Associação Propagadora Esdeva
Centro de Ensino Superior de Juiz de Fora – CES/JF
Curso de Engenharia de Software
Projeto de Iniciação Científica

TEMPLATES PARA A CONSTRUÇÃO DE APLICAÇÕES MÓVEIS

Davi Queiroz Barreira Pena[1]
Centro Universitário Academia, Juiz de Fora, MG
Romualdo Monteiro de Resende Costa[2]
Centro Universitário Academia, Juiz de Fora, MG

RESUMO

Este trabalho apresenta um estudo que tem por objetivo facilitar o desenvolvimento de aplicações móveis, que são aquelas construídas para serem executadas em dispositivos sem fio e com acesso à Internet, como celulares e *tablets*. Essas aplicações são, usualmente, diferentes das aplicações tradicionais para computadores, pois o ambiente de execução, formado por dispositivos móveis, apresenta desafios adicionais como, por exemplo, o compartilhamento de informações entre os usuários, bem como diferentes interfaces de acesso aos dispositivos. Assim, este trabalho busca levantar e construir componentes que possam facilitar o desenvolvimento de aplicações móveis e, particularmente, serem reusados, como templates, para a construção de novas aplicações.

Palavras-chave: Aplicações móveis. Componentes. Templates. React Native.

1 INTRODUÇÃO

A computação móvel é caracterizada pelo uso de dispositivos móveis, principalmente celulares, onde os usuários, através das aplicações, podem se comunicar, sobretudo, através da Internet. No contexto da computação móvel, as aplicações tendem a ser diferentes daquelas tradicionais, executadas nos computadores, pois os usuários podem acessar os seus serviços em qualquer lugar e a qualquer momento (WEISER, 1991). Além disso, dispositivos móveis, como celulares, são diferentes dos computadores, principalmente quando são

[1] Discente do Curso de Engenharia de Software, do Centro Universitário Academia.
[2] Docente do Curso de Engenharia de Software, do Centro Universitário Academia. Orientador.

consideradas as interfaces existentes nesses dispositivos. Se, por um lado, celulares não dispõem, usualmente, de interfaces populares, como teclados físicos e mouses, por outro lado, os dispositivos móveis podem possuir outras interfaces, como sensores e câmeras.

Visando aproveitar a maior possibilidade de interação que a computação móvel proporciona, diversas aplicações são desenvolvidas com foco na participação do usuário (KANHERE, 2013). Nesse tipo de aplicação, os dados consumidos pelos usuários são, em grande parte, produzidos por outros usuários. É possível, que nessas aplicações, nem todos os dados produzidos sejam compartilhados. Também existe a possibilidade de que o usuário defina quais dados serão compartilhados e que essa definição possa ser alterada ao longo do uso da aplicação. Essa última possibilidade é particularmente útil considerando que, embora o acesso à rede seja uma característica da computação móvel, esse acesso não necessita, e muitas vezes não é, persistente.

O conteúdo produzido pelas aplicações móveis pode ser armazenado localmente, no próprio dispositivo móvel. Essa forma de armazenamento tem como principal vantagem o fato de não demandar uma conexão à rede persistente sendo, portanto, mais simples e, muitas vezes, mais rápida. Por outro lado, essa solução não atende a muitas aplicações que, como mencionado, são baseadas na participação do usuário. Na verdade, tanto para realizar o armazenamento local dos dados, mas, principalmente, quando os dados são armazenados remotamente existe um grande número de soluções que podem ser adotadas.

Quando os dados são armazenados localmente, por exemplo, poderiam ser utilizadas bases relacionais (NAVATHE, 2000) ou o próprio sistema de arquivos (SILBERSCHATZ, 1998). No armazenamento remoto, por outro lado, poderiam ser utilizadas catálogos de documentos, bases de registro e valor ou, até mesmo, bases relacionais exigindo, neste caso, uma conexão persistente a fim de preservar a integridade dos dados, entre outras muitas soluções (BARBARA, 1999). Se a opção pelo armazenamento combinar o armazenamento local com o remoto, o número de possíveis soluções aumenta, com a combinação dos diferentes tipos.

Assim, para o desenvolvimento das aplicações móveis, pode ser interessante que exista um componente que abstraia a forma como os dados são armazenados e acessados. Como mencionado, a grande quantidade de soluções para essa tarefa pode tornar o desenvolvimento difícil, principalmente porque, usualmente, não existe

uma solução de armazenamento ideal. Na verdade, ainda que seja escolhida uma solução de armazenamento adequada, mudanças nos requisitos da aplicação ou ainda evoluções técnicas nas estratégias de armazenamento podem tornar a solução escolhida ultrapassada.

Assim, este trabalho propõe, inicialmente, a construção de um componente para abstrair a tarefa de armazenamento das aplicações móveis e que permita que o armazenamento seja realizado, localmente ou remotamente, de acordo com a escolha do desenvolvedor. Dessa forma, o desenvolvedor pode concentrar-se nos requisitos específicos da aplicação, enquanto a tarefa de armazenamento é abstraída pelo componente mencionado. Evidentemente, é possível que uma aplicação possua necessidades particulares em relação ao armazenamento. Nesse caso, o componente pode ser estendido, servindo como template, para a construção de um novo modelo de armazenamento.

Para o desenvolvimento de uma aplicação móvel, outros componentes podem ser empregados. Assim, o componente relacionado ao armazenamento, mencionado no parágrafo anterior, é apenas um exemplo, proposto neste trabalho a partir do levantamento das principais necessidades das aplicações móveis. Dessa forma, outros componentes podem ser propostos e, para, exemplificar, este trabalho também propõe a construção de um outro componente, um leitor de *QR-Code* (LIU, 2008).

QR-Code é um código bidimensional que, por meio de leitura óptica, pode ser decodificado por *scanners* ou pela forma mais popular conhecida hoje em dia, os celulares, através de suas câmeras. Esse código pode facilitar a entrada de dados nas aplicações evitando, por exemplo, que o usuário precise inserir informações nas aplicações, o que pode ser uma tarefa difícil considerando as limitações existentes nas interfaces dos dispositivos móveis de maneira geral.

As soluções de leitura e acesso ao valor representado no *QR-Code* também podem ter variações como quando é necessária a recuperação de alguns dados, ou quando o código apresenta partes danificadas ou com falhas. Dessa forma, a existência de um componente para essa leitura abstrai para as aplicações os detalhes existentes, facilitando o desenvolvimento e, além disso, eventuais futuras mudanças na tecnologia empregada podem ser absorvidas por modificações no componente, preservando a aplicação da necessidade de refatoração.

Para a sua implementação, este trabalho utiliza uma biblioteca *JavaScript* (FLANAGAN, 2011) para a construção dos componentes e eventuais templates conhecida como *React Native* (EISENMAN, 2018). Essa biblioteca herda o comportamento definido pela biblioteca *React* (BANKS, 2017) mas é voltada para o desenvolvimento de aplicativos móveis, principalmente em razão dos componentes visuais que ela especifica. Dessa forma, *React Native* já é baseada em componentes, o que favorece e justifica a escolha dessa biblioteca para o desenvolvimento deste trabalho. A próxima seção apresenta algumas características dessa biblioteca importantes para o desenvolvimento deste trabalho.

A seguir, a terceira seção apresenta uma aplicação de controle financeiro que serve como ambiente de desenvolvimento dos componentes mencionados. Nessa aplicação, os dados relativos aos gastos pessoais do usuário são armazenados usando o componente proposto nesta introdução. Também nessa aplicação, os valores gastos em bens de consumo podem ser inseridos através da leitura da nota fiscal eletrônica (NFC-E, 2020) usando o componente leitor de *QR-Code* já mencionado. A quarta seção apresenta os resultados alcançados com discussões correlatas e é seguida pela última seção, onde são apresentadas as conclusões.

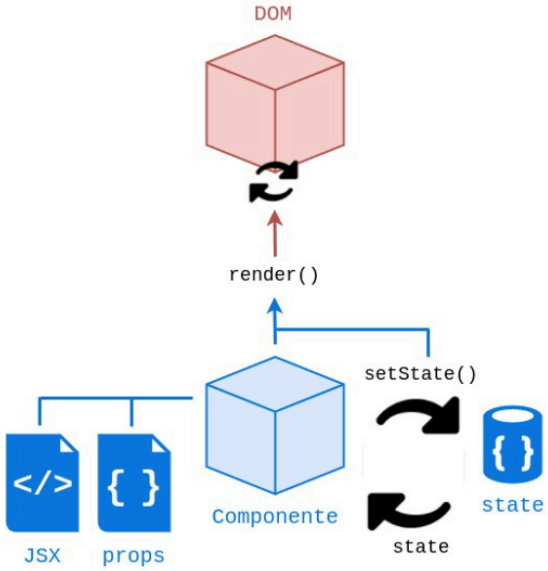
2 REFERENCIAL TEÓRICO

2.1 React Native

React Native é uma biblioteca para construção de aplicativos móveis baseada no *framework* definido originalmente pela biblioteca *React*. Assim, as especificações básicas do *React Native* são as mesmas definidas no *React*, isto é, as aplicações são escritas de forma declarativa, seja usando as especificações funcionais do *Javascript*, ou usando as especificações declarativas do *JSX*[3]. De forma similar, o ciclo de vida das aplicações é o mesmo, isto é, o *framework* para a construção das aplicações permanece inalterado. A Figura 1 apresenta a representação básica desse framework, com destaque para seus elementos. Cabe destacar que a atualização dos componentes, no caso do *React Native*, é realizada diretamente na plataforma de execução, como será visto adiante.

[3] https://facebook.github.io/jsx

FIGURA 1: Elementos de um Componente *React Native* definidos pelo Framework



Fonte: PONTES, 2018.

Na Figura 1, o método *render* aparece em destaque. Esse é o único método obrigatório em todo componente. Quando o componente é construído, esse método é chamado e o seu conteúdo define a forma de apresentação desse componente. Além de ser executado no processo de construção do componente, esse método também deve ser executado por ocasião de modificações sobre o seu estado. O estado, definido também em destaque na Figura 1 (*state*), representa um conjunto de valores que, toda vez que são alterados, provocam novamente a execução do método *render*. Para destacar as alterações no estado, um método especial, conhecido como *setState*, normalmente é empregado.

Além do *render*, o ciclo de vida dos componentes do *React Native* é completado pelos métodos apresentados na Tabela 1.

TABELA 1 - Métodos do Ciclo de Vida da Construção de um Componente *React Native*

Método	Contexto
constructor()	*Método invocado quando o componente é montado*
getDerivedStateFromProps()	*Método invocado antes da renderização*
render()	*Método invocado toda vez que o estado muda*
componentDidMount()	*Método invocado depois da renderização*
componentWillUnmount()	*Método invocado quando o componente é removido*

Fonte: do autor.

O método construtor (*constructor*) é usado para a inicialização do componente. É nesse método que o estado é inicializado. Esse método tem acesso

as propriedades do componente e, se necessário, pode utilizar esses valores para definir seu estado inicial. Logo após a obtenção das propriedades e inicialização do estado, o método *getDerivedStateFromProps* é executado. Esse método é chamado antes da apresentação do componente e pode ser utilizado, por exemplo, para a requisição de conteúdos e configurações adicionais que precisem ser realizadas antes da apresentação. Também é possível, ainda nesse método, realizar modificações sobre o estado. Esse método recebe o estado como um argumento e retorna um objeto com as mudanças no estado.

Após a apresentação do componente, o método *componenteDidMount* é chamado. Esse método pode ser utilizado para fazer requisições de conteúdos, porém, toda vez que o estado for modificado nesse método o ciclo de vida será recomeçado e o componente novamente apresentado. Finalmente, o método *componentWillUnmount* é chamado antes do componente ser removido.

Voltando à Tabela 1, os métodos nela apresentados são chamados em decorrência de ações no ciclo de vida de um componente durante a sua construção. Atualizações nesse componente, por outro lado, decorrentes de mudanças no seu estado ou devido ao recebimento de novas propriedades definem a chamada dos métodos de atualização, que são apresentados na Tabela 2.

TABELA 2. Métodos do Ciclo de Vida da Atualização de um Componente *React Native*

Método	Contexto
getDerivedStateFromProps()	*Método inicial invocado quando o componente é atualizado*
sholdComponentUpdate()	*Método invocado antes da renderização*
render()	*Método invocado toda vez que o estado muda*
getSnapshotBeforeUpdate()	*Método invocado depois da renderização*
componentDidUpdate()	*Método invocado quando o componente é removido*

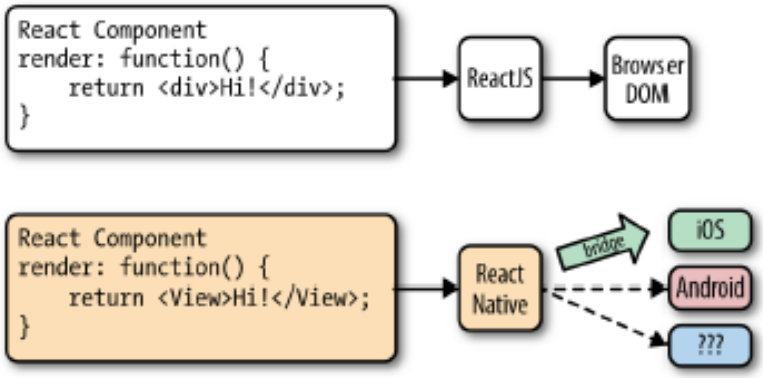
Fonte: do autor.

Quando um componente é atualizado, o primeiro método a ser chamado é o *getDerivedStateFromProps*. Esse método recebe o estado e as propriedades de um componente e retorna as modificações que forem necessárias. Nesse método é possível realizar modificações necessárias para preservar valores no estado de um componente, apesar das modificações realizadas. Após a atualização, mas ainda antes da apresentação, o método *sholdComponentUpdate* é chamado. Esse método, quando usado, pode retornar um valor booleano que especifica se o componente deve ser atualizado, sendo o valor padrão verdadeiro. O método *getSnapshotBeforeUpdate* é utilizado nos casos em que é necessário acessar os

valores das propriedades e do estado de um componente antes da atualização. Assim, através desse método, mesmo após uma atualização, é possível verificar os valores anteriores do estado do componente. Finalmente, o método *componentDidUpdate* é chamado após a apresentação do componente, isto é, depois que o componente foi novamente apresentado a partir de modificações no seu estado.

O método *render*, apresentado nas Tabelas 1 e 2, é o único, entre todos esses métodos, obrigatório na concepção do componente. Esse método, como já mencionado, realiza a apresentação visual do componente. Particularmente, no caso do *React Native*, a apresentação visual é realizada por componentes próprios dessa biblioteca, como representado na Figura 2. Na parte superior dessa figura é representada a apresentação tradicional do *React*, onde os componentes são apresentados no navegador Web. Na parte inferior, por outro lado, é representada a apresentação dos componentes em diferentes plataformas, como acontece no React Native.

FIGURA 2. Comportamento da Apresentação da Aplicação React Native em relação ao React.



Fonte: EISENMAN, 2018.

Como apresentado na Figura 2, as modificações nos componentes da aplicação *React Native*, através de uma ponte (*bridge*, na Figura 2), são traduzidas em modificações para a apresentação da plataforma alvo. Dessa forma, aplicações do *React Native* podem ser apresentadas em qualquer plataforma, desde que seja construída uma interface intermediária que realize a tradução das modificações do componente considerado para as representações na plataforma destino.

Uma vez que as especificações dos componentes no *React Native* podem ser convertidas em apresentações diversas dependendo da plataforma de destino, ao

invés de especificar elementos do HTML para apresentação, como é o caso do *React*, nessa outra biblioteca são definidos elementos com semântica de apresentação própria (EISENMAN, 2018). A Tabela 3 apresenta, como exemplo, alguns desses elementos.

TABELA 3 - Alguns Elementos definidos no *React Native* e sua equivalência com HTML.

Componente	HTML	Objetivo
<View>	* *	*Organizar a disposição dos elementos*
<Text>	**	*Especificar informações textuais*
<FlatList>	* *	*Criar uma lista com informações*
<Image>	*<img>*	*Apresentar uma imagem*
<TextInput>	*<input>*	*Receber entrada de dados dos usuários*

Fonte: do autor.

Para organizar a apresentação dos componentes, o *React Native* define o elemento <*View*>. Esse elemento funciona de maneira semelhante ao <*div*> do HTML permitindo organizar a representação visual dos seus elementos relacionados. Durante a apresentação da aplicação esse elemento deverá ser traduzido para um elemento nativo da plataforma. No caso do sistema operacional Android, por exemplo, esse elemento será traduzido para o elemento homônimo <*View*>. Já no caso do sistema iOS, por outro lado, esse elemento será traduzido para o componente *UIView*.

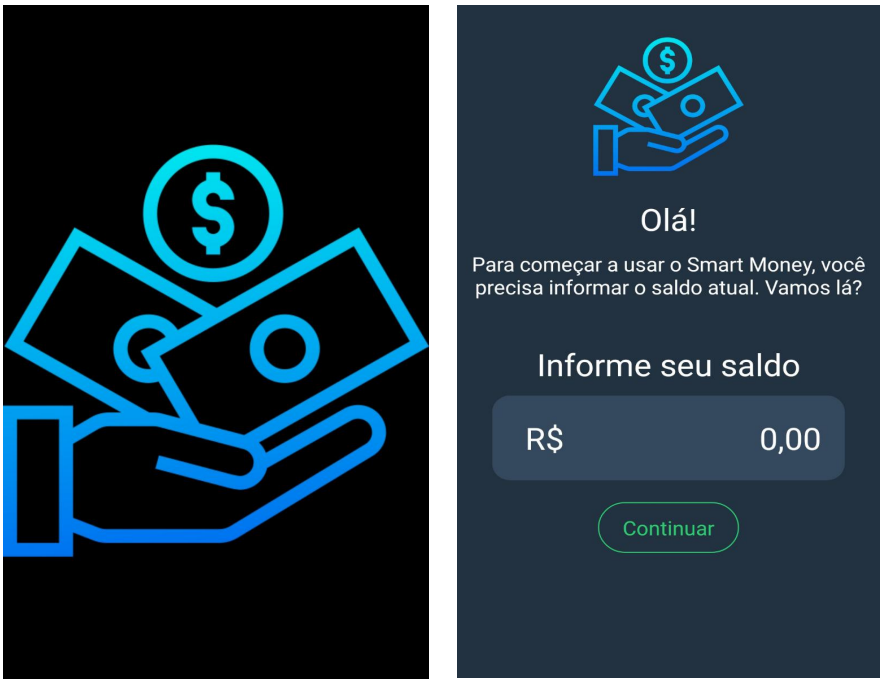
A Tabela 3 apresenta, como exemplo, alguns outros elementos do *React Native*, como o elemento <*Text*> que permite a apresentação de conteúdos textuais, o elemento <*FlatList*> para a construção e apresentação de listas. O elemento <*Image*>, para a apresentação de imagens e o elemento <*TextInput*> que permite ao usuário preencher um campo com informações no formato texto. Diversos outros elementos são oferecidos por essa biblioteca (EISENMAN, 2018).

Cada um dos elementos apresentados, bem como os demais elementos existentes na biblioteca *React Native* são implementados como componentes. Essa biblioteca, portanto, é baseada em componentes visuais. Os componentes propostos neste trabalho, por outro lado, até podem ter elementos visuais mas são, sobretudo, especificados em razão do seu comportamento, como será visto na seção a seguir.

3 METODOLOGIA

A partir da especificação inicial dos componentes e, com o objetivo de avaliar a sua utilização, esses componentes foram desenvolvidos e inseridos em uma aplicação de gestão financeira. Essa aplicação, chamada de *SmartMoney*, foi inicialmente desenvolvida como um trabalho na disciplina de Desenvolvimento para Dispositivos Móveis, nos cursos de Engenharia de Software e Bacharelado em Sistemas de Informação e tem por objetivo facilitar o gerenciamento dos gastos financeiros dos usuários. A Figura 3 apresenta as telas iniciais da aplicação. Após o carregamento, apresentado na primeira tela da Figura 3 o usuário deve inserir o seu saldo inicial na primeira vez que utiliza a aplicação, conforme a segunda tela da Figura 3.

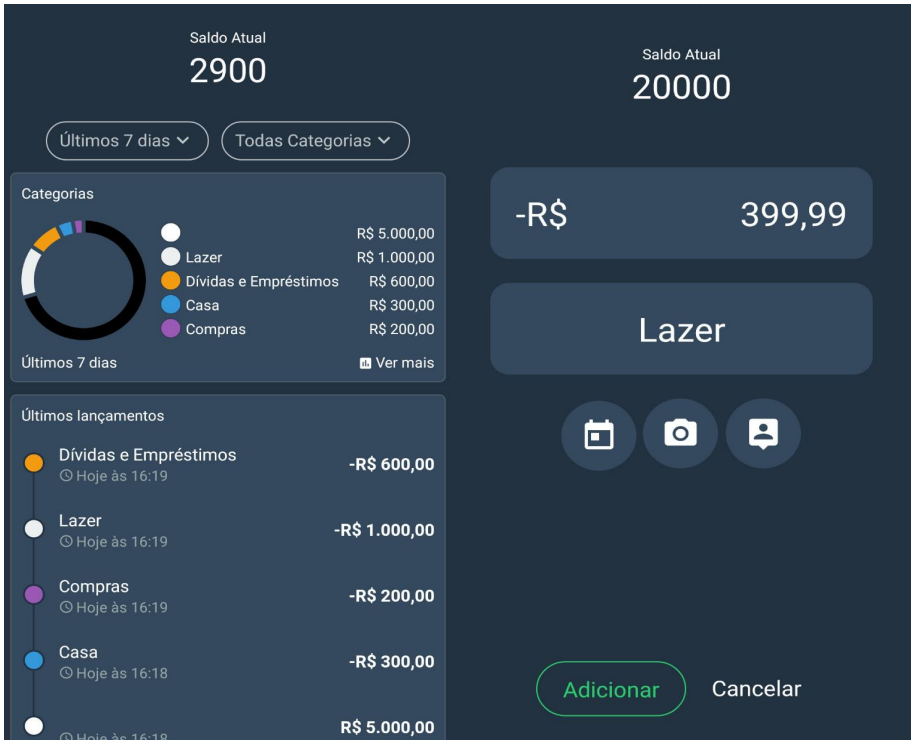
FIGURA 3. Telas iniciais da aplicação *SmartMoney*.



Fonte: do autor.

Quando a aplicação já possui o saldo inicial cadastrado, a tela de gerenciamento, apresentada na primeira parte da Figura 4 é apresentada. Nessa tela o usuário pode visualizar todos os lançamentos, listados pela ordem de cadastro. Também é possível visualizar um gráfico onde as despesas são agrupadas de acordo com o tipo escolhido no momento do lançamento. A segunda tela da Figura 4 apresenta um evento de lançamento. Nessa tela o usuário escolhe o valor, se consiste de um débito ou crédito e o tipo do lançamento que, no caso do exemplo apresentado na Figura 4 consiste em uma despesa relacionada ao “Lazer”.

FIGURA 4: Telas de gestão financeira e de inserção de lançamentos



Fonte: do autor.

Todos os lançamentos realizados na aplicação precisam ser armazenados para que ela funcione adequadamente. Na versão original, esses lançamentos eram realizados localmente, usando o banco de dados Realm[4]. Essa solução, além de não permitir o compartilhamento dos dados, foi implementada com um alto grau de acoplamento entre a aplicação e a base de dados o que dificultava, tanto a instalação da aplicação, que era dependente da instalação do banco, quanto a substituição desse banco de dados por outro que, eventualmente, atendesse melhor as necessidades da aplicação.

Na versão atual, foi adicionado à aplicação um componente que abstrai a tarefa de armazenamento. Esse componente, atualmente, pode ser configurado para realizar o armazenamento local ou na nuvem. Os métodos para o armazenamento são os mesmos, independente do contexto escolhido. Cabe ao desenvolvedor utilizar o arquivo de configuração (*database-configuration.js*) do componente, apresentado na Figura 5 e escolher a opção de armazenamento. No caso do exemplo apresentado na Figura 5 foi escolhida a opção de armazenamento local.

[4] https://girhub.com/realm/realm-js

FIGURA 5: Configuração do Componente de Armazenamento

```
01. export const CONNECTION_TYPES = {LOCAL: 'LOCAL', CLOUD: 'CLOUD'}
02. export const DATABASE_CONNECTION = CONNECTION_TYPES.LOCAL
```

Fonte: do autor.

Na versão atual, os dados podem ser salvos na nuvem usando o *Firebase*[5]. Esse banco, suporta dois modelos de bancos de dados não relacionais: o *Realtime Database* e o *Cloud Firestore*. No primeiro os dados são armazenados em uma estrutura hierárquica de uma árvore, em um formato muito próximo a um documento JSON[6], sendo o formato original do *Firebase*. No segundo, os dados são armazenados em dois níveis, sendo o primeiro o conteúdo e o segundo os documentos. Um conteúdo pode ser formado por diversos documentos, cada um, por sua vez, sendo formado por campos. Os campos definem um tipo e um valor. Documentos podem ser formados por campos diferentes.

Para armazenar os dados localmente, o *AsyncStorage*[7] é empregado. Essa é, inicialmente, uma ferramenta nativa do *React Native* utilizada para armazenar dados persistentes no dispositivo. Essa ferramenta armazena os dados em um modelo de chave, valor, em uma organização semiestruturada bastante parecida com os modelos de armazenamento anteriormente mencionados, particularmente o *Realtime Database*. Nesse caso, o valor a ser armazenado pode ser um objeto, que é convertido, inicialmente para uma *string*. Dessa forma, os dados armazenados assumem a forma de um *vetor* de objetos, similar ao formato do *Realtime Database,* como mencionado.

O modelo de armazenamento semiestruturado onde tipos e valores são armazenados foi definido como base para o componente de armazenamento pois permite armazenar os dados independente da estrutura previamente definida na aplicação. Dessa forma, é esperado que o componente possa ser utilizado em diversas aplicações sem a necessidade de modificações. Caso o desenvolvedor opte por utilizar uma estrutura particular de armazenamento, o componente deverá ser estendido para suportar essa modificação.

[5] https://firebase.google.com
[6] http://www.json.org
[7] https://github.com/react-native-async-storage/async-storage

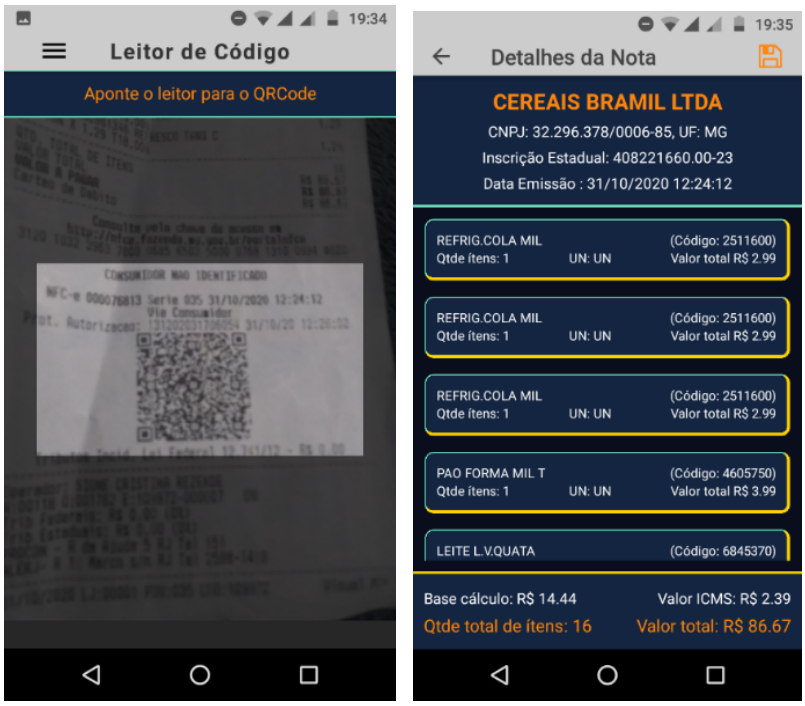
É esperado que o componente de armazenamento seja estendido para, por exemplo, suportar o armazenamento em outras bases de dados. Atualmente, por exemplo, está sendo desenvolvido o armazenamento no banco Postgres[8] usando uma instância remota (nuvem). Nesse caso, a estrutura de armazenamento, do tipo chave, valor, será preservada e, para o desenvolvedor, essa será apenas mais uma opção para o armazenamento dos dados.

Um segundo componente, um leitor de *QR-Code*, conforme mencionado na Introdução, também é construído. Para as aplicações, esse componente pode facilitar a entrada de dados. No caso da gestão financeira, os itens adquiridos podem ser obtidos a partir da leitura da NFC-e (Nota Fiscal de Consumidor Eletrônica), que corresponde a um cupom fiscal emitido e armazenado eletronicamente. Uma NFC-e possui um *QR-Code* que, quando lido, permite o acesso às informações disponibilizadas na Web pelas Secretarias Estaduais da Fazenda. Nesse contexto, é necessário que este componente realize esse acesso, receba e faça a análise das informações para lançamento das despesas na base de dados.

A Figura 6 apresenta as telas referentes à leitura do *QR-Code* e obtenção e gravação dos lançamentos dos itens consumidos. A primeira tela consiste na leitura código onde, para realizar este procedimento, o usuário deve apontar a câmera do dispositivo para o código da NFC-e e, se o código for válido, o componente do aplicativo apresentará a segunda tela, que exibe as informações da nota fiscal obtida na Secretaria Estadual da Fazenda de Minas Gerais. Essas informações podem ser armazenadas se for selecionado o ícone presente no menu superior desta tela.

[8] https://www.postgresql.org

FIGURA 6 — Telas referentes a leitura e gravação da NFC-e.



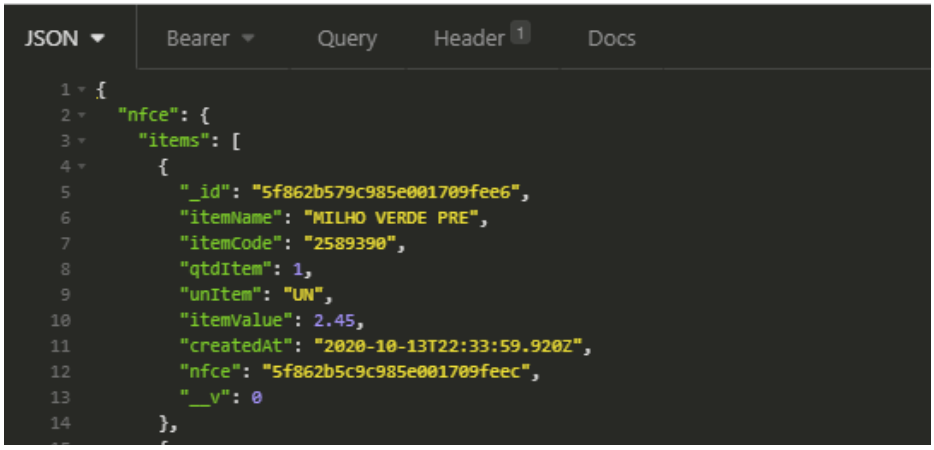
Fonte: Elaborada pelo autor.

Atualmente, o componente obtém os dados apenas da Secretaria Estadual da Fazenda de Minas Gerais mas este componente pode ser estendido para obter informações de outros locais a partir da leitura da NFC-e. Para isso, além do acesso específico ao repositório em questão, é necessário realizar a implementação para a análise dos dados fornecidos.

A Figura 7, apresenta a estrutura das informações obtidas na NFC-e a partir do acesso à Secretaria Estadual da Fazenda de Minas Gerais. Nessa estrutura estão os itens individuais adquiridos (*itemCode*) e os seus valores (*itemValue*), bem como a quantidade (*qtdItem*). Cada item também tem uma representação através de um código hexadecimal único (*_id*), uma descrição (*itemName*) e a unidade de comercialização (*unItem*). A chave da NFC-e onde o item foi consumido também é inserida na especificação do item e corresponde a um valor único especificado em hexadecimal (*nfce*).

Caso outras Secretarias Estaduais definam estruturas diferentes da apresentada na Figura 7, as adaptações que forem necessárias para acesso permanecerão dentro do escopo do componente sem alterar, portanto, a aplicação.

Figura 7 – Estrutura obtidas a partir da Secretaria Estadual da Fazenda de Minas Gerais.



Fonte: Elaborada pelo autor.

4 RESULTADOS E DISCUSSÃO

O processo de desenvolvimento das aplicações móveis é uma questão relevante nos dias atuais, uma vez que, atualmente, essas aplicações fazem parte das tarefas diárias dos usuários e, portanto, existe uma demanda significativa relacionada ao desenvolvimento dessas aplicações. Assim, é importante estabelecer e implementar estratégias que facilitem esse desenvolvimento. Uma possibilidade de aprimorar esse desenvolvimento foi explorada neste trabalho, através da construção de componentes que implementam funções relevantes nas aplicações. Esses componentes podem ser estendidos, o que permite defini-los como templates.

Os componentes construídos, referentes ao armazenamento e à leitura do *QR-Code*, representam uma demonstração da possibilidade dessa abordagem junto às aplicações. Na verdade, a própria identificação, especificação e implementação desses componentes representa uma contribuição importante deste trabalho. Tratam-se de componentes que implementam funcionalidades usuais em aplicações para dispositivos móveis e, assim, é esperado que a utilização desses componentes possa favorecer o desenvolvimento das aplicações.

Para o desenvolvimento, a biblioteca *React Native* permitiu, como esperado, a construção dos componentes. Essa biblioteca já oferece componentes para aplicações móveis e sua estrutura favorece esse tipo de abordagem (EISENMAN, 2018). A proposta deste trabalho, no entanto, define componentes diferentes daqueles oferecidos nessa biblioteca, uma vez que não se tratam de componentes

visuais e sim de componentes que possuem um comportamento complexo, incluindo eventos e o processamento de dados.

Neste trabalho não foi necessário estender os componentes desenvolvidos porque eles foram diretamente utilizados na aplicação que serviu para a sua construção. A própria estrutura de desenvolvimento, baseada nos componentes do *React Native* permite que esses componentes possam ser empregados em outras aplicações e, se necessário, estendidos. Como trabalho futuro, esses componentes serão disponibilizados aos alunos da disciplina de Desenvolvimento para Dispositivos Móveis nos cursos de Engenharia de Software e Bacharelado em Sistemas de Informação e é esperado que, ao menos o componente referente ao armazenamento, possa ser estendido nas aplicações desenvolvidas pelos alunos dessa disciplina.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foram levantados componentes importantes às aplicações móveis, que incluem um componente para o armazenamento e outro para a leitura de *QR-Code* e obtenção dos dados de uma NFC-e. Esses componentes foram implementados e testados em um processo de gestão financeira eletrônica.

A construção desses componentes demonstra que a biblioteca *React Native* é adequada para a construção de componentes mais complexos, além daqueles visuais, podendo ser utilizada na construção de novos componentes.

Como trabalho futuro, além da extensão dos componentes criados, em novas aplicações, novos componentes devem ser construídos, definindo uma biblioteca completa para a construção das aplicações.

ABSCTRACT

This work presents a study that aims to facilitate the development of mobile applications, which are those built to run on wireless devices and with Internet access, such as cell phones and tablets. These applications are usually different from tradicional applications for computers, since the environment, formed by mobile devices, presents additional challenges such as, for example, the sharing of information between users, as well as different interfaces for accessing devices. Thus, this work seeks to raise and build components that can facilitate the

development of mobile applications and, particularly, be reused, as templates for the construction of new applications.

Keywords: Mobile Applications. Components. Templates. React Native.

REFERÊNCIAS

BANKS, A.; PORCELLO, E. **Learning React: Functional Web Development With React and Redux**. ISBN 978-1-491-95462-1. O'Reilly Media. 2017.

BARBARA, D. **Mobile Computing and Databases – A Survey**. IEEE Transactions on Knowledge and Data Engineering, pp 108-117. Janeiro, 1999.

EISENMAN, B. **Learning React Native**. ISBN 978-1-491-98914-2. O'Reilly Media. 2018.

FLANAGAN, D. **JavaScript: The Definitive Guide**. ISBN 978-05-96805-52-4. O'Reilly Media. 2011.

KANHERE, S. S. **Participatory Sensing: Crowdsourcing Data from Mobile Smartphones in Urban Space**. ICDCIT 2013: International Conference on Distributed Computing and Internet Technology, pp 19-26. Janeiro, 2013.

LIU, Y.; YANG, J.; LIU, M. **Recognition of QR Code with mobile phones**. Chinese Control and Decision Conference. Agosto, 2008.

NAVATHE, S. B; ELMASRI, R. **Fundamentals of Database Systems**. ISBN 0-8053-1755-4. Addison-Wesley. 2000.

NFC-E – Secretaria de Estado da Economia. NFC-e – Nota Fiscal do Consumidor Eletrônica, 2019. Disponível em: https://www.economia.go.gov.br/receita-estadual/documentos-fiscais/nfce.html. Acesso em: 28 out. 2020.

SILBERSCHATS, A; GALVIN, P. B. **Operating System Concepts**. ISBN0-201-54262-5. Addison-Wesley. 1998.

WEISER, M. **The Computer for the 21st Century**. Scientific American, pp. 94-104. Setembro, 1991.