13º Congresso Interinstitucional de Iniciação Científica – CIIC 2019
30 e 31 de julho de 2019 – Campinas, São Paulo
ISBN: 978-85-7029-149-3

INOVANDO A CONSTRUÇÃO DE APLICATIVOS AGRÍCOLAS COM PROGRESSIVE WEB APP

Emanuel de Souza **Oliveira**[1], Luciana Alvim Santos **Romani**[2], Sílvio Roberto Medeiros **Evangelista**[3].

Nº 19601

RESUMO *– O desenvolvimento de aplicativos móveis teve um crescimento significativo de 2008 até 2015, apresentando declínio nos últimos anos. O custo de implementação de um aplicativo nativo é elevado, assim como sua complexidade de construção. Como o custo de dados móveis e tempo de download de alguns apps nativos pode ser elevado levando-se em consideração uma utilização de curta duração, um modelo de desenvolvimento poderia ser notado positivamente por uma ausência de instalação no aparelho celular. Uma alternativa eficaz que foi criada para conquistar esse espaço foi a metodologia Progressive Web App (PWA). Este artigo apresenta essa metodologia contemporânea, suas características, motivos pela escolha e o processo de construção de um aplicativo web utilizando o framework React App na linguagem JavaScript, que tem a finalidade de dar acesso às informações do ZARC - Zoneamento Agrícola de Risco Climático. Esse app pretende atender a todo público que necessita de informações a respeito de zoneamento, desde agricultores de pequenas, médias e grandes propriedades até consultores, bancos e seguradoras. O modelo de construção foca na interatividade para o usuário e baixo custo de dados móveis .*

Palavras-chaves: aplicativo móvel, PWA, React App, ZARC, agricultura

[1]Autor, Bolsista CNPq (PIBIC): Graduação em Engenharia de Computação, Unicamp, Campinas-SP; emanueloliveira38@yahoo.com.br
[2]Orientadora: Pesquisadora da Embrapa Informática Agropecuária, Campinas-SP; luciana.romani@embrapa.com
[3]Orientador: Pesquisador da Embrapa Informática Agropecuária, Campinas-SP; silvio.evangelista@embrapa.com

ISBN: 978-85-7029-149-3

ABSTRACT – Mobile application development grew significantly from 2008 to 2015, declining in recent years. The cost of implementing a native application is high, as is its construction complexity. As the cost of mobile data and download time for some native apps may be high considering a short-lived usage, a development model could be positively noticed by a lack of installation on the mobile device. An effective alternative that was created to conquer this space was the Progressive Web App (PWA) methodology. This article presents this contemporary methodology, its characteristics, reasons for the choice and the process of building a web application using the React App framework in the JavaScript language, which has the purpose of giving access to the ZARC - Zoning Information System. This app is intended to serve all audiences who need information about zoning, from farmers of the small, medium and large properties to consultants, banks and insurance companies. The building model focuses on user-friendly and low-cost mobile data.
Keywords: application, PWA, React App, ZARC, agriculture

1 INTRODUÇÃO

No seu sentido mais simples, um *Progressive Web App* é uma aplicação híbrida entre *web* e *mobile*. O *PWA* é uma metodologia de desenvolvimento que torna a experiência de uso de uma página *web* pelo celular semelhante a de um aplicativo *mobile*, fazendo com que soluções digitais apresentem características de aplicações nativas, consumindo menos dados e sem a necessidade de ser feita a instalação (ANTUNES, 2019).

Essa metodologia foi criada pela Google, sendo caracterizada como confiável, rápida e engajante. A confiança refere-se ao seu carregamento instantâneo, mesmo com condições precárias de internet. A rapidez é referente à resposta ao usuário de acordo com sua interação, com animações suaves e sem nenhuma rolagem. Já o engajamento, talvez a característica mais importante que cumpre com o propósito deste artigo, é funcionar permitir sua execução em desktops, laptops e em qualquer dispositivo móvel, independente do seu sistema operacional.

Um *PWA* pode comportar-se de maneira muito parecida a um aplicativo nativo, funcionando *offline*, esperando *push notification* - o *app* é um receptor de notificações do servidor -, permitindo que o usuário adicione um ícone na tela principal do *smartphone*, atualizando automaticamente e possibilitando visualização em tela cheia. Uma vantagem importante que apresenta é reter o público, evitando passo-a-passos tediosos de buscar o *app*, instalar, abrir, se cadastrar, interagir e compartilhar. Além de ser um aplicativo de desenvolvimento mais barato, onde apenas uma plataforma pode ser acessível por qualquer dispositivo.

Para a construção de um aplicativo nesse método, foi utilizado como desafio inicial, a visualização dos dados do Zoneamento Agrícola de Risco Climático (ZARC) publicado em forma de portarias pelo Ministério da Agricultura e Abastecimento (MAPA) e que pode ser visualizado em

ISBN: 978-85-7029-149-3

forma de tabela ou mapa no sistema Agritempo (BAMBINI et at., 2018). A nova proposta de visualização por meio do *PWA*, permite uma interação do usuário que fornece como entrada de dados: a localização, a cultura, o ciclo e o tipo de solo, obtendo como saída os períodos mais indicados para o plantio, informando também o nível de risco, de maneira mais intuitiva. As consultas dos dados do ZARC foram implementadas pelo consumo da API Agritec, disponível na Plataforma AgroAPI da Embrapa (VAZ et al., 2017)).

2 MATERIAL E MÉTODOS

Para a construção desse aplicativo foi necessário instalar *NodeJS* na versão 12.3.0, ou superior. O gerenciador de pacotes *npm*, para linguagem de programação *JavaScript*, vem instalado junto com o *Node*.

O desenvolvimento do aplicativo foi utilizando ambiente virtual *Linux*, com *IDE* de programação *Visual Studio Code* e controle de versão no *Git*.

2.1 Criando um novo aplicativo com *npm*

A inicialização do projeto é feita com as seguintes linhas de comando no terminal *Linux*:

```
npm install -g create-react-app
create-react-app react-app
cd react-app
npm start
```

Figura 1. Comandos de instalação e criação de uma nova aplicação.

Assim que o servidor de desenvolvimento *react* inicia, pode ser que apareça um erro em relação a *webpack-dev-server* e a solução estará escrita no terminal. Essa solução é a melhor, caso funcione, caso contrário é possível criar um arquivo nomeado *.env* na raiz da pasta *react-app*, com a seguinte linha:

```
SKIP_PREFLIGHT_CHECK=true
```

Figura 2. Arquivo .env .

2.2 Componentes

Alguns componentes utilizados nesse projeto buscaram a criação de um *design* interativo e criativo. Os mais importantes para atingir os objetivos foram: *Axios*, *componentDidMount* e *Hooks API*.

ISBN: 978-85-7029-149-3

2.2.1 Axios

Axios é uma função de um cliente *HTTP* baseado em compromissos para o navegador e para o *node.js*. O desenvolvedor faz uma requisição para um link, usando *GET* ou *POST*, e o link retorna alguma resposta. Para deixar a biblioteca à disposição é necessário o seguinte comando no terminal:

```
npm install --save axios
```

Figura 3. Comando de instalação da biblioteca *Axios*.

2.2.2 *ComponentDidMount*()

O *componentDidMount* é um método que é chamado imediatamente após a primeira renderização. Quando é necessário carregar dados de remotamente, este é um bom local para instanciar a solicitação de rede. Esse método aciona uma renderização extra, isso acontece antes que o navegador atualize a tela. Assim, garante que, mesmo que o *render()* seja chamado duas vezes nesse caso, o usuário não veja o estado intermediário.

Esse método serve para atualizar um estado assim que o programa é renderizado, por isso a função *axios*, apresentada anteriormente, é utilizada dentro desse método nesse aplicativo. Esse método não precisa de importação, pois é nativo da linguagem e sua utilização é dada da seguinte maneira:

```
componentDidMount(){
        axios.get('https://api.cnptia.embrapa.br/agritec/v1/culturas', {headers:
{'Authorization': 'Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'}})
          .then(response => {
            this.setState(()=>{
                return {
                    culturas: response.data.data,
                }
          })
        })
    }
```

Figura 4. Importação e utilização da biblioteca *Axios*, dentro do *componentDidMount*.

2.2.3 *Hooks API*

Hooks API é uma função que simplifica a comunicação entre as camadas de componentes. É possível transportar os estados da aplicação entre suas várias camadas abstraindo-se a profundidade.

ISBN: 978-85-7029-149-3

Para a utilização dessa função é necessário implementar o projeto com a biblioteca do *Hooks API*, executando o seguinte comando no terminal:

```
npm install --save react-hooks-global-state
```

Figura 5. Comando de instalação da biblioteca *Hooks API*.

Na sua utilização é possível em apenas um arquivo criar a *Store* - container que armazena e centraliza o estado geral da aplicação -, as *Actions* - fontes de informações que são enviadas da aplicação para a *Store* - e os *Reducers* - recebem das *actions* as informações e tratam-nas para que sejam enviadas para a *Store*. O exemplo a seguir é parte do código do projeto e está na linguagem *TypeScript*:

```
import { createStore } from 'react-hooks-global-state';
type Action =
  |  { type: 'setCiclo'; cicloNome: string; cicloID: string}
  |  { type: 'setTextura'; texturaNome: string; texturaID: string};
export const { GlobalStateProvider, dispatch, useGlobalState } = createStore(
    (state, action: Action) => {
      switch (action.type) {
        case 'setCiclo': return {
            ...state,
            cicloNome: action.cicloNome,
            cicloID: action.cicloID,
        };
        case 'setTextura': return {
            ...state,
            texturaNome: action.texturaNome,
            texturaID: action.texturaID,
        };
        default: return state;
      }
    },
    {
      cicloNome: '',
      cicloID: '',
      texturaNome: '',
      texturaID: '',
    },
);
```

Figura 6. Importação da biblioteca Hooks API, criação da Store e Actions.

Para utilizar a *Store*, suas *actions* e *reducers*, é necessário envolver o componente de nível superior com um *GlobalStateProvider*:

ISBN: 978-85-7029-149-3

```
import { GlobalStateProvider } from './state';
export default class App extends Component {
  render(){
    return (
      <GlobalStateProvider>
        <Router>
          (...)
        </Router>
      </GlobalStateProvider>
    );
  }
}
```

Figura 7. Importação da *Store* e envolvimento do componente superior pelo *GlobalStateProvider*.

Nos componentes sua utilização pode ser feita da seguinte maneira:

```
import { dispatch, useGlobalState } from '../state';
const setTextura = (event) => dispatch({
    texturaNome: event.target.value,
    texturaID: event.target.id,
    type: 'setTextura',
});
export default function Texture(){
  let [texturaID] = useGlobalState('texturaID');
  return(
    <div className='options' onChange={setTextura}>
      (...)
    </div>
  );
}
```

Figura 8. Importação da *Store* e uso dos *Reducers*.

O projeto completo pode ser analisado com mais detalhes no repositório do *github* do autor: *https://github.com/souzolliveira/primeiro-app* e testado como site pelo navegador do dispositivo pelo endereço *https://souzolliveira.github.io/primeiro-app/* .

3 RESULTADOS E DISCUSSÃO

Com o aplicativo todo estruturado, alguns testes de busca foram feitos no intuito de comparar as interfaces gráficas visando sua maior intuitividade. Os resultados abaixo mostram a pesquisa para cidade de Campinas - SP, cultura Milho, ciclo Grupo I e solo tipo Textura Média.

ISBN: 978-85-7029-149-3

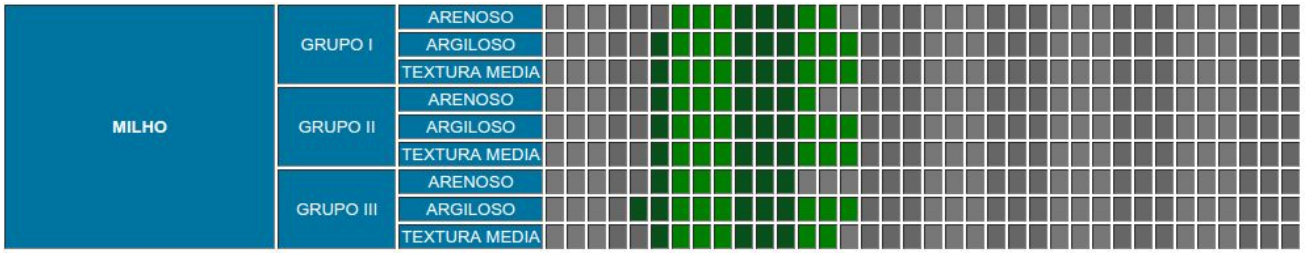

Figura 9. Modelo de visualização dos dados de zoneamento em *www.agritempo.gov.br* .


```
{
  "data": [
    {
      "municipio": "CAMPINAS",
      "uf": "SP",
      "cultura": "MILHO",
      "ciclo": "GRUPO I",
      "solo": "ARENOSO",
      "diaIni": 1,
      "mesIni": 10,
      "diaFim": 10,
      "mesFim": 10,
      "safraIni": 2018,
      "safraFim": 2019,
      "risco": 20
    },
    {
      "municipio": "CAMPINAS",
      "uf": "SP",
      "cultura": "MILHO",
      "ciclo": "GRUPO I",
      "solo": "ARENOSO",
      "diaIni": 11,
      "mesIni": 10,
      "diaFim": 20,
      "mesFim": 10,
      "safraIni": 2018,
      "safraFim": 2019,
      "risco": 20
    },
    {
      "municipio": "CAMPINAS",
      "uf": "SP",
      "cultura": "MILHO",
      "ciclo": "GRUPO I",
      "solo": "ARENOSO",
      "diaIni": 21,
      "mesIni": 10,
      "diaFim": 31,
      "mesFim": 10,
      "safraIni": 2018,
      "safraFim": 2019,
      "risco": 20
    },
```


Figuras 10. a) Modelo de visualização dos dados de zoneamento em *www.agroapi.cnptia.embrapa.br* (esquerda); b) Modelo de visualização dos dados de zoneamento no aplicativo *PWA* desenvolvido (direita).

Pode-se notar que a disposição dos dados no aplicativo facilita a identificação dos decêndios recomendados com diferentes percentuais de risco, a saber 20%, 30% e 40%, visando uma visualização direta de dados de uma forma mais enxuta. Além disso, o *PWA* mostrou-se com

ISBN: 978-85-7029-149-3

um comportamento de aplicativo nativo, preenchendo a tela toda do dispositivo móvel - de sistema operacional *Android* -, que foi uma das vantagens citadas neste artigo no desenvolvimento de um *PWA*.

O aplicativo foi aberto no celular pelo navegador *Google Chrome*, sem necessidade de *download* ou qualquer cadastro. Seu aporte prático e rápido mostra-se muito vantajoso sobre um aplicativo nativo, baixado da *App Store* ou *Play Store*.

Figura 11. Da esquerda para a direita. a) tela para seleção para Localização; b) tela para seleção de Cultura; c) tela para seleção de Ciclo; d) tela para seleção de Solo.

O projeto teve seu início de desenvolvimento em duas frentes: utilizando a linguagem Kotlin e IDE Android Studio; e, utilizando o framework React App com linguagem JavaScript na IDE VSCode. Ambas frentes tinham a linguagem com um conhecimento equivalente pelo autor, porém a estruturação do projeto era desconhecida e foi necessário a leitura de tutoriais e vídeo-aulas para aprender. O escopo utilizado foi o mesmo em ambas, descrito em Oliveira et al., 2018. Assim sendo, o desenvolvimento de do aplicativo nativo em Kotlin teve progresso de 50% em seis meses, enquanto em JavaScript foi completo em quatro meses. Portanto a construção de um modelo PWA evidencia-se mais rápida e de menor complexidade para desenvolver-se.

ISBN: 978-85-7029-149-3

4 CONCLUSÃO

Este artigo demonstrou a construção de um aplicativo utilizando a metodologia *Progressive Web App*, possibilitando uma nova ferramenta no cenário tecnológico da agricultura. Os resultados obtidos revelam que tal método apresenta-se com menor complexidade para o desenvolvimento, mostrando-se mais vantajoso nos quesitos de tempo de aprendizado para desenvolver e adaptação a diferentes equipamentos para visualização.

O uso dessa metodologia é uma forte tendência para o desenvolvimento de aplicativos de rápido acesso e uso de dados reduzido, ocupando uma capacidade de memória relativamente pequena em comparação a um aplicativo nativo. Para aplicativos de uso mais frequente e maior necessidade de *cache*, ainda indica-se o app nativo como uma melhor opção.

Para uma continuação desse trabalho pretende-se fazer o aplicativo acompanhar as atualizações da *AgroAPI* e massificar conceitos de *PWA*, criando artigos e reunindo materiais, para o uso em outras áreas da Embrapa Informática Agropecuária.

5 AGRADECIMENTOS

Ao programa CNPq/PIBIC pela concessão da bolsa de Iniciação Científica, processo Nº 125049/2018-8 para o aluno Emanuel de Souza Oliveira.

6 REFERÊNCIAS

ANTUNES, A. PWA: O que são progressive web apps e por que usar? In: GOBACKLOG, 2019. gobacklog.com/blog/progressive-web-apps/

BAMBINI, M. D.; LUCHIARI JUNIOR, A.; ROMANI, L. A. S.; OTAVIAN, A. F.; KOENIGKAN, L. V.; EVANGELISTA, S. R. M. Manual on-line do sistema Agritempo versão 2.0. Campinas: Embrapa informática Agropecuária, 2015. 70 p. (Embrapa Informática Agropecuária. Documentos, 132).

MAGALHÃES, G. B.; ROMANI, L. A. S. Redesign participativo do aplicativo móvel Agritempo: a importância da interação usuário-desenvolvedor. In: MOSTRA DE ESTAGIÁRIOS E BOLSISTAS DA EMBRAPA INFORMÁTICA AGROPECUÁRIA, 10., 2014, Campinas. Resumos... Brasília, DF: Embrapa, 2014. p. 57-60.

OLIVEIRA, E. de S.; EVANGELISTA, S. R. M.; ROMANI, L. A. S. Aplicativo para consulta interativa e otimizada ao Zoneamento Agrícola de Risco Climático. In: MOSTRA DE ESTAGIÁRIOS E

ISBN: 978-85-7029-149-3

BOLSISTAS DA EMBRAPA INFORMÁTICA AGROPECUÁRIA, 14., 2018, Campinas. Resumos expandidos... Brasília, DF: Embrapa, 2018. p. 63-67.

ROMANI, L. A. S.; MAGALHÃES, G. B.; EVANGELISTA, S. R. M. Desenvolvimento de aplicativos móveis em agricultura: Agritempo mobile. In: CONGRESSO BRASILEIRO DE AGROINFORMÁTICA, 10., 2015, Ponta Grossa. Uso de VANTs e sensores para avanços no agronegócio: anais. Ponta Grossa: Universidade Estadual de Ponta Grossa, 2015. Não paginado. SBIAgro 2015.

VAZ, G. J.; APOLINÁRIO, D. R. de F.; CORREA, J. L.; VACARI, I.; GONZALES, L. E.; DRUCKER, D. P.; BARIANI, J. M.; EVANGELISTA, S. R. M.; ROMANI, L. A. S. AgroAPI: criação de valor para a Agricultura Digital por meio de APIs. In: CONGRESSO BRASILEIRO DE AGROINFORMÁTICA, 11., 2017, Campinas. Ciência de dados na era da agricultura digital: anais. Campinas: Editora da Unicamp: Embrapa Informática Agropecuária, 2017. p. 59-68. SBIAgro 2017.