilizando *Node.js* e *Express*, nomeado por "Dash-Money-Server". Nele importou-se o conteúdo gerado na pasta "dist" (obtido ao compilar o projeto "Dash-Money-Vue") para a pasta "app>public". Sendo todos os acessos via *Url* redirecionados para o arquivo "index.html".

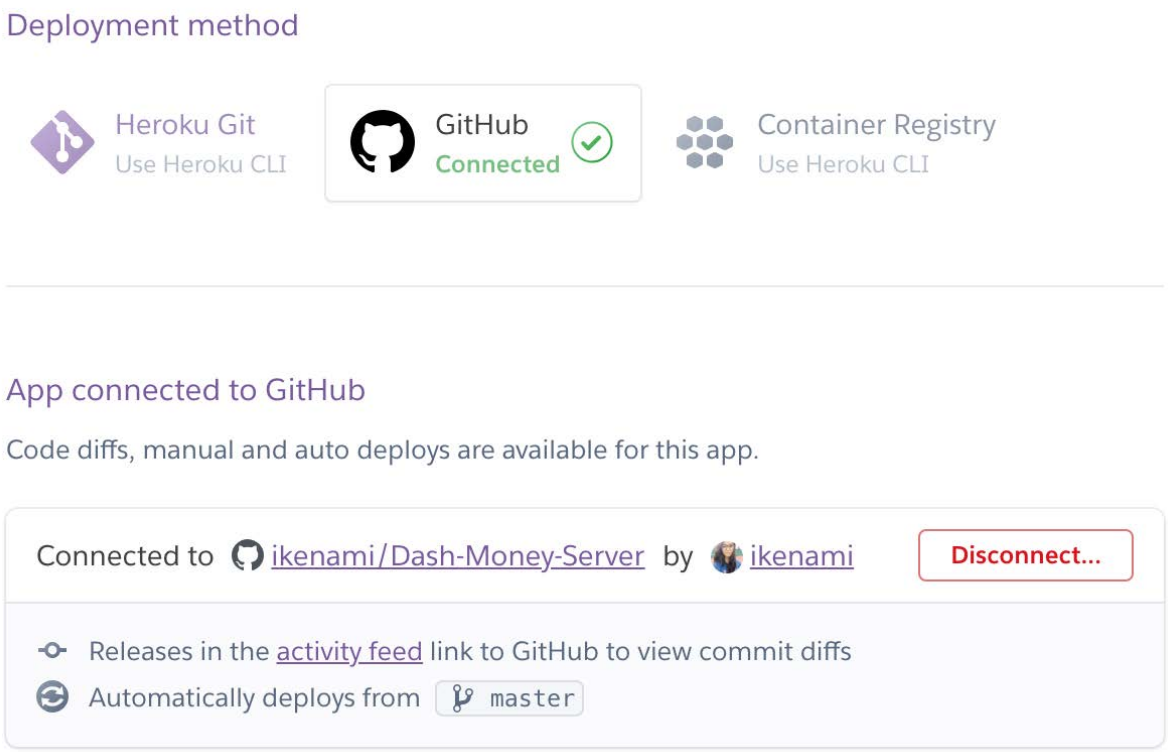
4.3.1.2 Produção

Agora que a parte do *Front-End* está pronta para ser servida, já é possível servir o projeto no *Heroku*. *Heroku* é um *PaaS*, o que facilita na hora de servir aplicações pois não é preciso se preocupar com a configuração do servidor, redes, balanceamento de carga, entre outros detalhes.

Utiliza-se a versão gratuita na qual é disponível uma máquina, chamada de *Dyno*, com 512 MB de *RAM*. Seu recurso é bem limitado, o suficiente para o uso neste projeto acadêmico. Na conta gratuita, o *Dyno* tem um limite de 1000 horas mensais, sendo essas horas compartilhadas entre todos os projetos da conta. O estado padrão do *Dyno* é de não gastar recursos, sendo "acordado" ao receber uma requisição. Caso não haja mais nenhuma atividade, após meia hora, o *Dyno* volta ao seu estado "adormecido", economizando as horas disponíveis.

Para servir o *Front-End* criou-se o projeto "dash-money" no *Heroku*, selecionando a região dos Estados Unidos já que não havia uma opção para a América do Sul. Utilizou-se o método de *Deploy* através da integração com o *GitHub*, no qual é possível selecionar uma *branch* de um repositório para fazer *deploy* automático sempre que aquela *branch* receber um novo *commit*. Foi escolhido a *branch* "master" do repositório "Dash-Money-Server" (Figura 22).

Figura 22 – Método de *deploy* para a aplicação do *Front-End*.

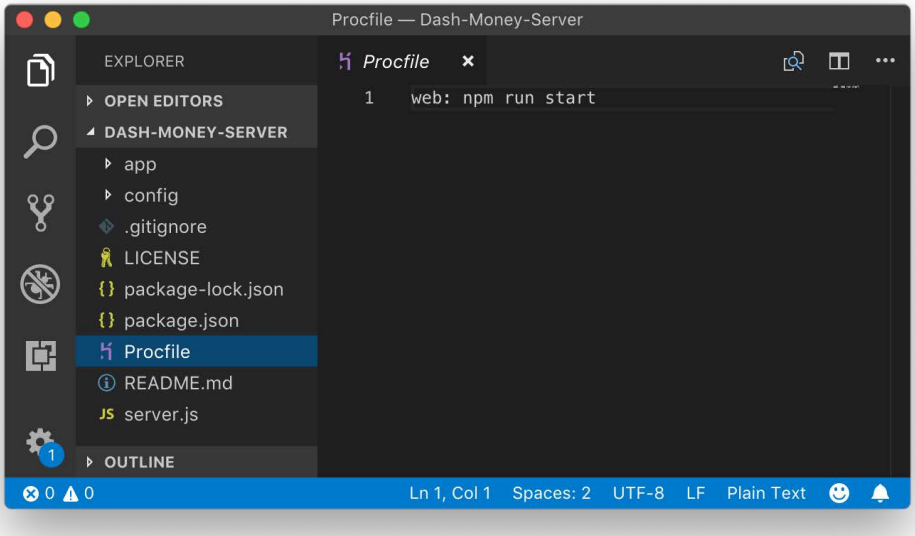


Fonte: Elaborado pelo autor.

Da mesma forma que é preciso executar o comando "npm start" para rodar a aplicação localmente, também é necessário escrever um *script* para iniciar a aplicação no servidor. Para isso é necessário adicionar um *buildpack* no Heroku. O *buildpack* é responsável por instalar as dependências da aplicação e configurar o ambiente. Para este projeto, utiliza-se o *buildpack* para *Node.js*.

Também é preciso adicionar um arquivo "Procfile" na raiz do diretório do projeto com o *script* a ser rodado. Nesse caso o conteúdo do arquivo é "web: npm run start", conforme a Figura 23.

Figura 23 – Script a ser executado no Heroku.



Fonte: Elaborado pelo autor.

Ao salvar as alterações e fazer um *commit* para a *branch master*, o *deploy* com a versão mais recente do projeto é iniciado na plataforma *Heroku*. Ao final deste processo é possível acessar a aplicação *web* através da *url* disponibilizada pelo *Heroku*. No caso desta aplicação, o link disponível é "*https://dash-money.herokuapp.com/*".

4.3.2 API

A *API* deste projeto trabalha como intermediário entre a aplicação *web* e o banco de dados. Seu papel é de receber as requisições via métodos *HTTP*, realizar operações com o banco de dados, tratar as informações recebidas e enviar a resposta através de um objeto *JSON*.

Como os dados a serem salvos no banco já foram definidos assim como os componentes das páginas da aplicação, restou definir quais requisições precisavam serem desenvolvidas. Por questão de organização, foram separados alguns cartões com o título da requisição, seus parâmetros, possíveis filtros e qual a resposta esperada. A seguir, marcou-se nos cartões com uma caneta rosa quando a requisição terminou de ser desenvolvida, e na cor roxa caso tenha funcionado corretamente.

Todos os métodos da *API* são relacionados aos gastos e não aos usuários. Para obter mais informações da conta do usuário, assim como autenticação, realização de *login*, *logout* e cadastro, utilizou-se *Auth0*. Assim, os detalhes da conta do usuário estão salvos no banco de dados próprio do *Auth0*, enquanto que os dados sobre os gastos realizados estão salvos no banco de dados do *mLab*.

Para desenvolver a *API* criou-se o projeto "Dash-Money-API" utilizando *Node.js* e *Express*. Para a conexão com o banco de dados *MongoDB* no *mLab* utilizou-se a biblioteca *mongoose* (Figura 24). As informações do banco de dados no mLab, tais quais o usuário, senha e *URI* de acesso são disponibilizadas pela plataforma do banco. Esses dados ficam disponíveis após a criação do banco de dados. Por questões de segurança, utilizou-se variáveis de ambiente para guardar as informações do banco.

Figura 24 – Configuração do acesso ao banco de dados.

```
const mongoose = require('mongoose');
let dbUrl = `mongodb://${process.env.DB_USER}:${process.env.DB_PASS}@${process.env.DB_URI}`;
let mongoDB = process.env.MONGODB_URI || dbUrl;
let option = { useNewUrlParser: true }
mongoose.connect(mongoDB, option);
mongoose.Promise = global.Promise;
let db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

Fonte: Elaborado pelo autor.

Com os *endpoints* definidos, o próximo passo foi definir as rotas de requisição. Para cada rota também foi definido um *controller*, ou seja, um método a ser acionado assim que houver uma requisição naquela rota.

Esse *controller* é responsável por identificar os parâmetros e filtros utilizados na requisição, fazer a requisição para o banco de dados e acionar outros métodos de apoio para o tratamento do resultado obtido. Esse método é acionado pela aplicação *web* quando um usuário importa dados da fatura de um arquivo CSV. A aplicação *web* transforma o arquivo em um objeto *JSON* e envia o conjunto de gastos para a *API* (no caso a variável "req.body.costs"), junto com a identificação do usuário (recebida pela variável "req.body.user_id").

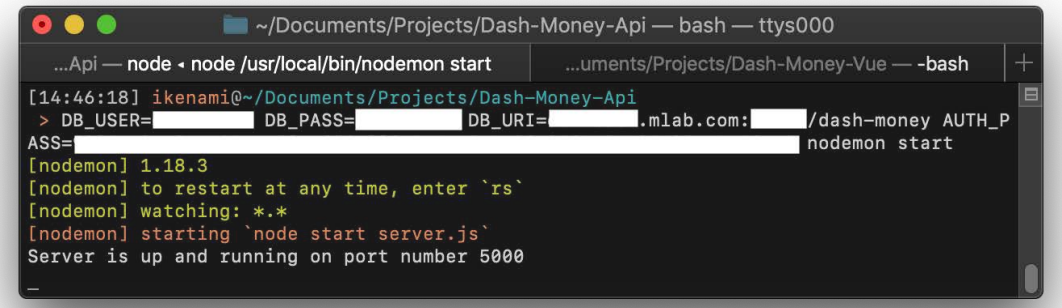
Antes de ser salvo no banco, cada gasto presente no conjunto passa por uma transformação na qual atributos como "dia da semana", "mês" e "ano" são adicionados. Para essas funções simples utilizadas foi criado um arquivo específico que funciona como uma simples biblioteca, chamada de "*dm_moment*". Cada um

dos gastos é inicializado como um objeto *Cost* definido através do *mongoose Schema*, antes de serem adicionados a uma lista para salvá-los no banco.

A fim de testar a *API* de forma prática e independente do *Front-End*, utilizou-se a ferramenta Postman. Nela foi criada uma *Collection* específica para este projeto, chamada de "Dash Money API". As *Collections* funcionam como diretórios onde se pode agrupar diversas requisições e definir variáveis de ambiente para serem utilizadas nos testes.

Para executar a API localmente, utilizou-se o *Nodemon* (Figura 25). Ele permite que a *API* executada seja atualizada automaticamente sempre que uma alteração for feita no código, eliminando a necessidade de reiniciar a *API* manualmente toda vez. Neste caso, as variáveis de ambiente foram definidas ao iniciar a aplicação. Os dados sensíveis foram omitidos da imagem.

Figura 25 – Inicializando a API localmente.

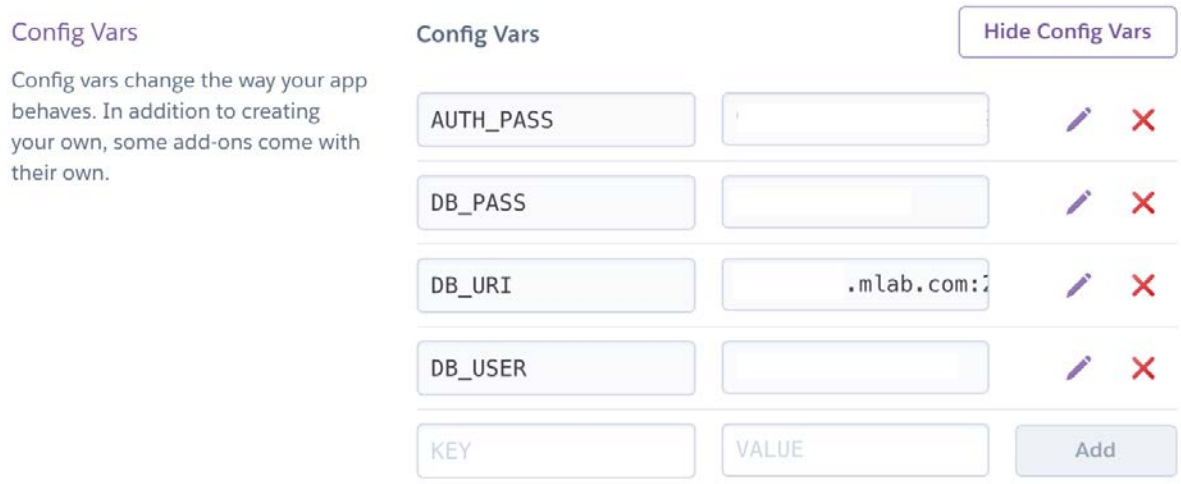


Fonte: Elaborado pelo autor.

4.3.2.1 Produção

Para servir a *API* no *Heroku* foram utilizados os mesmos passos do *Front-End*. Com a exceção de que desta vez foi preciso configurar variáveis de ambiente no projeto do *Heroku* através da opção "Config Vars" (Figura 26). Pode-se observar que essas são as mesmas variáveis utilizadas na Figura 25 para realizar conexão com o *mLab* (as informações sensíveis foram omitidas da imagem).

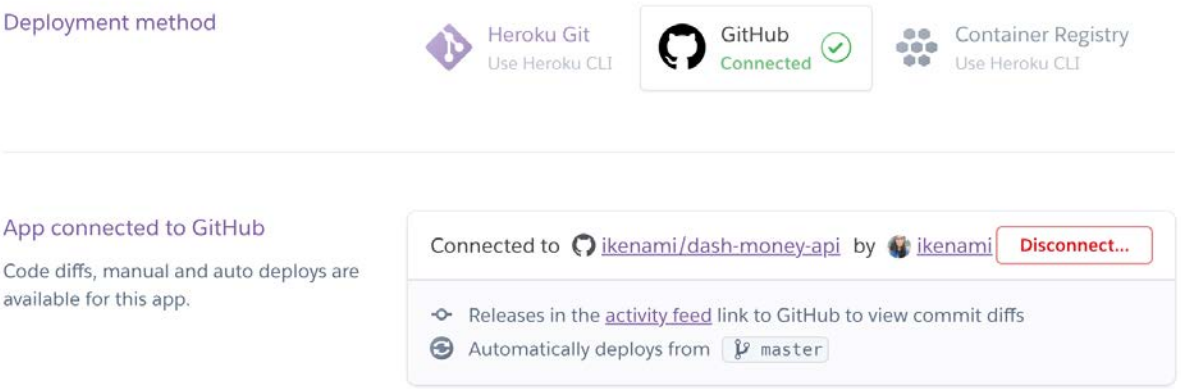
Figura 26 – Configuração das variáveis globais no Heroku.



Fonte: Elaborado pelo autor.

Assim como citado anteriormente, utilizou-se o recurso de *deploy* através da integração com o *GitHub*. O processo de *deploy* é iniciado toda vez que um *commit* for feito na *branch master* do repositório "dash-money-api". Quando o processo for finalizado, a aplicação estará disponível para acesso pelo *link* "https://dash-money-api.herokuapp.com/" fornecido pelo *Heroku*.

Figura 27 – Configuração do deploy da API no Heroku.



Fonte: Elaborado pelo autor.

5 APLICAÇÃO WEB

A aplicação *web* desenvolvida tem como o objetivo auxiliar o usuário a compreender melhor seus gastos. Como o público alvo escolhido são usuários do cartão de crédito *NuBank*, é preciso que o usuário a utilizar esta plataforma tenha uma conta no *NuBank* para que consiga utilizar todas as funcionalidades da aplicação desenvolvida.

Ao acessar o *site* (Figura 11) é possível acessar a página de *login* pelo botão “Login” localizado no topo da página. Para cadastrar um novo usuário basta preencher o formulário com seu *e-mail*, *username* e uma senha na tela de registro (Figura 13). Em seguida será possível utilizar as informações fornecidas para acessar a plataforma pela tela de *login* (Figura 12).

Após realizar o login com sucesso o usuário é redirecionado para a página principal (Figura 8, 9 e 10). A princípio esta página não irá constar dados como no exemplo fornecido. Para adicionar seus dados sobre os gastos realizados, primeiro o usuário precisa baixar a fatura de seu cartão em formato CSV através do site do *NuBank*. Com este arquivo em mãos, na página principal o usuário clica no botão “Importar Dados” localizado no topo da tela e será redirecionado para a página de importação de dados (Figura 16). Para importar os dados, o usuário pode tanto arrastar os arquivos *CSV* para a caixa tracejada presente na tela, ou clicar na caixa e selecionar os arquivos na janela que aparecer. É possível cancelar a ação, ou, se estiver tudo correto, realizar a importação dos dados para a plataforma clicando no botão verde “Importar Arquivos” em baixo da caixa tracejada.

Com os dados importados para a plataforma, o usuário pode voltar à página principal clicando no botão “Dash Money” ou no botão “Home” presentes no topo esquerdo da tela. Na página principal estão presentes as informações de todos os dados importados, sem nenhum filtro (Figura 8, 9, 10). Caso o usuário deseje, ele pode filtrar as informações por mês selecionando o botão “Selecione um período” localizado no topo direito da página. Para filtrar os dados por categoria, basta selecionar uma das categorias localizadas no lado esquerdo da página.

Na página principal, quando se tem dados na plataforma, é possível obter informações como: o total de compras realizadas; média de valor por compra; média de valor gasto por dia; total gasto por dia (gráfico de linha no qual é possível obter

mais detalhes sobre a data e o valor total gasto em um dia ao passar o mouse por cima do gráfico); custo médio por dia da semana (gráfico de barras vertical no qual o dia da semana com maior valor é destacado); compras mais caras (lista com as três compras mais caras e informações do valor, local, data e categoria dos gastos); compras mais frequentes (lista com os três locais mais frequentados com a quantidade de vezes que se realizou compra no estabelecimento e a categoria); total gasto por mês (gráfico de barras horizontal constando os meses e o valor total gasto em cada um deles, dando destaque para o mês de maior valor); custo total por categoria (gráfico de barras horizontal com o valor total gasto em todas as categorias com valor acima de zero, ordenado em ordem decrescente e dando destaque à categoria de maior valor).

No topo da página, está presente o botão “@username” com o respectivo *username* do usuário. Este botão redireciona para a página com as informações do usuário (Figura 17). Atualmente não é possível que o usuário altere estas informações manualmente pela plataforma. Para finalizar a sessão basta clicar no botão “Logout” localizado no topo direito da página e o usuário será redirecionado de volta para a tela inicial (Figura 11).

6 CONCLUSÃO

Conforme definido na introdução, este projeto teve como objetivo desenvolver uma aplicação *web* para auxiliar pessoas a compreenderem melhor seus gastos através da aplicação dos princípios de *Data Visualization*. Para a realização deste projeto foi necessário estudar os princípios de *Data Visualization*, assim como quais as melhores representações gráficas para se utilizar em cada caso, sua composição e o que é melhor evitar independente da mensagem a ser transmitida.

Neste projeto foi possível colocar em prática conceitos de desenvolvimento de aplicações web, planejamento, definição da arquitetura e estratégias para lidar com dados. Ele possibilitou um ambiente para descoberta e aprendizado de ferramentas incríveis como *Heroku*, *Auth0* e *Postman*, conhecidas e utilizadas mundialmente por milhares de desenvolvedores.

O principal desafio foi colocar em prática *Data Visualization* para uma situação generalizada como um *dashboard* de gastos pessoais. Em um relatório é possível observar os resultados obtidos nos gráficos e escolher um título ou uma composição que resuma bem o que aconteceu naquele contexto. Já na aplicação apresentada foi necessário escolher títulos mais abrangentes que não focam no resultado dos dados (se os gastos estão aumentando ou diminuindo), mas sim no contexto a que eles pertencem (variação por dia, média por dia da semana, ou total por mês).

Uma sugestão de melhoria seria aplicar *insights* sobre o dado de forma automática utilizando Inteligência Artificial para os títulos dos gráficos. Também seria possível abranger a quantidade de cartões de dados, mostrando as alterações e os gastos diários por outras perspectivas. Com uma base vasta de dados, abriria oportunidade para reconhecimento de padrões e detecção de anomalias sobre as contas realizadas.

Concluiu-se, portanto, que o projeto proposto atingiu seus principais objetivos e espera-se que este trabalho possa fornecer ajuda às pessoas que desejam compreender melhor seus gastos pessoais.

REFERÊNCIAS

The Atlas Team, Database as a Service (DBaaS): What is Database-as-a-Service (DBaaS)?, 2017. Disponível em: <https://www.ibm.com/developerworks/community/blogs/8f058ee2-f3aa-4976-aeb8-4e6102dc86f8/entry/what-is-database-as-a-service-dbaas?lang=en>. Acesso em: 9 de outubro de 2018.

Auth0, Never Compromise on Identity, c2018. Disponível em: <https://auth0.com/>. Acesso em: 12 de agosto de 2018.

AWS Amazon, O que é banco de dados relacional?, c2018. Disponível em: <https://aws.amazon.com/pt/relational-database/>. Acesso em: 9 de outubro de 2018.

BERINATO, S. (2016). Good Charts: The HBR Guide to Making Smarter, More Persuasive Data Visualizations. Harvard Business Review Press. Paginação irregular.

Bootstrap, About, c2018. Disponível em: <https://getbootstrap.com/docs/3.3/about/>. Acesso em: 9 de outubro de 2018.

Bootstrap, Bootstrap, c2018. Disponível em: <https://getbootstrap.com/>. Acesso em: 12 de agosto de 2018.

Chart.js, Chart.js, c2018. Disponível em: <http://www.chartjs.org/>. Acesso em: 12 de Agosto de 2018.

CHOI, Eunmi; VARMA, Mohan Krishna. COMPARATIVE STUDY OF VARIOUS PLATFORM AS A SERVICE FRAMEWORKS, 2016. Disponível em: <http://www.aircconline.com/ijccsa/V6N1/6116ijccsa03.pdf>. Acesso em: 10 de outubro de 2018.

Crunchbase, Auth0, c2018. Disponível em: <https://www.crunchbase.com/organization/auth0#section-overview>. Acesso em: 9 de outubro de 2018.

DUBEY, Abhijit; WAGLE, Dilip. Delivering software as a service, 2007. Disponível em: <http://www.pocsolutions.net/Delivering_software_as_a_service.pdf>. Acesso em: 9 de outubro de 2018.

Express, Fast, unopinionated, minimalist web framework for Node.js, c2018. Disponível em: <https://expressjs.com/>. Acesso em: 12 de agosto de 2018.

FusionCharts, About Us, c2018. Disponível em: <https://www.fusioncharts.com/about-us>. Acesso em: 9 de outubro de 2018.

GitHub, Built for Developers, c2018. Disponível em: <https://github.com/>. Acesso em: 12 de agosto de 2018.

GitKraken, The legendary Git GUI client for Windows, Mac and Linux, c2018. Disponível em: <https://www.gitkraken.com/>. Acesso em: 12 de agosto de 2018.

GitLab, Auto DevOps, c2018. Disponível em: <https://about.gitlab.com/>. Acesso em: 12 de agosto de 2018.

HAN, Bowei. An Introduction to Serverless and FaaS (Functions as a Service), 2017. Disponível em: <https://medium.com/@BoweiHan/an-introduction-to-serverless-and-faas-functions-as-a-service-fb5cec0417b2>. Acesso em: 9 de outubro de 2018.

HELLER, Martin. What is nodejs? The JavaScript runtime explained, 2017. Disponível em: <https://www.infoworld.com/article/3210589/node-js/what-is-nodejs-javascript-runtime-explained.html>. Acesso em: 9 de outubro de 2018.

Heroku, Cloud Application Plataform, c2018. Disponível em: <https://www.heroku.com/>. Acesso em: 12 de agosto de 2018.

Highcharts, Make your data come alive, c2018. Disponível em: <https://www.highcharts.com/>. Acesso em: 12 de agosto de 2018.

JOHNSTON, Paul. Serverless: It's much much more than FaaS, 2018. Disponível em: <https://medium.com/@PaulDJohnston/serverless-its-much-much-more-than-faas-a342541b982e>. Acesso em: 9 de outubro de 2018.

JSON, Introducing JSON, c2018. Disponível em: <https://www.json.org/>. Acesso em: 12 de agosto de 2018.

KNAFLIC, C. N. (2015). Storytelling with Data: A Data Visualization Guide for Business Professionals. John Wiley & Sons. Paginação irregular.

MELL, Peter; GRACE, Timothy. The NIST Definition of Cloud Computing, 2011. Disponível em: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>. Acesso em: 9 de outubro de 2018.

mLab, Trusted. Loved. Most widely deployed, c2018. Disponível em: <https://mlab.com/>. Acesso em: 12 Ago 2018.

MongoDB, MongoDB Atlas Database as a Service, c2018. Disponível em: <https://www.mongodb.com/>. Acesso em: 12 de agosto de 2018.

MongoDB, MongoDB Strengthens Global Cloud Database with Acquisition of mLab, 2018. Disponível em: <https://www.mongodb.com/press/mongodb-strengthens-global-cloud-database-with-acquisition-of-mlab>. Acesso em: 10 de outubro de 2018.

MongoDB, NoSQL Databases Explained: What is NoSQL?, c2018. Disponível em: <https://www.mongodb.com/nosql-explained>. Acesso em: 9 de outubro de 2018.

Node.js, Node.js, c2018. Disponível em: <https://nodejs.org/en/>. Acesso em: 12 de agosto de 2018.

Nubank, Olá, liberdade, c2018. Disponível em: <https://www.nubank.com.br>. Acesso em: 12 de agosto de 2018.

OFOEGBU, Victor. The only NodeJs introduction you'll ever need, 2018. Disponível em: <https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219>. Acesso em: 9 de outubro de 2018.

OKAWA, Jeff. How to choose the best Authentication as a Service Provider for your company, 2018. Disponível em: <https://medium.freecodecamp.org/evaluating-authentication-as-a-service-providers-6903895a8450>. Acesso em: 9 de outubro de 2018.

Papa Parse, The powerful, in-browser CSV parser for big boys and girls, c2018. Disponível em: <https://www.papaparse.com/>. Acesso em: 12 de agosto de 2018.

PARK, C. (20-[?]). Data Visualization for Effective Analytics Comminucation (English Edition). eBook Kindle. Paginação irregular.

Postman, Postman Makes API Development Simple, 2018. Disponível em: <https://www.getpostman.com/>. Acesso em: 12 de agosto de 2018.

REST API Tutorial, What is REST, c2018. Disponível em: <https://restfulapi.net/>. Acesso em: 12 de agosto de 2018.

ROUSE, Margaret. Infrastructure as a Service (IaaS), c2018. Disponível em: <https://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS>. Acesso em: 9 de outubro de 2018.

ROUSE, Margaret. NoSQL (Not Only SQL database), c2018. Disponível em: <https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL>. Acesso em: 9 de outubro de 2018.

ROUSE, Margaret. Platform as a Service (PaaS), c2018. Disponível em: <https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS>. Acesso em: 9 de outubro de 2018.

ROUSE, Margaret. Relational database, c2018. Disponível em: <https://searchdatamanagement.techtarget.com/definition/relational-database>. Acesso em: 9 de outrubro de 2018.

SHULMAN, Will. mLab is becoming a part of MongoDB, Inc., 2018. Disponível em: <https://blog.mlab.com/2018/10/mlab-is-becoming-a-part-of-mongodb-inc/>. Acesso em: 10 de outubro de 2018.

Tableau, Data visualization beginner's guide: a definition, examples, and learning resources, c2018. Disponível em: <https://www.tableau.com/learn/articles/data-visualization#PwDTJKDC6bIdOxs1.99>. Acesso em: 9 de outubro de 2018.

Vue.js, The Progressive JavaScript Framework, c2018. Disponível em: <https://vuejs.org/>. Acesso em: 12 de agosto de 2018.

YAU, N. (2013). Data Points: Visualization That Means Something. John Wiley & Sons. Paginação irregular.