УДК 61:004

ИНТЕРНЕТ ВЕЩЕЙ В ЗДРАВООХРАНЕНИИ: ОБЗОР ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ

Т. В. Скрыль, А. С. Парамонов
Российский экономический университет им. Г. В. Плеханова, г. Москва
t_skryl@mail.ru, arsenyparamonov@gmail.com

В системе здравоохранения особенно остро стоит вопрос мониторинга пациентов, отслеживания их местоположения и состояния здоровья (мониторинг температуры, давления, сердцебиения и других физических параметров). Благодаря развитию интернета вещей цифровая трансформация в здравоохранении происходит высокими темпами. Авторы статьи ставят перед собой задачу разработать специальное веб-приложение для дистанционного отслеживания сердечного ритма пациентов группы высокого риска с помощью фитнес-браслета (fitness tracker). На основе двух существующих подходов к созданию веб-приложения авторы определяют эффективность его использования в системе здравоохранения для пациентов отдаленной группы высокого риска. Результаты показали, что описанный способ создания нативных гибридных приложений является самым быстрым, идеально подходящим для прототипирования и первоначального запуска проекта, однако для более серьезного и сильно загруженного приложения стоит выбрать React Native. **Выводы.** Данное исследование демонстрирует реализацию IoT и IoMT на основе веб-приложения. Применение любого подхода в здравоохранении должно основываться на поставленных перед разработчиком и медицинским учреждением задачах.

Ключевые слова: цифровая трансформация, медицина, медицинские информационные системы, интернет вещей, IoMT, Javascript, API, WebSocket API, Node JS, React JS, React Native, Create React App.

THE INTERNET OF THINGS IN HEALTHCARE: A REVIEW OF EFFECTIVENESS OF USING WEB APPLICATIONS

T. V. Skryl, A. S. Paramonov
Plekhanov Russian University of Economics, Moscow
t_skryl@mail.ru, arsenyparamonov@gmail.com

Modern medicine has risen to a previously unattainable level over the past decade. Simultaneously one of the main directions in the Internet of things is the development of medicine and public health. Within the health care system, the issue of monitoring patients, tracking their location and health status (monitoring of temperature, pressure, palpitation and other physical parameters) is especially topical. Due to the development of the Internet of things, the digital transformation in health care is expanding at a high rate. The authors focus on the creation of a special web application for distant high-risk group patient's heart-rate tracking with a fitness bracelet (fitness tracker). Based on the existing two main approaches for creating web applications the authors determine the effectiveness of bracelet's use in the health care system for distant high-risk group patients. The results showed that the described method of creating native hybrid applications is the fastest way, perfectly suitable for rapid prototyping and initial launch of the project, however, for a more serious and heavily loaded application, it is worth choosing React Native. **Conclusions.** This study demonstrates the implementation of IoT and IoMT based on a web application. The application of any approach in public health services should be based on the questions posed to the developer and the medical institution.

Keywords: digital transformation, medicine, medical information systems, Internet of things, IoMT, Javascript, API, WebSocket API, Node JS, React JS, React Native, Create React App.

Введение. На данный момент мы являемся свидетелями развития нескольких трендов информационных технологий, потенциально определяющих дальнейший вектор будущего индустрии. Во-первых, область интернета вещей переживает настоящий бум популярности по всему миру (рис. 1), что подтверждается целым рядом исследований. Например, в сентябре 2017 г. компания Hewlett Packard Enterprise (HPE) опубликовала результаты опроса ИТ-специалистов из 20 стран мира, согласно которым по состоянию на 2017 г. примерно 57 % компаний-респондентов уже тем или иным образом внедрили различные IoT-технологии, при этом подавляющее большинство из них – 88 % – уже оценили позитивный экономический эффект от внедрения [1]. Согласно прогнозу консалтинговой компании IDC, достигнутый в отрасли прогресс – лишь начало глобальной цифровой трансформации. По результатам их исследования, уже в 2017 году расходы на внедрение проектов, связанных с IoT-приборами, вырастут на 17 %, а к 2021 году совокупные отметки достигнут 1,4 трлн долл. [2].

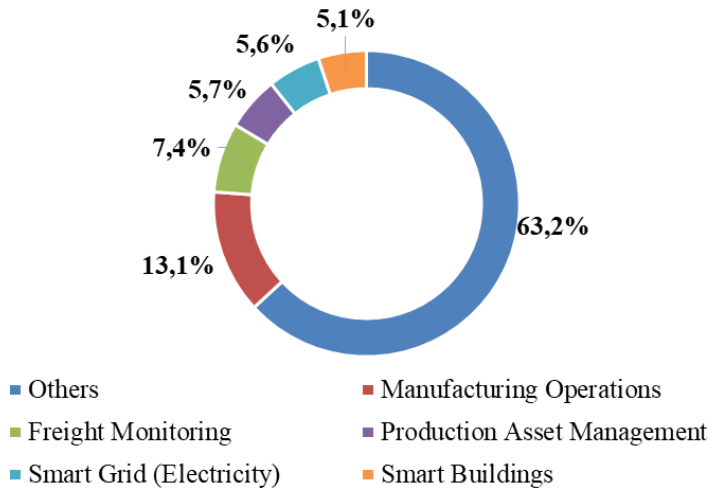

Рис. 1. Основные области применения приборов интернета вещей [3]

Среди наиболее привлекательных инвестиционных статей можно выделить производство оборудования, оказание услуг и разработку программного обеспечения (рис. 2).

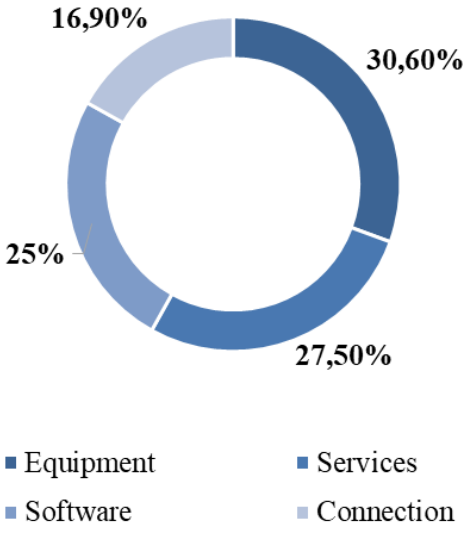

Рис. 2. Структура мирового рынка интернета вещей [3]

Согласно данным, приведенным в отчете компании McKinsey, наибольшую экономическую ценность в денежном эквиваленте составят внедрения IoT в промышленности (рис. 3).

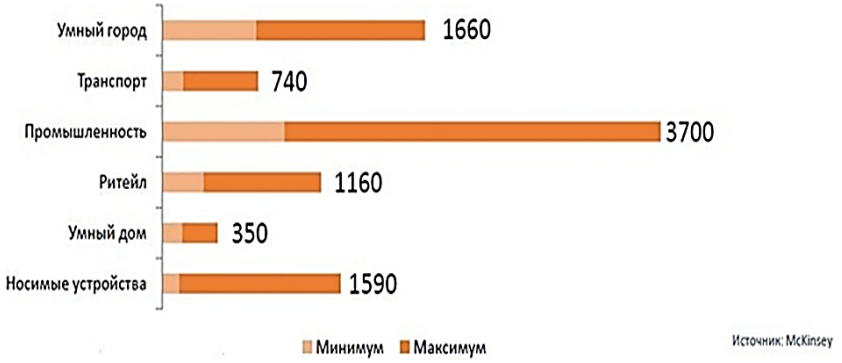

Рис. 3. Объем экономической ценности, создаваемой IoT по отраслям к 2025 г., $ млрд [4]

Общий объем рынка в численном выражении, согласно оценкам Juniper Research, к 2021 г. вырастет в три раза по сравнению с 2016 г. [4], однако такой лавинообразный рост популярности рассматриваемых решений неизбежно повлечет за собой и новые трудности, от степени адаптации к которым зависит успех всей отрасли в целом.

Во-вторых, огромную популярность набирают решения, связанные с отказом от нативных приложений и ориентацией на веб-приложения. Примером этого может быть опыт компании Patagonia, полностью отказавшейся от поддержки приложения в пользу оптимизации работы сайта [5], данной тенденции способствует и рост популярности Javascript (далее – JS) как универсального языка разработки (можем считать это третьим из рассматриваемых трендов): согласно исследованиям порталов Github и Stackoverflow, уже в 2016 г. этот язык вышел на первое место по количеству реализованных проектов и созданных репозиториев [6, 7]. Рассмотрим основные причины такого лавинообразного роста популярности языка программирования, чья сфера применения еще несколько лет назад сводилась лишь к созданию анимации и прочих мелочей.

Основные подходы к созданию веб-приложений

Сообщество разработчиков весьма практично подходит к вопросу использования того или иного языка программирования при разработке продуктов и сервисов – зачастую используется наиболее удобный инструмент, полностью отвечающий требованиям поставленной задачи [8, 9]. Таким образом, стремительное увеличение проектов, написанных на JS, во многом обусловлено появлением серверной реализации языка – Node.js, что в значительной степени унифицирует процесс разработки. В целом можно выделить два взаимодополняющих друг друга подхода к проектированию подобных приложений, которые будут рассмотрены ниже.

Гибридное приложение. Итак, перед нами стоит задача разработать веб-приложение, которое сможет использовать данные, полученные с фитнес-трекера для удаленного мониторинга состояния пациента из группы риска. В общем схема получения данных выглядит так (рис. 4):

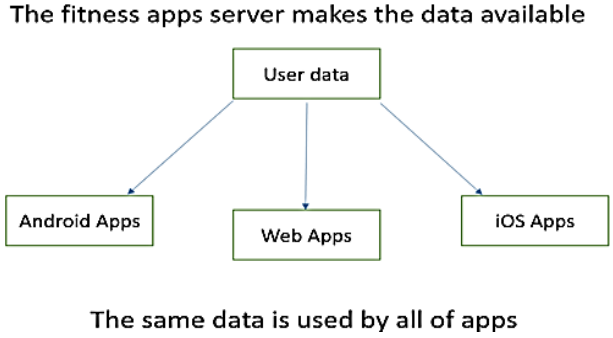

Рис. 4. Схема получения данных с фитнес-трекера [5]

Классический подход подразумевает создание трех различных версий приложения для разных платформ: по одному для Android, iOS и веба. Однако есть в этом нечто иррациональное, от чего в процессе разработки все стараются избавиться в первую очередь – фактическое повторение одного и того же кода, что нарушает один из основных принципов написания качественного кода – DRY (Don't Repeat Yourself). В качестве альтернативы предлагается создание гибридного приложения, что не только лишает необходимости повторять одну и ту же работу три раза, но и значительно экономит время и финансы, необходимые для разработки. Начнем создание веб-приложения с получения информации о частоте пульса с фитнес-трекера и постепенно превратим его в полноценное андроид-приложение.

Одна из основных проблем, встречающихся по мере разработки программного обеспечения, связанного с приборами интернета вещей, отсутствие открытого API для разработчиков с целью получения данных. Тем не менее существуют исключения из этого правила. Так, имеются решения как от гигантов индустрии, представляющих собой своеобразных агрегаторов медицинской информации (Apple HealthKit, Google Fit), так и от частных компаний, занимающихся созданием фитнес-трекеров (FitBit SDK). Рассмотрим пример получения данных через API FitBit, так как именно там все реализовано практически на нативном JS. После регистрации и авторизации на сайте FitBit войдем в панель разработки FitBit SDK, где из шаблона создадим новый проект работы с сенсорами трекера (рис. 5).

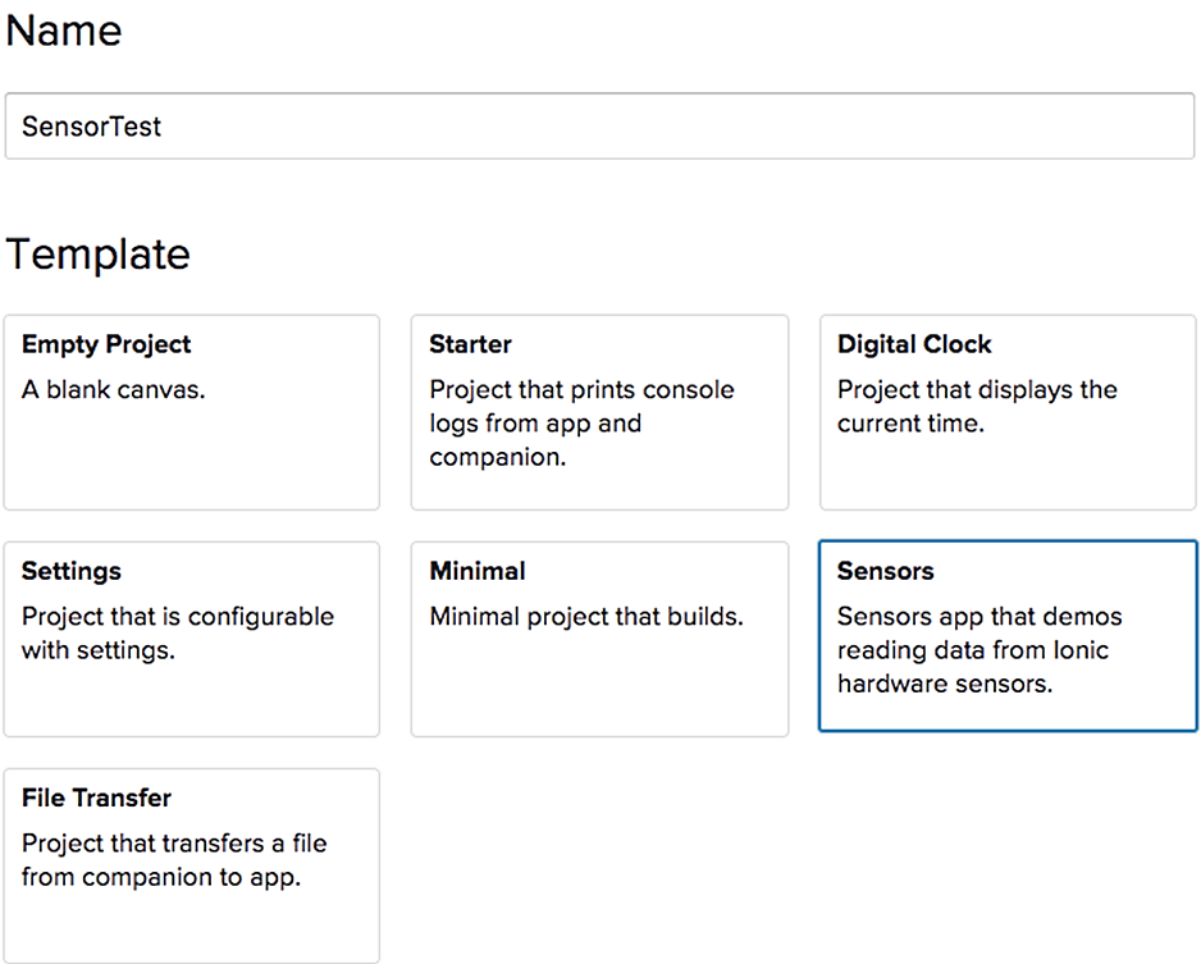

Рис. 5. Создание нового проекта по шаблону в FitBit SDK [6]

В файле index.js шаблона теперь содержится все необходимое для начала работы и получения данных с сенсоров устройства. Следующий код – вывод в консоль текущую частоту пульса владельца фитнес-трекера.

Сниппет кода вывода частоты пульса в консоль:

```
// Импорт модуля сенсора сердцебиения
import { HeartRateSensor } from "heart-rate";
// Создание экземпляра объекта
var hrm = new HeartRateSensor();

// Запуск отслеживания показателей
hrm.start();

hrm.onreading = function getRate() {
var currentRate = hrm.heartRate;

  // Получение текущих показателей частоты пульса
  console.log ("Current heart rate: " + currentRate);
}

//Вызов функции getRate с интервалом в 1000мс
setInterval(getRate, 1000);
```

FitBit API для передачи данных между устройствами использует WebSocket API, так что наше приложение может легко передавать данные на клиент. Предположим, у нас уже есть простенький сайт, основной функционал которого заключается в отображении принимаемых с фитнес-трекера данных. Как мы можем превратить его в гибридное веб-приложение? Для начала необходимо установить Android SDK и создать новый проект. Основой всех гибридных приложений на Android является элемент WebView, отображающий веб-страницы с помощью WebKit, при этом все наши стили и скрипты подключаются и используются как в обычном HTML-файле. Для того чтобы добиться корректного отображения, растянем данный элемент на весь экран в файле лейаута.

Сниппет лейаута элемента WebView:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent" >

  <WebView
  android:id="@+id/WebView"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"   />

</LinearLayout>
```

в файле AndroidManifest.xml разрешим доступ к сети Интернет:

```
<uses-permission android:name="android.permission.INTERNET" />
```

и допишем код класса нашего activity:

```
package com.example.apsoftware;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.webkit.*;
```

```
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       /* разворачиваем приложение на весь экран */
       requestWindowFeature(Window.FEATURE_NO_TITLE);
      getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

    /* применяем наш лейаут к текущему экрану */
      setContentView(R.layout.activity_main);

    /* находим WebView элемент по его id */
     WebView webView = (WebView) findViewById(R.id.WebView);

    /* создаем новые настройки для нашего WebView элемента */
      WebSettings webSettings = webView.getSettings();
       webSettings.setJavaScriptEnabled(true);

       webView.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
       /* здесь помещаем URL сайта */
        webView.loadUrl("http://example.com/");
  }

}
```

Фактически наше гибридное приложение готово, всего лишь несколько доработок позволят превратить его в настоящий инструмент мониторинга состояния пациента как им самим, так и врачом. Среди плюсов данного решения можно назвать универсальность и сокращение необходимых для запуска проекта ресурсов, из минусов – подобное приложение будет работать медленнее нативного, о чем всегда нужно помнить при выборе подхода для реализации по-настоящему высоконагруженных систем. Как можно улучшить подобное решение? Превратить его в изоморфное, процесс создания которого описан ниже.

Изоморфное приложение. В рамках популяризации JS на клиенте и на сервере сформировалась концепция SPA-приложения (Single Page Application), при котором единственный HTML-документ представляет собой оболочку для всех веб-страниц с динамически подгружаемыми стилями и скриптами. Фактически представляет собой нативное приложение, (только не для операционной системы, а для браузера) и нивелирует классические минусы в виде долгой перезагрузки при переходе по страницам, а также во время исполнения запросов на сервер (рис. 6). У данного подхода наблюдались и отрицательные стороны, в частности, подобные сайты крайне плохо индексировались поисковыми системами, что напрямую связано с порядком JS-компиляции контента, не распознаваемого в выдаче поисковика, однако и здесь было найдено соответствующее решение – изоморфные приложения, код в которых способен одновременно выполняться как на клиентской, так и на серверной стороне: при первом обращении компиляция происходит на сервере, и в браузер отдается привычный HTML-документ, при этом после полной загрузки остальные процессы функционируют уже в рамках классического SPA-подхода. Для достижения подобного результата активно используются JS библиотеки и фреймворки – например, React JS, являющаяся ядром такого сайта, как Facebook.

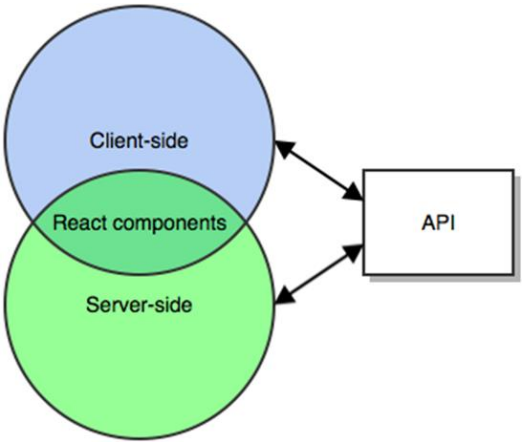

Рис. 6. Схема функционирования изоморфного приложения [7]

Использование данного подхода в значительной степени осложнено сравнительно низким порогом входа на старте и довольно высоким порогом входа для создания более сложных комплексных приложений. Так, для быстрого старта начинающему разработчику, желающему сосредоточиться на бизнес-логике приложения и не тратить время на конфигурирование проекта, достаточно начать использовать созданный Facebook готовый шаблон Create React App. Для этого необходимо иметь лишь менеджер пакетов Node-npm и набрать в консоли:

Сниппет инициирования нового проекта Create React App:

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

Теперь имеем готовый шаблон проекта и можем приступить к написанию кода:

Структура проекта Create-React-App:

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

Итак, предположим, что основной код и бизнес-логика приложения по отслеживанию пульса с использованием FitBit API была описана в файле App.js нашего проекта.

На этом этапе четко разделяем клиентский и серверный рендеринг и имеем две независимые точки входа – client.js и server.js. В файле client.js рендерим компонент App в общем создаваемом контейнере HTML-документа:

Сниппет клиентского рендеринга:

```
import React      from 'react';
import ReactDOM   from 'react-dom';
import App        from 'components/App';

ReactDOM.render(<App />, document.getElementById('react-view'));
```

Для функционирования серверной части необходимо установить фреймворк Express, в значительной степени облегчающий многие процессы разработки. После этого файл server.js будет иметь следующий вид:

Сниппет файла server.js:

```
//Импорт всех необходимых зависимостей
import express  from 'express';
import React    from 'react';
import ReactDom from 'react-dom/server';
import App      from 'components/App';

const app = express();
//Рендеринг компонента по шаблону функции render()
app.use((req, res) => {
  const component = ReactDom.renderToString(<App />);

  return res.end(render(component));
});
      //Шаблон компонента с контейнером react-view
function render(component) {
  return `
    <!DOCTYPE html>
      <html>
      <head>
         <meta charset="utf-8">
         <meta name="viewport" content="width=device-width, initial-scale=1.0">
         <title>Hello</title>
         <link rel="stylesheet" href="myapp/public/assets/styles.css">
      </head>
      <body>
        <div id="react-view">${component}</div>
        <script type="application/javascript" src="myapp/public/assets/bundle.js"></script>
      </body>
    </html>
  `;
}

const PORT = process.env.PORT || 3001;
//Инициализация сервера
app.listen(PORT, () => {
```

```
  console.log(`Server listening on: ${PORT}`);
});
```

Итак, результатом выполнения данного кода станет оптимизация работы сайта, более быстрый отклик и общее повышение надежности в случае наличия ошибок на сервере или в клиенте. Общая схема выглядит следующим образом: сервер практически мгновенно отдает основной контент, в то время как происходит одновременная загрузка клиентского кода. Если этого не происходит, то выполняется обычный запрос к серверу. В зависимости от степени предварительной минификации и оптимизации данный подход позволяет сократить время ожидания примерно на две секунды, что крайне положительно влияет на конверсию коммерческих сайтов, а также автоматически улучшает шансы на более высокое место в выдаче поисковых систем [10].

Отдельно стоит упомянуть тот факт, что в данный момент широкое развитие получила платформа React Native, ориентированная на разработку мобильных приложений [11]. На данный момент технология еще не является зрелой и лишь проходит начальный этап своего формирования, но огромный потенциал заметен уже сейчас: веб-приложения, написанные на React JS, легко могут быть превращены в мобильное кроссплатформенное приложение, однако, по сравнению с гибридным подходом, описанным выше, тут имеются как свои плюсы, так и минусы. Из очевидных плюсов – оптимизированная производительность на любой платформе, из минусов – код все же придется переписывать и модифицировать, так как отличия React JS от React Native достаточно велики. Таким образом, описанный выше метод создания нативных гибридных приложений является наиболее быстрым способом, отлично подходящим для быстрого прототипирования и первоначального запуска проекта, однако для более серьезного и высоконагруженного приложения стоит выбрать React Native, как концептуально более правильную парадигму.

Заключение. В данной статье были рассмотрены основные подходы создания современных веб-приложений мониторинга пульса, которые корректно работают не только в вебе, но и в качестве нативных мобильных приложений. В ходе работы были выделены следующие подходы – классическое гибридное приложение и изоморфное приложение React JS, работающее в связке с Node JS и React Native. Необходимо помнить о том, что применение того или иного подхода в реальной жизни должно основываться на поставленных перед разработчиком и компанией задачах: если в качестве приоритетного направления рассматривается скорейший вывод на рынок веб-приложения с одновременным захватом ниши в области мобильных приложений, то обычное гибридное приложение кажется наиболее оптимальным выбором, так как возможные проблемы с производительностью будут в любом случае устранены с выходом следующей версии (если будет отмечена заинтересованность среди пользователей). Можно сказать, что подходит такой метод и для быстрого прототипирования. В случае постановки более серьезных целей создания высоконагруженного комплексного приложения следует обратить внимание на изоморфные приложения, дополненные отдельной версией React Native, что позволит в значительной степени повысить отказоустойчивость, скорость работы и охват аудитории. Уже сейчас можно с твердой уверенностью сказать, что количество разработок в области интернета вещей будет расти с каждым годом, что делает вышеописанные технологии, позволяющие на совершенно разном уровне работать с подобными проектами, крайне актуальными.

Литература

1. HPE The Internet of Things: Today and Tomorrow, 2017. URL: http://www.arubanetworks.com/assets/eo/HPE_Aruba_IoT_Research_Report.pdf (дата обращения: 11.11.2018).

2. IDC, Worldwide Internet of Things Software Platform Forecast, 2017–2021, 2017. URL: https://www.idc.com (дата обращения: 11.11.2018).

3. IDC, Worldwide Semiannual Internet of Things Spending Guide, 2016. URL: https://www.idc.com (дата обращения: 15.11.2018).

4. Internet of Things (IoT) Market: Global Demand, Growth Analysis & Opportunity Outlook 2023. URL: https://www.researchnester.com/reports (дата обращения: 15.11.2018).

5. Patagonia shuts down its native app, 2016. URL: https://medium.com/stories-behind-the-screens/patagonia-shuts-down-its-native-apps-fe8156d500fe (дата обращения: 15.11.2018).

6. Github 15 most popular languages used on Github by opened Pull Request and percentage change from previous period, 2016. URL: https://octoverse.github.com/ (дата обращения: 23.11.2018).

7. Popular programs technologies of 2017 – study of Stack Overflow, 2017. URL: https://stackoverflow.com/ (дата обращения: 23.11.2018).

8. Osipov V. S., Skryl T. V., Evseev V. O. An analysis of economic issues of territories of priority development // Research Journal of Applied Sciences. 2016. Vol. 11. № 9. P. 833–842. DOI: 10.3923/rjasci.2016.833.842.

9. King C. E., Sarrafzadeh M. A Survey of Smartwatches in Remote Health Monitoring // Journal of Healthcare Informatics Research. 2017. P. 59–66.

10. Osipov V. S., Skryl T. V., Blinova E. A., Kosov M. E., Zeldner A. G., Alekseev A. N. An institutional analysis of public administration system // International Journal of Applied Business and Economic Research. 2017. Vol. 15. № 15. P. 193–203.

11. Скрыль Т. В., Парамонов А. С. Цифровая трансформация сферы здравоохранения: российская и зарубежная специфика // Карел. науч. журн. 2017. Т. 6. № 3 (20). С. 137–140.